Visual Numerics™

IMSL®

Fortran
Subroutines for
Mathematical
Applications

Math/Library

Volumes 1 and 2

Quick Tips on How to Use this Online Manual

Click to display only the page.

Click to display both bookmark and the page.

Double-click to jump to a topic when the bookmarks are displayed.

▽▷ Click to jump to a topic when the bookmarks are displayed.

Click to display both thumbnails and the page.

Click and use to drag the page in vertical direction and to select items on the page.

Click and drag to page to magnify the view.

Click and drag to page to reduce the view.

abc Click and drag to the page to select text.

|◀ Click to go to the first page.

◀ Click to go back to the previous page from which you jumped.

▶ Click to go to the next page.

▶| Click to go to the last page.

◀◀ Click to go back to the previous view and page from which you jumped.

▶▶ Click to return to the next view.

Click to view the page at 100% zoom.

Click to fit the entire page within the window.

Click to fit the page width inside the window.

Click to find part of a word, a complete word, or multiple words in a active document.

Printing an online file: Select **Print** from the **File** menu to print an online file. The dialog box that opens allows you to print full text, range of pages, or selection.

Important Note: The last blank page of each chapter (appearing in the hard copy documentation) has been deleted from the on-line documentation causing a skip in page numbering before the first page of the next chapter, for instance, Chapter 1 in the on-line documentation ends on page 317 and Chapter 2 begins on page 319.

Numbering Pages. When you refer to a page number in the PDF online documentation, be aware that the page number in the PDF online documentation will not match the page number in the original document. A PDF publication always starts on page 1, and supports only one page-numbering sequence per file.

Copying text. Click the abc button and drag to select and copy text.

Viewing Multiple Online Manuals: Select **Open** from the **File** menu, and open the .PDF file you need. Select Cascade from the Window menu to view multiple files.

Resizing the Bookmark Area in Windows: Drag the double-headed arrow that appears on the area's border as you pass over it.

Resizing the Bookmark Area in UNIX: Click and drag the button that appears on the area's border at the bottom of the vertical bar.

Jumping to Topics: Throughout the text of this manual, links to chapters and other sections appear in green color text to indicate that you can jump to them. To return to the page from which you jumped, click the return back icon ◀◀ on the toolbar. ***Note: If you zoomed in or out after jumping to a topic, you will return to the previous zoom view(s) before returning to the page from which you jumped.***

Let's try it, click on the following green color text: **Chapter 1: Linear Systems**

If you clicked on the green color in the example above, Chapter 1: Linear Systems opened. To return to this page, click the ◀◀ on the toolbar.

Visual Numerics, Inc.
Corporate Headquarters
9990 Richmond Avenue, Suite 400
Houston, Texas 77042-4548
USA

PHONE: 713-784-3131
FAX: 713-781-9260
e-mail: marketing@houston.vni.com

Visual Numerics International Ltd.
Centennial Court
Suite 1, North Wing
Easthampstead Road
BRACKNELL
RG12 1YQ
UNITED KINGDOM

PHONE: +44 (0) 1344-311300
FAX: +44 (0) 1344-311377
e-mail: info@vniuk.co.uk

Visual Numerics SARL
Tour Europe
33 Place des Corolles
F-92049 PARIS LA DEFENSE, Cedex
FRANCE

PHONE: +33-1-46-93-94-20
FAX: +33-1-46-93-94-39
e-mail: info@vni.paris.fr

Visual Numerics S. A. de C. V.
Cerrada de Berna #3
Tercer Piso Col. Juarez
Mexico D. F. C. P. 06600
MEXICO

PHONE: +52-5-514-9730 or 9628
FAX: +52-5-514-4873

Visual Numerics International GmbH
Zettachring 10, D-70567
Stuttgart
GERMANY

PHONE: +49-711-13287-0
FAX: +49-711-13287-99
e-mail: vni@visual-numerics.de

Visual Numerics Japan, Inc.
GOBANCHO HIKARI BLDG. 4TH Floor
14 GOBAN-CHO CHIYODA-KU
TOKYO, JAPAN 113

PHONE: +81-3-5211-7760
FAX: +81-3-5211-7769
e-mail: vnijapan@vnij.co.jp

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Road
Taipei, Taiwan 110
ROC

PHONE: (886) 2-727-2255
FAX: (886) 2-727-6798
e-mail: info@vni.com.tw

Visual Numerics Korea, Inc.
HANSHIN BLDG. Room 801
136-1, MAPO-DONG, MAPO-GU
SEOUL, 121-050
KOREA SOUTH

PHONE: +82-2-3273-2632 or 2633
FAX: +82-2-3273-2634
e-mail: leevni@chollian.dacom.co.kr

World Wide Web site: http://www.vni.com

The information contained in this document is subject to change without notice.

VISUAL NUMERICS, INC., MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual Numerics, Inc., shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend

Use, duplication or disclosure by the US Government is subject to restrictions as set forth in FAR 52.227-19, subparagraph (c) (l) (ii) of DOD FAR SUPP 252.227-7013, or the equivalent government clause for agencies.

Restricted Rights Notice: The version of the IMSL Numerical Libraries described in this document is sold under a per-machine license agreement. Its use, duplication, and disclosure are subject to the restrictions on the license agreement.

IMSL Fortran and C
Application Development Tools

Visual Numerics, Inc.
Corporate Headquarters
9990 Richmond Avenue, Suite 400
Houston, Texas 77042-4548
USA

PHONE: 713-784-3131
FAX: 713-781-9260
e-mail: marketing@houston.vni.com

Visual Numerics International Ltd.
Centennial Court
Suite 1, North Wing
Easthampstead Road
BRACKNELL
RG12 1YQ
UNITED KINGDOM

PHONE: +44 (0) 1344-311300
FAX: +44 (0) 1344-311377
e-mail: info@vniuk.co.uk

Visual Numerics SARL
Tour Europe
33 Place des Corolles
F-92049 PARIS LA DEFENSE, Cedex
FRANCE

PHONE: +33-1-46-93-94-20
FAX: +33-1-46-93-94-39
e-mail: info@vni.paris.fr

Visual Numerics S. A. de C. V.
Cerrada de Berna #3
Tercer Piso Col. Juarez
Mexico D. F. C. P. 06600
MEXICO

PHONE: +52-5-514-9730 or 9628
FAX: +52-5-514-4873

Visual Numerics International GmbH
Zettachring 10, D-70567
Stuttgart
GERMANY

PHONE: +49-711-13287-0
FAX: +49-711-13287-99
e-mail: vni@visual-numerics.de

Visual Numerics Japan, Inc.
GOBANCHO HIKARI BLDG. 4^{TH} Floor
14 GOBAN-CHO CHIYODA-KU
TOKYO, JAPAN 113

PHONE: +81-3-5211-7760
FAX: +81-3-5211-7769
e-mail: vnijapan@vnij.co.jp

Visual Numerics, Inc.
7/F, #510, Sect. 5
Chung Hsiao E. Road
Taipei, Taiwan 110
ROC

PHONE: (886) 2-727-2255
FAX: (886) 2-727-6798
e-mail: info@vni.com.tw

Visual Numerics Korea, Inc.
HANSHIN BLDG. Room 801
136-1, MAPO-DONG, MAPO-GU
SEOUL, 121-050
KOREA SOUTH

PHONE: +82-2-3273-2632 or 2633
FAX: +82-2-3273-2634
e-mail: leevni@chollian.dacom.co.kr

World Wide Web site: http://www.vni.com

The information contained in this document is subject to change without notice.

VISUAL NUMERICS, INC., MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual Numerics, Inc., shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend

Use, duplication or disclosure by the US Government is subject to restrictions as set forth in FAR 52.227-19, subparagraph (c) (l) (ii) of DOD FAR SUPP 252.227-7013, or the equivalent government clause for agencies.

Restricted Rights Notice: The version of the IMSL Numerical Libraries described in this document is sold under a per-machine license agreement. Its use, duplication, and disclosure are subject to the restrictions on the license agreement.

IMSL Fortran and C
Application Development Tools

IMSL®

Fortran Subroutines for Mathematical Applications

Math Library

Volumes 1 and 2

Version	Revision History	Year	Part Number
1.0	Original Issue	1984	MALB-USM-PERFCT-1.0
1.1	Fixed bugs and added significant changes to functionality.	1989	MALB-USM-PERFCT-1.0
2.0	Added routines to enhance functionality.	1991	MALB-USM-PERFCT-EN9109-2.0
3.0	Added more routines to improve functionality.	1994	Vol. 1 5371, Vol. 2 5372

Click here to go to F77/Stat/Vol.1/Library

Click here to go to F77/Sfun/ Library

Click here to go to F77/Stat/Vol. 2/Library

Click here to go to F90 MP Library

Contents

Volume I

Click here to go to F77/Stat/ Library

Click here to go to DNFL

Click here to go to F77/SFUN/ Library

Contents

Volumes I & II

Introduction

The IMSL Libraries

The IMSL Libraries consist of two separate but coordinated Libraries that allow easy user access. These Libraries are organized as follows:

- MATH/LIBRARY general applied mathematics and special functions
- STAT/LIBRARY statistics

The *IMSL MATH/LIBRARY User's Manual* has two parts: MATH/LIBRARY and MATH/LIBRARY Special Functions.

Most of the routines are available in both single and double precision versions. The same user interface is found on the many hardware versions that span the range from personal computer to supercomputer. Note that some IMSL routines are not distributed for FORTRAN compiler environments that do not support double precision complex data. The names of the IMSL routines that return or accept the type double complex begin with the letter "Z" and, occasionally, "DC."

Getting Started

The IMSL MATH/LIBRARY is a collection of FORTRAN routines and functions useful in research and mathematical analysis. Each routine is designed and documented to be used in research activities as well as by technical specialists.

To use any of these routines, you must write a program in FORTRAN (or possibly some other language) to call the MATH/LIBRARY routine. Each routine conforms to established conventions in programming and documentation. We give first priority in development to efficient algorithms, clear documentation, and accurate results. The uniform design of the routines makes it easy to use more than one routine in a given application. Also, you will find that the design consistency enables you to apply your experience with one MATH/LIBRARY routine to all other IMSL routines that you use.

Finding the Right Routine

lauds The MATH/LIBRARY is organized into chapters; each chapter contains routines with similar computational or analytical capabilities. To locate the right routine for a given problem, you may use either the table of contents located in each chapter introduction, or the alphabetical list of routines. The GAMS index uses GAMS classification (Boisvert, R.F., S.E. Howe, D.K. Kahaner, and J.L. Springmann 1990, *Guide to Available Mathematical Software*, National Institute of Standards and Technology NISTIR 90-4237). Use the GAMS index to locate which MATH/LIBRARY routines pertain to a particular topic or problem.

Often the quickest way to use the MATH/LIBRARY is to find an example similar to your problem and then to mimic the example. Each routine document has at least one example demonstrating its application. The example for a routine may be created simply for illustration, it may be from a textbook (with reference to the source), or it may be from the mathematical literature.

Organization of the Documentation

This manual contains a concise description of each routine, with at least one demonstrated example of each routine, including sample input and results. You will find all information pertaining to the MATH/LIBRARY in this manual. Moreover, all information pertaining to a particular routine is in one place within a chapter.

Each chapter begins with an introduction followed by a table of contents that lists the routines included in the chapter. Documentation of the routines consists of the following information:

- IMSL Routine Name
- Purpose: a statement of the purpose of the routine
- Usage: the form for referencing the subprogram with arguments listed. There are two usage forms:
  – `CALL sub(argument-list)` for subroutines
  – `fun(argument-list)` for functions
- Arguments: a description of the arguments in the order of their occurrence. Input arguments usually occur first, followed by input/output arguments, with output arguments described last. For functions, the function symbolic name is described after the argument descriptions.

  Input Argument must be initialized; it is not changed by the routine.

  Input/Output Argument must be initialized; the routine returns output through this argument; cannot be a constant or an expression.

  Input or Output Select appropriate option to define the argument as either input or output. See individual routines for further instructions.

  Output No initialization is necessary; cannot be a constant or an expression. The routine returns output through this argument.

- Remarks: details pertaining to code usage and workspace allocation
- Algorithm: a description of the algorithm and references to detailed information. In many cases, other IMSL routines with similar or complementary functions are noted.
- Programming notes: an optional section that contains programming details not covered elsewhere
- Example: at least one application of this routine showing input and required dimension and type statements
- Output: results from the example(s)
- References: periodicals and books with details of algorithm development

Naming Conventions

The names of the routines are mnemonic and unique. Most routines are available in both a single precision and a double precision version, with names of the two versions sharing a common root. The name of the double precision version begins with a "D." The single precision version is generally just the mnemonic root, but sometimes a letter "S" or "A" is used as a prefix. For example, the following pairs are names of routines in the two different precisions: GQRUL/DGQRUL (the root is "GQRUL ," for "Gauss quadrature rule"), RECCF/DRECCF (the root is "RECCF," for "recurrence coefficient"), and SADD/DADD (the root is "ADD"). The names of the IMSL routines that return or accept the type double complex begin with the letter "Z" or, occasionally, "DC."

Except when expressly stated otherwise, the names of the variables in the argument lists follow the FORTRAN default type for integer and floating point. In other words, a variable whose name begins with one of the letters "I" through "N" is of type INTEGER, and otherwise is of type REAL or DOUBLE PRECISION, depending on the precision of the routine.

An array with more than one dimension that is used as a FORTRAN argument can have an assumed-size declarator for the last dimension only. In the MATH/LIBRARY routines, this information is passed by a variable with the prefix "LD" and with the array name as the root. For example, the argument LDA contains the leading dimension of array *A*.

Where appropriate, the same variable name is used consistently throughout a chapter in the MATH/LIBRARY. For example, in the routines for random number generation, NR denotes the number of random numbers to be generated, and R or IR denotes the array that stores the numbers.

When writing programs accessing the MATH/LIBRARY, the user should choose FORTRAN names that do not conflict with names of IMSL subroutines, functions, or named common blocks. The careful user can avoid any conflicts with IMSL names if, in choosing names, the following rules are observed:

- Do not choose a name that appears in the Alphabetical Summary of Routines, at the end of the *User's Manual*.

- Do not choose a name consisting of more than three characters with a numeral in the second or third position.

  For further details, see the section on “Reserved Names” in the Reference Material.

Programming Conventions

In general, the IMSL MATH/LIBRARY codes are written so that computations are not affected by underflow, provided the system (hardware or software) places a zero value in the register. In this case, system error messages indicating underflow should be ignored.

IMSL codes also are written to avoid overflow. A program that produces system error messages indicating overflow should be examined for programming errors such as incorrect input data, mismatch of argument types, or improper dimensioning.

In many cases, the documentation for a routine points out common pitfalls that can lead to failure of the algorithm.

Library routines detect error conditions, classify them as to severity, and treat them accordingly. This error-handling capability provides automatic protection for the user without requiring the user to make any specific provisions for the treatment of error conditions. See the section on “User Errors” in the Reference Material for further details.

Error Handling

The routines in the IMSL MATH/LIBRARY attempt to detect and report errors and invalid input. Errors are classified and are assigned a code number. By default, errors of moderate or worse severity result in messages being automatically printed by the routine. Moreover, errors of worse severity cause program execution to stop. The severity level as well as the general nature of the error is designated by an “error type” with numbers from 0 to 5. An error type 0 is no error; types 1 through 5 are progressively more severe. In most cases, you need not be concerned with our method of handling errors. For those interested, a complete description of the error-handling system is given in the Reference Material, which also describes how you can change the default actions and access the error code numbers.

Work Arrays

A few routines in the IMSL MATH/LIBRARY require work arrays. On most systems, the workspace allocation is handled transparently, but on some systems, workspace is obtained from a large array in a `COMMON` block. On these systems, when you have a very large problem, the default workspace may be too small. The routine will print a message telling you the statements to insert in your program in order to provide the needed space (using the common block WORKSP for integer or real numbers or the common block WKSPCH for characters). The routine will then automatically halt execution. See “Automatic Workspace Allocation” in the Reference Material for details on common block names and default sizes.

For each routine that obtains workspace from the common area, a second routine is available that allows you to provide the workspace explicitly. For example, the routine LSLRG, page 11, uses workspace and automatically allocates the required amount, if available. The routine L2LRG does the same as LSLRG but has a work array in its argument list, which the user must declare to be of appropriate size. The “Automatic Workspace Allocation” section in the Reference Material contains further details on this subject.

Printing Results

Most of the routines in the IMSL MATH/LIBRARY (except the line printer routines and special utility routines) do not print any of the results. The output is returned in FORTRAN variables, and you can print these yourself. See Chapter 10, “Utilities,” for detailed descriptions of these routines.

A commonly used routine in the examples is the IMSL routine UMACH, which (page 1173), retrieves the FORTRAN device unit number for printing the results. Because this routine obtains device unit numbers, it can be used to redirect the input or output. The section on “Machine-Dependent Constants” in the Reference Material contains a description of the routine UMACH.

Chapter 1: Linear Systems

Routines

1.1. Solution of Linear Systems, Matrix Inversion, and Determinant Evaluation

1.1.1 Real General Matrices

1.1.2 Complex General Matrices

1.1.3 Real Triangular Matrices

1.1.4 Complex Triangular Matrices

Usage Notes

Solving Linear Equations

Many of the routines described in this chapter are for matrices with special properties or structure. Computer time and storage requirements for solving systems with coefficient matrices of these types can often be drastically reduced, using the appropriate routine, compared with using a routine for solving a general complex system.

The appropriate matrix property and corresponding routine can be located in the "Routines" section. Many of the linear equation solver routines in this chapter are derived from subroutines from LINPACK, Dongarra et al. (1979). Other routines have been developed by Visual Numerics staff, derived from draft versions of LAPACK subprograms, Bischof et al. (1988), or were obtained from alternate sources.

A system of linear equations is represented by $Ax = b$ where A is the $n \times n$ coefficient data matrix, b is the known right-hand-side n-vector, and x is the unknown or solution n-vector. Figure 1-1 summarizes the relationships among the subroutines. Routine names are in boxes and input/output data are in ovals. The suffix ** in the subroutine names depend on the matrix type. For example, to compute the determinant of *A* use LFC** or LFT** followed by LFD**.

The paths using LSA** or LFI** use iterative refinement for a more accurate solution. The path using LSA** is the same as using LFC** followed by LFI**. The path using LSL** is the same as the path using LFC** followed by LFS**. The matrix inversion routines LIN** are available only for certain matrix types.

Matrix Types

The two letter codes for the form of coefficient matrix, indicated by ** in Figure 1-1, are as follows:

RG	Real general (square) matrix.
CG	Complex general (square) matrix.
TR or CR	Real tridiagonal matrix.
RB	Real band matrix.
TQ or CQ	Complex tridiagonal matrix.
CB	Complex band matrix.
SF	Real symmetric matrix stored in the upper half of a square matrix.
DS	Real symmetric positive definite matrix stored in the upper half of a square matrix.
DH	Complex Hermitian positive definite matrix stored in the upper half of a complex square matrix.
HF	Complex Hermitian matrix stored in the upper half of a complex square matrix.
QS or PB	Real symmetric positive definite band matrix.
QH or QB	Complex Hermitian positive definite band matrix.
XG	Real general sparse matrix.
ZG	Complex general sparse matrix.
XD	Real symmetric positive definite sparse matrix.
ZD	Complex Hermitian positive definite sparse matrix.

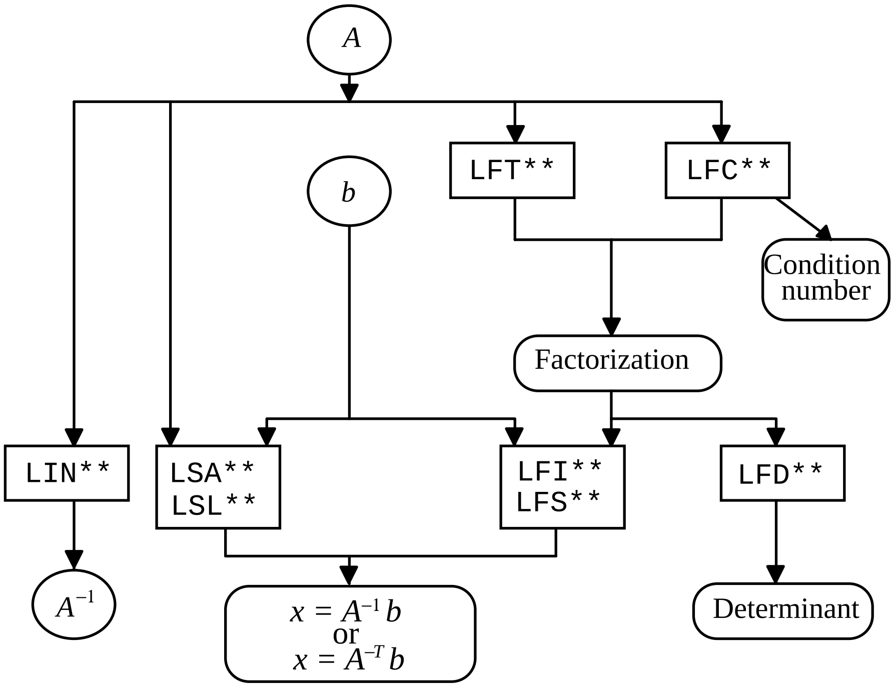


Figure 1-1 Solution and Factorization of Linear Systems

Solution of Linear Systems

The simplest routines to use for solving linear equations are `LSL**` and `LSA**` For example, the mnemonic for matrices of real general form is `RG`. So, the routines `LSLRG` (page 11) and `LSARG` (page 10) are appropriate to use for solving linear systems when the coefficient matrix is of real general form. The routine `LSARG` uses iterative refinement, and more time than `LSLRG`, to determine a high accuracy solution.

The high accuracy solvers provide maximum protection against extraneous computational errors. They do not protect the results from instability in the mathematical approximation. For a more complete discussion of this and other important topics about solving linear equations, see Rice (1983), Stewart (1973), or Golub and van Loan (1989).

Multiple Right Sides

There are situations where the `LSL**` and `LSA**` routines are not appropriate. For example, if the linear system has more than one right-hand-side vector, it is most economical to solve the system by first calling a factoring routine and then calling a solver routine that uses the factors. After the coefficient matrix has been factored, the routine `LFS**` or `LFI**` can be used to solve for one right-hand side at a time. Routines `LFI**` uses iterative refinement to determine a high accuracy solution but requires more computer time and storage than routines `LFS**`.

Determinants

The routines for evaluating determinants are named `LFD**`. As indicated in Figure 1-1, these routines require the factors of the matrix as input. The values of determinants are often badly scaled. Additional complications in structures for evaluating them result from this fact. See Rice (1983) for comments on determinant evaluation.

Iterative Refinement

Iterative refinement can often improve the accuracy of a well-posed numerical solution. The iterative refinement algorithm used is as follows:

```
$x_0 = A^{-1}b$
For $i$ = 1, 50
        $r_i = Ax_{i-1} - b$ computed in higher precision
        $p_i = A^{-1} r_i$
        $x_i = x_{i-1} + p_i$
        if ($\| p_i \|_\infty \le \varepsilon \| x_i \|_\infty$) Exit
End for
Error — Matrix is too ill-conditioned
```

If the matrix A is in single precision, then the residual $r_i = Ax_{i-1} - b$ is computed in double precision. If A is in double precision, then quadruple-precision arithmetic routines are used.

Matrix Inversion

An inverse of the coefficient matrix can be computed directly by one of the routines named `LIN**`. These routines are provided for general matrix forms and some special matrix forms. When they do not exist, or when it is desirable to compute a high accuracy inverse, the two-step technique of calling the factoring routine followed by the solver routine can be used. The inverse is the solution of the matrix system $AX = I$ where I denotes the $n \times n$ identity matrix, and the solution is $X = A^{-1}$.

Singularity

The numerical and mathematical notions of singularity are not the same. A matrix is considered numerically singular if it is sufficiently close to a mathematically singular matrix. If error messages are issued regarding an exact singularity then specific error message level reset actions must be taken to handle the error condition. By default, the routines in this chapter stop. The solvers require that the coefficient matrix be numerically nonsingular. There are some tests to determine if this condition is met. When the matrix is factored, using routines LFC**, the condition number is computed. If the condition number is large compared to the working precision, a warning message is issued and the computations are continued. In this case, the user needs to verify the usability of the output. If the matrix is determined to be mathematically singular, or ill-conditioned, a least-squares routine or the singular value decomposition routine may be used for further analysis.

Special Linear Systems

Toeplitz matrices have entries which are constant along each diagonal, for example:

$$A = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_{-1} & p_0 & p_1 & p_2 \\ p_{-2} & p_{-1} & p_0 & p_1 \\ p_{-3} & p_{-2} & p_{-1} & p_0 \end{bmatrix}$$

Real Toeplitz systems can be solved using `LSLTO`, page 248. Complex Toeplitz systems can be solved using `LSLTC`, page 249.

Circulant matrices have the property that each row is obtained by shifting the row above it one place to the right. Entries that are shifted off at the right reenter at the left. For example:

$$A = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_4 & p_1 & p_2 & p_3 \\ p_3 & p_4 & p_1 & p_2 \\ p_2 & p_3 & p_4 & p_1 \end{bmatrix}$$

Complex circulant systems can be solved using `LSLCC`, page 251.

Iterative Solution of Linear Systems

The preconditioned conjugate gradient routines `PCGRC`, page 253, and `JCGRC`, page 259, can be used to solve symmetric positive definite systems. The routines are particularly useful if the system is large and sparse. These routines use reverse communication, so *A* can be in any storage scheme. For general linear systems, use `GMRES`, page 262.

QR Decomposition

The *QR* decomposition of a matrix *A* consists of finding an orthogonal matrix *Q*, a permutation matrix *P*, and an upper trapezoidal matrix *R* with diagonal elements of nonincreasing magnitude, such that *AP* = *QR*. This decomposition is determined by the routines LQRRR, page 286, or LQRRV, page 275. It returns *R* and the information needed to compute *Q*. To actually compute *Q* use LQERR, page 289. Figure 1-2 summarizes the relationships among the subroutines.

The *QR* decomposition can be used to solve the linear system $Ax = b$. This is equivalent to $Rx = Q^T Pb$. The routine LQRSL, page 292, can be used to find $Q^T Pb$ from the information computed by LQRRR. Then *x* can be computed by solving a triangular system using LSLRT, page 45. If the system $Ax = b$ is overdetermined, then this procedure solves the least-squares problem, i.e., it finds an *x* for which

$$\|Ax - b\|_2^2$$

is a minimum.

If the matrix *A* is changed by a rank-1 update, $A \to A + \alpha xy^T$, the QR decomposition of *A* can be updated/down-dated using the routine LUPQR, page 295. In some applications a series of linear systems which differ by rank-1 updates must be solved. Computing the QR decomposition once and then updating or down-dating it usually faster than newly solving each system.

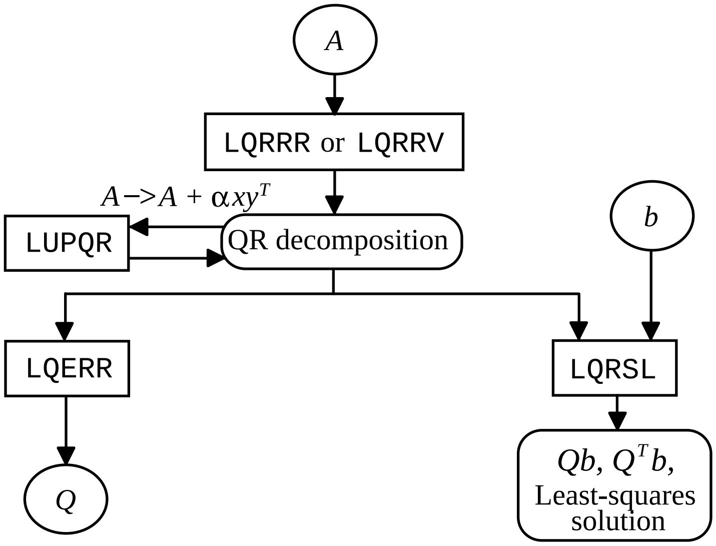


Figure 1-2   Least-Squares Routine

LSARG/DLSARG (Single/Double precision)

Solve a real general system of linear equations with iterative refinement.

Usage

`CALL LSARG (N, A, LDA, B, IPATH, X)`

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the coefficients of the linear system. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system $AX = B$ is solved.
`IPATH` = 2 means the system $A^TX = B$ is solved.

X — Vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSARG` $N^2 + 2N$ units, or
   `DLSARG` $2N^2 + 3N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2ARG/DL2ARG`. The reference is

   `CALL L2ARG (N, A, LDA, B, IPATH, X, FAC, IPVT, WK)`

   The additional arguments are as follows:

   FAC — Work vector of length N^2 containing the *LU* factorization of `A` on output.

   IPVT — Integer work vector of length `N` containing the pivoting information for the *LU* factorization of `A` on output.

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

Algorithm

Routine LSARG solves a system of linear algebraic equations having a real general coefficient matrix. It first uses the routine LFCRG, page 15, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using the iterative refinement routine LFIRG, page 22.

LSARG fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if *A* is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSARG solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of three linear equations is solved. The coefficient matrix has real general form and the right-hand-side vector *b* has three elements.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                 Set values for A and B
C
C                                 A = ( 33.0  16.0  72.0)
C                                     (-24.0 -10.0 -57.0)
C                                     ( 18.0 -11.0   7.0)
C
C                                 B = (129.0 -96.0   8.5)
C
      DATA A/33.0, -24.0, 18.0, 16.0, -10.0, -11.0, 72.0, -57.0, 7.0/
      DATA B/129.0, -96.0, 8.5/
C
      CALL LSARG (N, A, LDA, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
          X
    1       2       3
1.000   1.500   1.000
```

LSLRG/DLSLRG (Single/Double precision)

Solve a real general system of linear equations without iterative refinement.

Usage

```
CALL LSLRG (N, A, LDA, B, IPATH, X)
```

Arguments

N — Number of equations.   (Input)

A — N by N matrix containing the coefficients of the linear system.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator.   (Input)
IPATH = 1 means the system AX = B is solved.
IPATH = 2 means the system $A^T X = B$ is solved.

X — Vector of length N containing the solution to the linear system.   (Output)
If B is not needed, B and X can share the same storage locations.

Comments

1. Automatic workspace usage does not exceed

   LSLRG N(N + 1) + 2N units, or
   DLSLRG 2N(N + 1) + 3N units.

   Workspace may be explicitly provided, if desired, by use of L2LRG/DL2LRG. The reference is

   ```
   CALL L2LRG (N, A, LDA, B, IPATH, X, FAC, IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length N^2 containing the *LU* factorization of A on output. If A is not needed, A and FAC can share the same storage locations. See Item 3 below to avoid memory bank conflicts.

   IPVT — Integer work vector of length N containing the pivoting information for the *LU* factorization of A on output.

   WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.

4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LRG the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2); respectively, in LSLRG. Additional memory allocation for FAC and option value restoration are done automatically in LSLRG. Users directly calling L2LRG can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLRG or L2LRG. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLRG temporarily replaces IVAL(2) by IVAL(1). The routine L2CRG computes the condition number if IVAL(2) = 2. Otherwise L2CRG skips this computation. LSLRG restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLRG solves a system of linear algebraic equations having a real general coefficient matrix. It first uses the routine LFCRG (page 15) to compute an *LU* factorization of the coefficient matrix based on Gauss elimination with partial pivoting. Experiments were analyzed to determine efficient implementations on several different computers. For some supercomputers, particularly those with efficient vendor-supplied BLAS, page 1046, versions that call Level 1, 2 and 3 BLAS are used. The remaining computers use a factorization method provided to us by Dr. Leonard J. Harding of the University of Michigan. Harding’s work involves “loop unrolling and jamming” techniques that achieve excellent performance on many computers. Using an option, LSLRG will estimate the condition number of the matrix. The solution of the linear system is then found using LFSRG (page 20).

The routine LSLRG fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This occurs only if *A* is close to a singular matrix.

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that small changes in *A* can cause large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that either LSVRR, page 307, or LSARG, page 10, be used.

Example 1

A system of three linear equations is solved. The coefficient matrix has real general form and the right-hand-side vector *b* has three elements.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                 Set values for A and B
C
C                                 A = ( 33.0  16.0  72.0)
C                                     (-24.0 -10.0 -57.0)
C                                     ( 18.0 -11.0   7.0)
C
C                                 B = (129.0 -96.0   8.5)
C
      DATA A/33.0, -24.0, 18.0, 16.0, -10.0, -11.0, 72.0, -57.0, 7.0/
      DATA B/129.0, -96.0, 8.5/
C
      CALL LSLRG (N, A, LDA, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
           X
    1       2       3
1.000   1.500   1.000
```

Example 2

A system of `N` = 16 linear equations is solved using the routine `L2LRG`. The option manager is used to eliminate memory bank conflict inefficiencies that may occur when the matrix dimension is a multiple of 16. The leading dimension of `FAC` = `A` is increased from `N` to `N` + `IVAL`(3)=17, since `N`=16=`IVAL`(4). The data used for the test is a nonsymmetric Hadamard matrix and a right-hand side generated by a known solution, $x_j = j,\ j = 1, ..., N$.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=17, N=16)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    ICHP, IPATH, IPUT, KBANK
      REAL       ONE, ZERO
      PARAMETER  (ICHP=1, IPATH=1, IPUT=2, KBANK=16, ONE=1.0E0,
     &           ZERO=0.0E0)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IPVT(N), J, K, NN
      REAL       A(LDA,N), B(N), WK(N), X(N)
C                                 SPECIFICATIONS FOR SAVE VARIABLES
      INTEGER    IOPT(1), IVAL(4)
      SAVE       IVAL
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IUMAG, L2LRG, SGEMV, WRRRN
C                                 Data for option values.
      DATA IVAL/1, 16, 1, 16/
```

```
C                                  Set values for A and B:
      A(1,1) = ONE
      NN     = 1
C                                  Generate Hadamard matrix.
      DO 20  K=1, 4
         DO 10  J=1, NN
            DO 10  I=1, NN
               A(NN+I,J) = -A(I,J)
               A(I,NN+J) = A(I,J)
               A(NN+I,NN+J) = A(I,J)
   10    CONTINUE
         NN = NN + NN
   20 CONTINUE
C                                  Generate right-hand-side.
      DO 30  J=1, N
         X(J) = J
   30 CONTINUE
C                                  Set B = A*X.
      CALL SGEMV ('N', N, N, ONE, A, LDA, X, 1, ZERO, B, 1)
C                                  Clear solution array.
      DO 40  J=1, N
         X(J) = ZERO
   40 CONTINUE
C                                  Set option to avoid memory
C                                  bank conflicts.
      IOPT(1) = KBANK
      CALL IUMAG ('MATH', ICHP, IPUT, 1, IOPT, IVAL)
C                                  Solve A*X = B.
      CALL L2LRG (N, A, LDA, B, IPATH, X, A, IPVT, WK)
C                                  Print results
      CALL WRRRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                                          X
   1      2       3       4       5       6       7       8       9      10
1.00   2.00    3.00    4.00    5.00    6.00    7.00    8.00    9.00   10.00

   11      12      13      14      15      16
11.00   12.00   13.00   14.00   15.00   16.00
```

LFCRG/DLFCRG (Single/Double precision)

Compute the *LU* factorization of a real general matrix and estimate its L_1 condition number.

Usage

CALL LFCRG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)

Arguments

N — Order of the matrix. (Input)

A — N by N matrix to be factored. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC— `N` by `N` matrix containing the *LU* factorization of the matrix `A`. (Output) If *A* is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCRG` `N` units, or
   `DLFCRG` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CRG`/`DL2CRG`. The reference is

   ```
   CALL L2CRG (N, A, LDA, FAC, LDFAC, IPVT, RCOND, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is singular.

Algorithm

Routine `LFCRG` performs an *LU* factorization of a real general coefficient matrix. It also estimates the condition number of the matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

The L_1 condition number of the matrix *A* is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described in a paper by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system.

`LFCRG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if *A* either is singular or is very close to a singular matrix.

The *LU* factors are returned in a form that is compatible with routines LFIRG, page 22, LFSRG, page 20, and LFDRG, page 24. To solve systems of equations with multiple right-hand-side vectors, use LFCRG followed by either LFIRG or LFSRG called once for each right-hand side. The routine LFDRG can be called to compute the determinant of the coefficient matrix after LFCRG has performed the factorization.

Let F be the matrix FAC and let p be the vector IPVT. The triangular matrix U is stored in the upper triangle of F. The strict lower triangle of F contains the information needed to reconstruct L^{-1} using

$$L^{-1} = L_{N\text{-}1}P_{N\text{-}1} \dots L_1 P_1$$

where P_k is the identity matrix with rows k and p_k interchanged and L_k is the identity with F_{ik} for $i = k + 1, \dots, N$ inserted below the diagonal. The strict lower half of F can also be thought of as containing the negative of the multipliers. LFCRG is based on the LINPACK routine SGECO; see Dongarra et al. (1979). SGECO uses unscaled partial pivoting.

Example

The inverse of a 3 × 3 matrix is computed. LFCRG is called to factor the matrix and to check for singularity or ill-conditioning. LFIRG is called to determine the columns of the inverse.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), J, NOUT
      REAL       A(LDA,LDA), AINV(LDA,LDA), FAC(LDFAC,LDFAC), RCOND,
     &           RES(N), RJ(N)
C                                 Set values for A
C                                 A = (  1.0   3.0   3.0)
C                                     (  1.0   3.0   4.0)
C                                     (  1.0   4.0   3.0)
C
      DATA A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
C
      CALL LFCRG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the reciprocal condition number
C                                 and the L1 condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998) RCOND, 1.0E0/RCOND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIRG
C                                 reference places the J-th column of
```

```
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIRG (N, A, LDA, FAC, LDFAC, IPVT, RJ, IPATH,
     &               AINV(1,J), RES)
         RJ(J) = 0.0
   10 CONTINUE
C                                 Print results
      CALL WRRRN ('AINV', N, N, AINV, LDA, 0)
C
99998 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.015
L1 Condition number = 66.471

          AINV
        1       2       3
1   7.000  -3.000  -3.000
2  -1.000   0.000   1.000
3  -1.000   1.000   0.000
```

LFTRG/DLFTRG (Single/Double precision)

Compute the *LU* factorization of a real general matrix.

Usage

CALL LFTRG (N, A, LDA, FAC, LDFAC, IPVT)

Arguments

N — Order of the matrix. (Input)

A — N by N matrix to be factored. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

FAC — N by N matrix containing the *LU* factorization of the matrix A. (Output) If *A* is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

`LFTRG` `N` units, or
`DLFTRG` 2`N` units.

Workspace may be explicitly provided, if desired, by use of `L2TRG`/`DL2TRG`. The reference is

```
CALL L2TRG (N, A, LDA, FAC, LDFAC, IPVT, WK)
```

The additional argument is

WK — Work vector of length `N` used for scaling.

2. Informational error

Type	Code	
4	2	The input matrix is singular.

Algorithm

Routine `LFTRG` performs an *LU* factorization of a real general coefficient matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

The routine `LFTRG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if *A* is singular or very close to a singular matrix.

The *LU* factors are returned in a form that is compatible with routines `LFIRG` (page 22), `LFSRG` (page 20) and `LFDRG` (page 24). To solve systems of equations with multiple right-hand-side vectors, use `LFTRG` followed by either `LFIRG` or `LFSRG` called once for each right-hand side. The routine `LFDRG` can be called to compute the determinant of the coefficient matrix after `LFTRG` has performed the factorization. Let *F* be the matrix `FAC` and let *p* be the vector `IPVT`. The triangular matrix *U* is stored in the upper triangle of *F*. The strict lower triangle of *F* contains the information needed to reconstruct L^{-1} using

$$L^{-1} = L_{N-1}P_{N-1} \,.\,.\,.\, L_1P_1$$

where P_k is the identity matrix with rows k and p_k interchanged and L_k is the identity with F_{ik} for $i = k + 1, ..., N$ inserted below the diagonal. The strict lower half of *F* can also be thought of as containing the negative of the multipliers.

Routine `LFTRG` is based on the LINPACK routine `SGEFA`. See Dongarra et al. (1979). The routine `SGEFA` uses partial pivoting.

Example

A linear system with multiple right-hand sides is solved. Routine `LFTRG` is called to factor the coefficient matrix. The routine `LFSRG` is called to compute the two solutions for the two right-hand sides. In this case, the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call LFCRG (page 15) to perform the factorization, and LFIRG (page 22) to compute the solutions.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), J
      REAL       A(LDA,LDA), B(N,2), FAC(LDFAC,LDFAC), X(N,2)
C
C                                 Set values for A and B
C
C                                 A = (  1.0   3.0   3.0)
C                                     (  1.0   3.0   4.0)
C                                     (  1.0   4.0   3.0)
C
C                                 B = (  1.0  10.0)
C                                     (  4.0  14.0)
C                                     ( -1.0   9.0)
C
      DATA A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
      DATA B/1.0, 4.0, -1.0, 10.0, 14.0, 9.0/
C
      CALL LFTRG (N, A, LDA, FAC, LDFAC, IPVT)
C                                 Solve for the two right-hand sides
      DO 10  J=1, 2
         CALL LFSRG (N, FAC, LDFAC, IPVT, B(1,J), IPATH, X(1,J))
   10 CONTINUE
C                                 Print results
      CALL WRRRN ('X', N, 2, X, N, 0)
      END
```

Output

```
         X
        1        2
1  -2.000   1.000
2  -2.000  -1.000
3   3.000   4.000
```

LFSRG/DLFSRG (Single/Double precision)

Solve a real general system of linear equations given the *LU* factorization of the coefficient matrix.

Usage

CALL LFSRG (N, FAC, LDFAC, IPVT, B, IPATH, X)

Arguments

N — Number of equations.   (Input)

FAC — N by N matrix containing the *LU* factorization of the coefficient matrix A as output from routine LFCRG (page 15).   (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization of `A` as output from subroutine `LFCRG` (page 15) or `LFTRG/DLFTRG` (page 18). (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system $AX = B$ is solved.
`IPATH` = 2 means the system $A^TX = B$ is solved.

X — Vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Algorithm

Routine `LFSRG` computes the solution of a system of linear algebraic equations having a real general coefficient matrix. To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCRG`, page 15, or `LFTRG`, page 18. The solution to $Ax = b$ is found by solving the triangular systems $Ly = b$ and $Ux = y$. The forward elimination step consists of solving the system $Ly = b$ by applying the same permutations and elimination operations to b that were applied to the columns of A in the factorization routine. The backward substitution step consists of solving the triangular system $Ux = y$ for x.

`LFSRG` and `LFIRG`, page 22, both solve a linear system given its *LU* factorization. `LFIRG` generally takes more time and produces a more accurate answer than `LFSRG`. Each iteration of the iterative refinement algorithm used by `LFIRG` calls `LFSRG`. The routine `LFSRG` is based on the LINPACK routine `SGESL`; see Dongarra et al. (1979).

Example

The inverse is computed for a real general 3 × 3 matrix. The input matrix is assumed to be well-conditioned, hence, `LFTRG` is used rather than `LFCRG`.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    I, IPVT(N), J
      REAL       A(LDA,LDA), AINV(LDA,LDA), FAC(LDFAC,LDFAC), RJ(N)
C
C                                 Set values for A
C                                 A = (  1.0   3.0   3.0)
C                                     (  1.0   3.0   4.0)
C                                     (  1.0   4.0   3.0)
C
      DATA A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
C
      CALL LFTRG (N, A, LDA, FAC, LDFAC, IPVT)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0, RJ, 1)
```

```
DO 10  J=1, N
         RJ(J) = 1.0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSRG
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSRG (N, FAC, LDFAC, IPVT, RJ, IPATH, AINV(1,J))
         RJ(J) = 0.0
   10 CONTINUE
C                                 Print results
      CALL WRRRN (’AINV’, N, N, AINV, LDA, 0)
      END
```

Output

```
           AINV
        1       2       3
1   7.000  -3.000  -3.000
2  -1.000   0.000   1.000
3  -1.000   1.000   0.000
```

LFIRG/DLFIRG (Single/Double precision)

Use iterative refinement to improve the solution of a real general system of linear equations.

Usage

```
CALL LFIRG (N, A, LDA, FAC, LDFAC, IPVT, B, IPATH, X, RES)
```

Arguments

N — Number of equations.   (Input)

A — N by N matrix containing the coefficient matrix of the linear system.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

FAC — N by N matrix containing the *LU* factorization of the coefficient matrix A as output from routine LFCRG/DLFCRG or LFTRG/DLFTRG.   (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization of A as output from routine LFCRG/DLFCRG or LFTRG/DLFTRG. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system A * X = B is solved.
IPATH = 2 means the system $A^T X = B$ is solved.

X — Vector of length N containing the solution to the linear system. (Output)

RES — Vector of length N containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	2	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine LFIRG computes the solution of a system of linear algebraic equations having a real general coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either LFCRG, page 15, or LFTRG, page 18.

Iterative refinement fails only if the matrix is very ill-conditioned.

Routines LFIRG and LFSRG (page 20) both solve a linear system given its *LU* factorization. LFIRG generally takes more time and produces a more accurate answer than LFSRG. Each iteration of the iterative refinement algorithm used by LFIRG calls LFSRG.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.5 to the second element.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       A(LDA,LDA), B(N), FAC(LDFAC,LDFAC), RCOND, RES(N), X(N)
C
C                                 Set values for A and B
C
C                                 A = (  1.0   3.0   3.0)
C                                     (  1.0   3.0   4.0)
C                                     (  1.0   4.0   3.0)
C
C                                 B = ( -0.5  -1.0   1.5)
C
      DATA A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
      DATA B/-0.5, -1.0, 1.5/
C
      CALL LFCRG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)
C                                  Print the reciprocal condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                  Solve the three systems
      DO 10  J=1, 3
         CALL LFIRG (N, A, LDA, FAC, LDFAC, IPVT, B, IPATH, X, RES)
C                                  Print results
         CALL WRRRN ('X', 1, N, X, 1, 0)
C                                  Perturb B by adding 0.5 to B(2)
         B(2) = B(2) + 0.5
   10 CONTINUE
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.015
L1 Condition number = 66.471
            X
     1       2       3
-5.000   2.000  -0.500

            X
     1       2       3
-6.500   2.000   0.000

            X
     1       2       3
-8.000   2.000   0.500
```

LFDRG/DLFDRG (Single/Double precision)

Compute the determinant of a real general matrix given the *LU* factorization of the matrix.

Usage

CALL LFDRG (N, FAC, LDFAC, IPVT, DET1, DET2)

Arguments

N — Order of the matrix.   (Input)

FAC — N by N matrix containing the *LU* factorization of the matrix A as output from routine LFCRG/DLFCRG (page 15).   (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization as output from routine LFTRG/DLFTRG or LFCRG/DLFCRG.   (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value `DET1` is normalized so that 1.0 ≤ `|DET1|` < 10.0 or `DET1` = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det(A) = `DET1 *` $10^{\texttt{DET2}}$.

Algorithm

Routine `LFDRG` computes the determinant of a real general coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an LU factorization. This may be done by calling either `LFCRG` (page 15) or `LFTRG` (page 18). The formula det A = det L det U is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements

$$\det U = \prod_{i=1}^{N} U_{ii}$$

(The matrix U is stored in the upper triangle of `FAC`.) Since L is the product of triangular matrices with unit diagonals and of permutation matrices, det $L = (-1)^k$ where k is the number of pivoting interchanges.

Routine `LFDRG` is based on the LINPACK routine `SGEDI`; see Dongarra et al. (1979).

Example

The determinant is computed for a real general 3 × 3 matrix.

```
C                                 Declare variables
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       A(LDA,LDA), DET1, DET2, FAC(LDFAC,LDFAC)
C
C                                 Set values for A
C                                 A = ( 33.0  16.0  72.0)
C                                     (-24.0 -10.0 -57.0)
C                                     ( 18.0 -11.0   7.0)
C
      DATA A/33.0, -24.0, 18.0, 16.0, -10.0, -11.0, 72.0, -57.0, 7.0/
C
      CALL LFTRG (N, A, LDA, FAC, LDFAC, IPVT)
C                                 Compute the determinant
      CALL LFDRG (N, FAC, LDFAC, IPVT, DET1, DET2)
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant of A is ’, F6.3, ’ * 10**’, F2.0)
      END
```

Output

```
The determinant of A is -4.761 * 10**3.
```

LINRG/DLINRG (Single/Double precision)

Compute the inverse of a real general matrix.

Usage

`CALL LINRG (N, A, LDA, AINV, LDAINV)`

Arguments

N — Order of the matrix `A`. (Input)

A — `N` by `N` matrix containing the matrix to be inverted. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

AINV — `N` by `N` matrix containing the inverse of `A`. (Output)
If `A` is not needed, `A` and `AINV` can share the same storage locations.

LDAINV — Leading dimension of `AINV` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   `LINRG` 2`N` + `N`(`N` − 1)/2 units, or
   `DLINRG` 3`N` + `N`(`N` − 1) units.

   Workspace may be explicitly provided, if desired, by use of `L2NRG`/`DL2NRG`. The reference is

   `CALL L2NRG (N, A, LDA, AINV, LDAINV, WK, IWK)`

   The additional arguments are as follows:

   WK — Work vector of length `N`+ `N`(`N` − 1)/2.

   IWK — Integer work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The inverse might not be accurate.
4	2	The input matrix is singular.

Algorithm

Routine `LINRG` computes the inverse of a real general matrix. It first uses the routine `LFCRG` (page 15) to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. Routine `LFCRG` computes *U* and the information needed to compute L^{-1}. `LINRT`, page 49, is then used to compute U^{-1}. Finally, A^{-1} is computed using $A^{-1} = U^{-1}L^{-1}$.

The routine `LINRG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. This error occurs only if *A* is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in A^{-1}.

Example

The inverse is computed for a real general 3 × 3 matrix.

```
C                                 Declare variables
      PARAMETER  (LDA=3, LDAINV=3, N=3)
      INTEGER    I, J, NOUT
      REAL       A(LDA,LDA), AINV(LDAINV,LDAINV)
C
C                                 Set values for A
C                                 A = (  1.0   3.0   3.0)
C                                     (  1.0   3.0   4.0)
C                                     (  1.0   4.0   3.0)
C
      DATA A/1.0, 1.0, 1.0, 3.0, 3.0, 4.0, 3.0, 4.0, 3.0/
C
      CALL LINRG (N, A, LDA, AINV, LDAINV)
C                                 Print results
      CALL WRRRN (’AINV’, N, N, AINV, LDAINV, 0)
      END
```

Output

```
             AINV
        1        2        3
1   7.000  -3.000  -3.000
2  -1.000   0.000   1.000
3  -1.000   1.000   0.000
```

LSACG/DLSACG (Single/Double precision)

Solve a complex general system of linear equations with iterative refinement.

Usage

`CALL LSACG (N, A, LDA, B, IPATH, X)`

Arguments

N — Number of equations. (Input)

A — Complex `N` by `N` matrix containing the coefficients of the linear system. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX = B is solved.

IPATH = 2 means the system $A^H X$ = B is solved.

X — Complex vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSACG $2N^2 + 3N$ units, or
   DLSACG $4N^2 + 5N$ units.

   Workspace may be explicitly provided, if desired, by use of L2ACG/DL2ACG. The reference is

   ```
   CALL L2ACG (N, A, LDA, B, IPATH, X, FAC, IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length N^2 containing the *LU* factorization of A on output.

   IPVT — Integer work vector of length N containing the pivoting information for the *LU* factorization of A on output.

   WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2ACG the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2); respectively, in LSACG. Additional memory allocation for FAC and option value restoration are done automatically in LSACG. Users directly calling L2ACG can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSACG or L2ACG. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSACG temporarily replaces IVAL(2) by IVAL(1). The routine L2CCG computes the condition number if IVAL(2) = 2. Otherwise L2CCG skips this computation. LSACG restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSACG solves a system of linear algebraic equations with a complex general coefficient matrix. It first uses the routine LFCCG, page 32, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using the iterative refinement routine LFICG, page 39.

LSACG fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if *A* is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSACG solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of three linear equations is solved. The coefficient matrix has complex general form and the right-hand-side vector *b* has three elements.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C                                 Set values for  A and B
C
C                                 A = ( 3.0-2.0i  2.0+4.0i  0.0-3.0i)
C                                     ( 1.0+1.0i  2.0-6.0i  1.0+2.0i)
C                                     ( 4.0+0.0i -5.0+1.0i  3.0-2.0i)
C
C                                 B = (10.0+5.0i  6.0-7.0i -1.0+2.0i)
C
      DATA A/(3.0,-2.0), (1.0,1.0),  (4.0,0.0), (2.0,4.0), (2.0,-6.0),
     &       (-5.0,1.0), (0.0,-3.0), (1.0,2.0), (3.0,-2.0)/
      DATA B/(10.0,5.0), (6.0,-7.0), (-1.0,2.0)/
C                                 Solve AX = B     (IPATH = 1)
      CALL LSACG (N, A, LDA, B, IPATH, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
                                    X
                 1                 2                 3
( 1.000,-1.000)  ( 2.000, 1.000)  ( 0.000, 3.000)
```

LSLCG/DLSLCG (Single/Double precision)

Solve a complex general system of linear equations without iterative refinement.

Usage

```
CALL LSLCG (N, A, LDA, B, IPATH, X)
```

Arguments

N — Number of equations. (Input)

A — Complex `N` by `N` matrix containing the coefficients of the linear system. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system `AX` = `B` is solved.
`IPATH` = 2 means the system $A^H X = B$ is solved.

X — Complex vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Comments

1. Automatic workspace usage is

   `LSLCG` $2N^2 + 3N$ units, or

   `DLSLCG` $4N^2 + 5N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2LCG/DL2LCG`. The reference is

   ```
   CALL L2LCG (N, A, LDA, B, IPATH, X, FAC, IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length N^2 containing the *LU* factorization of `A` on output. If `A` is not needed, `A` and `FAC` can share the same storage locations.

IPVT — Integer work vector of length N containing the pivoting information for the *LU* factorization of A on output.

WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LCG the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2); respectively, in LSLCG. Additional memory allocation for FAC and option value restoration are done automatically in LSLCG. Users directly calling L2LCG can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLCG or L2LCG. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLCG temporarily replaces IVAL(2) by IVAL(1). The routine L2CCG computes the condition number if IVAL(2) = 2. Otherwise L2CCG skips this computation. LSLCG restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLCG solves a system of linear algebraic equations with a complex general coefficient matrix. It first uses the routine LFCCG, page 32, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using LFSCG, page 37.

LSLCG fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This occurs only if *A* either is a singular matrix or is very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that LSACG, page 27, be used.

Example

A system of three linear equations is solved. The coefficient matrix has complex general form and the right-hand-side vector *b* has three elements.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C                                 Set values for  A and B
C
C                                 A = ( 3.0-2.0i  2.0+4.0i  0.0-3.0i)
C                                     ( 1.0+1.0i  2.0-6.0i  1.0+2.0i)
C                                     ( 4.0+0.0i -5.0+1.0i  3.0-2.0i)
C
C                                 B = (10.0+5.0i  6.0-7.0i -1.0+2.0i)
C
      DATA A/(3.0,-2.0), (1.0,1.0),  (4.0,0.0), (2.0,4.0), (2.0,-6.0),
     &       (-5.0,1.0), (0.0,-3.0), (1.0,2.0), (3.0,-2.0)/
      DATA B/(10.0,5.0), (6.0,-7.0), (-1.0,2.0)/
C                                 Solve AX = B     (IPATH = 1)
      CALL LSLCG (N, A, LDA, B, IPATH, X)
C                                 Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                          X
              1                 2                 3
( 1.000,-1.000)  ( 2.000, 1.000)  ( 0.000, 3.000)
```

LFCCG/DLFCCG (Single/Double precision)

Compute the *LU* factorization of a complex general matrix and estimate its L_1 condition number.

Usage

CALL LFCCG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)

Arguments

N — Order of the matrix.   (Input)

A — Complex N by N matrix to be factored.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

FAC — Complex N by N matrix containing the *LU* factorization of the matrix A (Output)
If A is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCCG` 2`N` units, or
   `DLFCCG` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CCG/DL2CCG`. The reference is

   ```
   CALL L2CCG (N, A, LDA, FAC, LDFAC, IPVT, RCOND, WK)
   ```

   The additional argument is

   WK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is singular.

Algorithm

Routine `LFCCG` performs an *LU* factorization of a complex general coefficient matrix. It also estimates the condition number of the matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

The L_1 condition number of the matrix *A* is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system.

`LFCCG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if *A* either is singular or is very close to a singular matrix.

The *LU* factors are returned in a form that is compatible with routines `LFICG`, page 39, `LFSCG`, page 37, and `LFDCG`, page 42. To solve systems of equations with multiple right-hand-side vectors, use `LFCCG` followed by either `LFICG` or `LFSCG` called once for each right-hand side. The routine `LFDCG` can be called to compute the determinant of the coefficient matrix after LFCCG has performed the factorization.

Let F be the matrix FAC and let p be the vector IPVT. The triangular matrix U is stored in the upper triangle of F. The strict lower triangle of F contains the information needed to reconstruct L^{-1} using

$$L^{-1} = L_{N\text{-}1}P_{N\text{-}1} \ldots L_1P_1$$

where P_k is the identity matrix with rows k and p_k interchanged and L_k is the identity with F_{ik} for $i = k + 1, ...,$ N inserted below the diagonal. The strict lower half of F can also be thought of as containing the negative of the multipliers.

LFCCG is based on the LINPACK routine CGECO; see Dongarra et al. (1979). CGECO uses unscaled partial pivoting.

Example

The inverse of a 3 × 3 matrix is computed. LFCCG is called to factor the matrix and to check for singularity or ill-conditioning. LFICG (page 39) is called to determine the columns of the inverse.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       RCOND, THIRD
      COMPLEX    A(LDA,LDA), AINV(LDA,LDA), RJ(N), FAC(LDFAC,LDFAC),
     &           RES(N)
C                                 Declare functions
      COMPLEX    CMPLX
C                                 Set values for  A
C
C                                 A = (  1.0+1.0i  2.0+3.0i  3.0+3.0i)
C                                     (  2.0+1.0i  5.0+3.0i  7.0+4.0i)
C                                     ( -2.0+1.0i -4.0+4.0i -5.0+3.0i)
C
      DATA A/(1.0,1.0), (2.0,1.0), (-2.0,1.0), (2.0,3.0), (5.0,3.0),
     &     (-4.0,4.0), (3.0,3.0), (7.0,4.0), (-5.0,3.0)/
C
C                                 Scale A by dividing by three
      THIRD = 1.0/3.0
      DO 10  I=1, N
         CALL CSSCAL (N, THIRD, A(1,I), 1)
   10 CONTINUE
C                                 Factor A
      CALL LFCCG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the L1 condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0,0.0), RJ, 1)
      DO 20  J=1, N
         RJ(J) = CMPLX(1.0,0.0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIRG
```

```
C                                  reference places the J-th column of
C                                  the inverse of A in the J-th column
C                                  of AINV
         CALL LFICG (N, A, LDA, FAC, LDFAC, IPVT, RJ, IPATH,
     &               AINV(1,J), RES)
         RJ(J) = CMPLX(0.0,0.0)
   20 CONTINUE
C                                  Print results
      CALL WRCRN ('AINV', N, N, AINV, LDA, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.016
L1 Condition number = 63.104

                         AINV
                 1                  2                  3
1  ( 6.400,-2.800)  (-3.800, 2.600)  (-2.600, 1.200)
2  (-1.600,-1.800)  ( 0.200, 0.600)  ( 0.400,-0.800)
3  (-0.600, 2.200)  ( 1.200,-1.400)  ( 0.400, 0.200)
```

LFTCG/DLFTCG (Single/Double precision)

Compute the *LU* factorization of a complex general matrix.

Usage

CALL LFTCG (N, A, LDA, FAC, LDFAC, IPVT)

Arguments

N — Order of the matrix. (Input)

A — Complex N by N matrix to be factored. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex N by N matrix containing the *LU* factorization of the matrix A. (Output)
If A is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

`LFTCG` 2`N` units, or
`DLFTCG` 4`N` units.

Workspace may be explicitly provided, if desired, by use of `L2TCG/DL2TCG`. The reference is

```
CALL L2TCG (N, A, LDA, FAC, LDFAC, IPVT, WK)
```

The additional argument is

WK — Complex work vector of length `N`.

2. Informational error

Type	Code	
4	2	The input matrix is singular.

Algorithm

Routine `LFTCG` performs an *LU* factorization of a complex general coefficient matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

`LFTCG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if *A* either is singular or is very close to a singular matrix.

The *LU* factors are returned in a form that is compatible with routines `LFICG`, page 39, `LFSCG`, page 37, and `LFDCG`, page 42. To solve systems of equations with multiple right-hand-side vectors, use `LFTCG` followed by either `LFICG` or `LFSCG` called once for each right-hand side. The routine `LFDCG` can be called to compute the determinant of the coefficient matrix after `LFCCG` (page 32) has performed the factorization.

Let *F* be the matrix `FAC` and let *p* be the vector `IPVT`. The triangular matrix *U* is stored in the upper triangle of *F*. The strict lower triangle of *F* contains the information needed to reconstruct L^{-1} using

$$L^{-1} = L_{N\text{-}1}P_{N\text{-}1} \dots L_1P_1$$

where P_k is the identity matrix with rows *k* and P_k interchanged and L_k is the identity with F_{ik} for $i = k + 1, ..., N$ inserted below the diagonal. The strict lower half of *F* can also be thought of as containing the negative of the multipliers.

`LFTCG` is based on the LINPACK routine `CGEFA`; see Dongarra et al. (1979). `CGEFA` uses unscaled partial pivoting.

Example

A linear system with multiple right-hand sides is solved. `LFTCG` is called to factor the coefficient matrix. `LFSCG` is called to compute the two solutions for the two right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call `LFCCG` to perform the factorization, and `LFICG` to compute the solutions.

```
C                                  Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N)
      COMPLEX    A(LDA,LDA), B(N,2), X(N,2), FAC(LDFAC,LDFAC)
C                                  Set values for  A
C                                  A = ( 1.0+1.0i  2.0+3.0i  3.0-3.0i)
C                                      ( 2.0+1.0i  5.0+3.0i  7.0-5.0i)
C                                      (-2.0+1.0i -4.0+4.0i  5.0+3.0i)
C
      DATA A/(1.0,1.0), (2.0,1.0), (-2.0,1.0), (2.0,3.0), (5.0,3.0),
     &     (-4.0,4.0), (3.0,-3.0), (7.0,-5.0), (5.0,3.0)/
C
C                                  Set the right-hand sides, B
C                                  B = (  3.0+ 5.0i  9.0+ 0.0i)
C                                      ( 22.0+10.0i 13.0+ 9.0i)
C                                      (-10.0+ 4.0i  6.0+10.0i)
C
      DATA B/(3.0,5.0), (22.0,10.0), (-10.0,4.0), (9.0,0.0),
     &     (13.0,9.0), (6.0,10.0)/
C
C                                  Factor A
      CALL LFTCG (N, A, LDA, FAC, LDFAC, IPVT)
C                                  Solve for the two right-hand sides
      DO 10  J=1, 2
         CALL LFSCG (N, FAC, LDFAC, IPVT, B(1,J), IPATH, X(1,J))
   10 CONTINUE
C                                  Print results
      CALL WRCRN (’X’, N, 2, X, N, 0)
      END
```

Output

```
                X
               1                 2
1  ( 1.000,-1.000)  ( 0.000, 2.000)
2  ( 2.000, 4.000)  (-2.000,-1.000)
3  ( 3.000, 0.000)  ( 1.000, 3.000)
```

LFSCG/DLFSCG (Single/Double precision)

Solve a complex general system of linear equations given the *LU* factorization of the coefficient matrix.

Usage

`CALL LFSCG (N, FAC, LDFAC, IPVT, B, IPATH, X)`

Arguments

N — Number of equations.   (Input)

FAC — Complex `N` by `N` matrix containing the *LU* factorization of the coefficient matrix `A` as output from routine `LFCCG/DLFCCG` or `LFTCG/DLFTCG`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization of `A` as output from routine `LFCCG/DLFCCG` or `LFTCG/DLFTCG`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system `AX` = `B` is solved.

`IPATH` = 2 means the system $A^H X = B$ is solved.

X — Complex vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Algorithm

Routine `LFSCG` computes the solution of a system of linear algebraic equations having a complex general coefficient matrix. To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCCG`, page 32, or `LFTCG`, page 35. The solution to $Ax = b$ is found by solving the triangular systems $Ly = b$ and $Ux = y$. The forward elimination step consists of solving the system $Ly = b$ by applying the same permutations and elimination operations to b that were applied to the columns of A in the factorization routine. The backward substitution step consists of solving the triangular system $Ux = y$ for x.

Routines `LFSCG` and `LFICG` (page 39) both solve a linear system given its *LU* factorization. `LFICG` generally takes more time and produces a more accurate answer than `LFSCG`. Each iteration of the iterative refinement algorithm used by `LFICG` calls `LFSCG`.

`LFSCG` is based on the LINPACK routine `CGESL`; see Dongarra et al. (1979).

Example

The inverse is computed for a complex general 3 × 3 matrix. The input matrix is assumed to be well-conditioned, hence `LFTCG` (page 35) is used rather than `LFCCG`.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N)
      REAL       THIRD
      COMPLEX    A(LDA,LDA), AINV(LDA,LDA), RJ(N), FAC(LDFAC,LDFAC)
C                                 Declare functions
```

```
      COMPLEX    CMPLX
C                                Set values for  A
C
C                                A = (  1.0+1.0i  2.0+3.0i  3.0+3.0i)
C                                    (  2.0+1.0i  5.0+3.0i  7.0+4.0i)
C                                    ( -2.0+1.0i -4.0+4.0i -5.0+3.0i)
C
      DATA A/(1.0,1.0), (2.0,1.0), (-2.0,1.0), (2.0,3.0), (5.0,3.0),
     &     (-4.0,4.0), (3.0,3.0), (7.0,4.0), (-5.0,3.0)/
C
C                                Scale A by dividing by three
      THIRD = 1.0/3.0
      DO 10  I=1, N
         CALL CSSCAL (N, THIRD, A(1,I), 1)
   10 CONTINUE
C                                Factor A
      CALL LFTCG (N, A, LDA, FAC, LDFAC, IPVT)
C                                Set up the columns of the identity
C                                matrix one at a time in RJ
      CALL CSET (N, (0.0,0.0), RJ, 1)
      DO 20  J=1, N
         RJ(J) = CMPLX(1.0,0.0)
C                                RJ is the J-th column of the identity
C                                matrix so the following LFSCG
C                                reference places the J-th column of
C                                the inverse of A in the J-th column
C                                of AINV
         CALL LFSCG (N, FAC, LDFAC, IPVT, RJ, IPATH, AINV(1,J))
         RJ(J) = CMPLX(0.0,0.0)
   20 CONTINUE
C                                Print results
      CALL WRCRN (’AINV’, N, N, AINV, LDA, 0)
      END
```

Output

```
                          AINV
                1                 2                 3
1  ( 6.400,-2.800)  (-3.800, 2.600)  (-2.600, 1.200)
2  (-1.600,-1.800)  ( 0.200, 0.600)  ( 0.400,-0.800)
3  (-0.600, 2.200)  ( 1.200,-1.400)  ( 0.400, 0.200)
```

LFICG/DLFICG (Single/Double precision)

Use iterative refinement to improve the solution of a complex general system of linear equations.

Usage

`CALL LFICG (N, A, LDA, FAC, LDFAC, IPVT, B, IPATH, X, RES)`

Arguments

N — Number of equations.   (Input)

A — Complex `N` by `N` matrix containing the coefficient matrix of the linear system. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex `N` by `N` matrix containing the *LU* factorization of the coefficient matrix *A* as output from routine `LFCCG/DLFCCG` or `LFTCG/DLFTCG`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization of `A` as output from routine `LFCCG/DLFCCG` or `LFTCG/DLFTCG`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system `AX` = `B` is solved.
`IPATH` = 2 means the system $\mathtt{A}^H\mathtt{X}$ = `B` is solved.

X — Complex vector of length `N` containing the solution to the linear system. (Output)

RES — Complex vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	2	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFICG` computes the solution of a system of linear algebraic equations having a complex general coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCCG`, page 32, or `LFTCG`, page 35.

Iterative refinement fails only if the matrix is very ill-conditioned. Routines `LFICG` and `LFSCG` (page 37) both solve a linear system given its *LU* factorization. `LFICG` generally takes more time and produces a more accurate answer than LFSCG. Each iteration of the iterative refinement algorithm used by LFICG calls LFSCG.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.5 + 0.5*i* to the second element.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       RCOND
      COMPLEX    A(LDA,LDA), B(N), X(N), FAC(LDFAC,LDFAC), RES(N)
C                                 Declare functions
      COMPLEX    CMPLX
C                                 Set values for  A
C
C                                 A = (  1.0+1.0i  2.0+3.0i  3.0-3.0i)
C                                     (  2.0+1.0i  5.0+3.0i  7.0-5.0i)
C                                     ( -2.0+1.0i -4.0+4.0i  5.0+3.0i)
C
      DATA A/(1.0,1.0), (2.0,1.0), (-2.0,1.0), (2.0,3.0), (5.0,3.0),
     &     (-4.0,4.0), (3.0,-3.0), (7.0,-5.0), (5.0,3.0)/
C
C                                 Set values for B
C                                 B = ( 3.0+5.0i 22.0+10.0i -10.0+4.0i)
C
      DATA B/(3.0,5.0), (22.0,10.0), (-10.0,4.0)/
C                                 Factor A
      CALL LFCCG (N, A, LDA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the L1 condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Solve the three systems
      DO 10  J=1, 3
         CALL LFICG (N, A, LDA, FAC, LDFAC, IPVT, B, IPATH, X, RES)
C                                 Print results
         CALL WRCRN (’X’, 1, N, X, 1, 0)
C                                 Perturb B by adding 0.5+0.5i to B(2)
         B(2) = B(2) + CMPLX(0.5,0.5)
   10 CONTINUE
C
99999 FORMAT (’  RCOND = ’,F5.3,/,’  L1 Condition number = ’,F6.3)
      END
```

Output

```
RCOND = 0.023
L1 Condition number = 42.799
                        X
              1                 2                 3
( 1.000,-1.000)  ( 2.000, 4.000)  ( 3.000, 0.000)

                        X
              1                 2                 3
( 0.910,-1.061)  ( 1.986, 4.175)  ( 3.123, 0.071)
```

```
                     X
              1                 2                 3
( 0.821,-1.123)  ( 1.972, 4.349)  ( 3.245, 0.142)
```

LFDCG/DLFDCG (Single/Double precision)

Compute the determinant of a complex general matrix given the *LU* factorization of the matrix.

Usage

`CALL LFDCG (N, FAC, LDFAC, IPVT, DET1, DET2)`

Arguments

N — Order of the matrix. (Input)

FAC — Complex `N` by `N` matrix containing the *LU* factorization of the matrix `A` as output from routine `LFTCG/DLFTCG` or `LFCCG/DLFCCG`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization as output from routine `LFTCG/DLFTCG` or `LFCCG/DLFCCG`. (Input)

DET1 — Complex scalar containing the mantissa of the determinant. (Output) The value `DET1` is normalized so that 1.0 ≤ |`DET1`| < 10.0 or `DET1` = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output) The determinant is returned in the form det(*A*) = `DET1 *` 10^{DET2}.

Algorithm

Routine `LFDCG` computes the determinant of a complex general coefficient matrix. To compute the determinant the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCCG`, page 32, or `LFTCG`, page 35. The formula det *A* = det *L* det *U* is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det U = \prod\nolimits_{i=1}^{N} U_{ii}$$

(The matrix *U* is stored in the upper triangle of `FAC`.) Since *L* is the product of triangular matrices with unit diagonals and of permutation matrices, det $L = (-1)^k$ where *k* is the number of pivoting interchanges.

`LFDCG` is based on the LINPACK routine `CGEDI`; see Dongarra et al. (1979).

Example

The determinant is computed for a complex general 3 × 3 matrix.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, LDFAC=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       DET2
      COMPLEX    A(LDA,LDA), FAC(LDFAC,LDFAC), DET1
C                                 Set values for  A
C
C                                 A = (  3.0-2.0i  2.0+4.0i  0.0-3.0i)
C                                     (  1.0+1.0i  2.0-6.0i  1.0+2.0i)
C                                     (  4.0+0.0i -5.0+1.0i  3.0-2.0i)
C
      DATA A/(3.0,-2.0), (1.0,1.0), (4.0,0.0), (2.0,4.0), (2.0,-6.0),
     &       (-5.0,1.0), (0.0,-3.0), (1.0,2.0), (3.0,-2.0)/
C
C                                 Factor A
      CALL LFTCG (N, A, LDA, FAC, LDFAC, IPVT)
C                                 Compute the determinant for the
C                                 factored matrix
      CALL LFDCG (N, FAC, LDFAC, IPVT, DET1, DET2)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant of A is’,3X,’(’,F6.3,’,’,F6.3,
     &        ’) * 10**’,F2.0)
      END
```

Output

```
The determinant of A is ( 0.700, 1.100) * 10**1.
```

LINCG/DLINCG (Single/Double precision)

Compute the inverse of a complex general matrix.

Usage

```
CALL LINCG (N, A, LDA, AINV, LDAINV)
```

Arguments

N — Order of the matrix `A`. (Input)

A — Complex `N` by `N` matrix containing the matrix to be inverted. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

AINV — Complex `N` by `N` matrix containing the inverse of `A`. (Output)
If *A* is not needed, `A` and `AINV` can share the same storage locations.

LDAINV — Leading dimension of `AINV` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   `LINCG` 3`N` + `N`(`N` – 1) units, or
   `DLINCG` 5`N` + 2`N`(`N` – 1) units.

   Workspace may be explicitly provided, if desired, by use of `L2NCG/DL2NCG`. The reference is

   `CALL L2NCG (N, A, LDA, AINV, LDAINV, WK, IWK)`

   The additional arguments are as follows:

   WK — Complex work vector of length `N` + `N`(`N` – 1)/2.

   IWK — Integer work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The inverse might not be accurate.
4	2	The input matrix is singular.

Algorithm

Routine `LINCG` computes the inverse of a complex general matrix.

It first uses the routine `LFCCG`, page 32, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. `LFCCG` computes *U* and the information needed to compute L^{-1}. `LINCT`, page 55, is then used to compute U^{-1}. Finally A^{-1} is computed using $A^{-1} = U^{-1}L^{-1}$.

`LINCG` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. This errors occurs only if *A* is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in A^{-1}.

Example

The inverse is computed for a complex general 3 × 3 matrix.

```
C                                  Declare variables
      PARAMETER  (LDA=3, LDAINV=3, N=3)
      REAL       THIRD
      COMPLEX    A(LDA,LDA), AINV(LDAINV,LDAINV)
C                                  Set values for  A
C
C                                  A = (  1.0+1.0i  2.0+3.0i  3.0+3.0i)
C                                      (  2.0+1.0i  5.0+3.0i  7.0+4.0i)
C                                      ( -2.0+1.0i -4.0+4.0i -5.0+3.0i)
C
```

```
     DATA A/(1.0,1.0), (2.0,1.0), (-2.0,1.0), (2.0,3.0), (5.0,3.0),
     &     (-4.0,4.0), (3.0,3.0), (7.0,4.0), (-5.0,3.0)/
C
C                                  Scale A by dividing by three
      THIRD = 1.0/3.0
      DO 10  I=1, N
         CALL CSSCAL (N, THIRD, A(1,I), 1)
   10 CONTINUE
C                                  Calculate the inverse of A
      CALL LINCG (N, A, LDA, AINV, LDAINV)
C                                  Print results
      CALL WRCRN ('AINV', N, N, AINV, LDAINV, 0)
      END
```

Output

```
                         AINV
                  1                 2                 3
1  ( 6.400,-2.800)  (-3.800, 2.600)  (-2.600, 1.200)
2  (-1.600,-1.800)  ( 0.200, 0.600)  ( 0.400,-0.800)
3  (-0.600, 2.200)  ( 1.200,-1.400)  ( 0.400, 0.200)
```

LSLRT/DLSLRT (Single/Double precision)

Solve a real triangular system of linear equations.

Usage

`CALL LSLRT (N, A, LDA, B, IPATH, X)`

Arguments

N — Number of equations.   (Input)

A — `N` by `N` matrix containing the coefficient matrix for the triangular linear system.   (Input)
For a lower triangular system, only the lower triangular part and diagonal of `A` are referenced. For an upper triangular system, only the upper triangular part and diagonal of `A` are referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator.   (Input)
`IPATH` = 1 means solve `AX` = `B`, `A` lower triangular.
`IPATH` = 2 means solve `AX` = `B`, `A` upper triangular.
`IPATH` = 3 means solve A^TX = `B`, `A` lower triangular.
`IPATH` = 4 means solve A^TX = `B`, `A` upper triangular.

X — Vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X can share the same storage locations.

Algorithm

Routine LSLRT solves a system of linear algebraic equations with a real triangular coefficient matrix. LSLRT fails if the matrix A has a zero diagonal element, in which case A is singular. LSLRT is based on the LINPACK routine STRSL; see Dongarra et al. (1979).

Example

A system of three linear equations is solved. The coefficient matrix has lower triangular form and the right-hand-side vector, *b*, has three elements.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      REAL       A(LDA,LDA), B(LDA), X(LDA)
C                                 Set values for A and B
C
C                                 A = (  2.0               )
C                                     (  2.0    -1.0       )
C                                     ( -4.0     2.0    5.0)
C
C                                 B = (  2.0     5.0    0.0)
C
      DATA A/2.0, 2.0, -4.0, 0.0, -1.0, 2.0, 0.0, 0.0, 5.0/
      DATA B/2.0, 5.0, 0.0/
C
C                                 Solve AX = B     (IPATH = 1)
      CALL LSLRT (N, A, LDA, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
           X
    1       2       3
1.000  -3.000   2.000
```

LFCRT/DLFCRT (Single/Double precision)

Estimate the condition number of a real triangular matrix.

Usage

CALL LFCRT (N, A, LDA, IPATH, RCOND)

Arguments

N — Order of the matrix. (Input)

A **—** N by N matrix containing the triangular matrix whose condition number is to be estimated. (Input)

For a lower triangular matrix, only the lower triangular part and diagonal of `A` are referenced. For an upper triangular matrix, only the upper triangular part and diagonal of `A` are referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means `A` is lower triangular.
`IPATH` = 2 means `A` is upper triangular.

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCRT` `N` units, or
   `DLFCRT` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CRT`/`DL2CRT`. The reference is

   ```
   CALL L2CRT (N, A, LDA, IPATH, RCOND, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational error

Type	Code	
3	1	The input triangular matrix is algorithmically singular.

Algorithm

Routine `LFCRT` estimates the condition number of a real triangular matrix. The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in `A` can cause very large changes in the solution *x*.

`LFCRT` is based on the LINPACK routine `STRCO`; see Dongarra et al. (1979).

Example

An estimate of the reciprocal condition number is computed for a 3 × 3 lower triangular coefficient matrix.

```
C                                 Declare variables
      PARAMETER  (IPATH=1, LDA=3, N=3)
      REAL       A(LDA,LDA), RCOND
      INTEGER    NOUT
C                                 Set values for A and B
C                                 A = (  2.0              )
C                                     (  2.0    -1.0      )
C                                     ( -4.0     2.0    5.0)
C
      DATA A/2.0, 2.0, -4.0, 0.0, -1.0, 2.0, 0.0, 0.0, 5.0/
C
C                                 Compute the reciprocal condition
C                                 number  (IPATH=1)
      CALL LFCRT (N, A, LDA, IPATH, RCOND)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.091
L1 Condition number = 10.968
```

LFDRT/DLFDRT (Single/Double precision)

Compute the determinant of a real triangular matrix.

Usage

`CALL LFDRT (N, A, LDA, DET1, DET2)`

Arguments

N – Order of the matrix `A`. (Input)

A — `N` by `N` matrix containing the triangular matrix. (Input)
The matrix can be either upper or lower triangular.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value `DET1` is normalized so that $1.0 \le |\mathtt{DET1}| < 10.0$ or `DET1` = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det(`A`) = `DET1` * $10^{\mathtt{DET2}}$.

Comments

Informational error

Type	Code	
3	1	The input triangular matrix is singular.

Algorithm

Routine LFDRT computes the determinant of a real triangular coefficient matrix. The determinant of a triangular matrix is the product of the diagonal elements .

$$\det A = \prod\nolimits_{i=1}^{N} A_{ii}$$

LFDRT is based on the LINPACK routine STRDI; see Dongarra et al. (1979).

Example

The determinant is computed for a 3 × 3 lower triangular matrix.

```
C                                  Declare variables
      PARAMETER  (LDA=3, N=3)
      REAL       A(LDA,LDA), DET1, DET2
      INTEGER    NOUT
C                                  Set values for  A
C                                  A = (  2.0               )
C                                      (  2.0    -1.0       )
C                                      ( -4.0     2.0    5.0)
C
      DATA A/2.0, 2.0, -4.0, 0.0, -1.0, 2.0, 0.0, 0.0, 5.0/
C
C                                  Compute the determinant of A
      CALL LFDRT (N, A, LDA, DET1, DET2)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
99999 FORMAT (’ The determinant of A is ’, F6.3, ’ * 10**’, F2.0)
      END
```

Output

```
The determinant of A is -1.000 * 10**1.
```

LINRT/DLINRT (Single/Double precision)

Compute the inverse of a real triangular matrix.

Usage

CALL LINRT (N, A, LDA, IPATH, AINV, LDAINV)

Arguments

N — Order of the matrix.   (Input)

A — N by N matrix containing the triangular matrix to be inverted.   (Input)
For a lower triangular matrix, only the lower triangular part and diagonal of A are referenced. For an upper triangular matrix, only the upper triangular part and diagonal of A are referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

IPATH — Path indicator.   (Input)
IPATH = 1 means A is lower triangular.
IPATH = 2 means A is upper triangular.

AINV — N by N matrix containing the inverse of A.   (Output)
If A is lower triangular, AINV is also lower triangular. If A is upper triangular, AINV is also upper triangular. If A is not needed, A and AINV can share the same storage locations.

LDAINV — Leading dimension of AINV exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

Routine LINRT computes the inverse of a real triangular matrix. It fails if A has a zero diagonal element.

Example

The inverse is computed for a 3 × 3 lower triangular matrix.

```
C                                 Declare variables
      PARAMETER  (LDA=3, LDAINV=3, N=3)
      REAL       A(LDA,LDA), AINV(LDA,LDA)
C                                 Set values for  A
C                                 A = (  2.0              )
C                                     (  2.0    -1.0      )
C                                     ( -4.0     2.0    5.0)
C
      DATA A/2.0, 2.0, -4.0, 0.0, -1.0, 2.0, 0.0, 0.0, 5.0/
C
C                                 Compute the inverse of A
      IPATH = 1
      CALL LINRT (N, A, LDA, IPATH, AINV, LDAINV)
C                                 Print results
      CALL WRRRN ('AINV', N, N, AINV, LDA, 0)
      END
```

Output

```
           AINV
        1       2       3
1   0.500   0.000   0.000
2   1.000  -1.000   0.000
3   0.000   0.400   0.200
```

LSLCT/DLSLCT (Single/Double precision)

Solve a complex triangular system of linear equations.

Usage

```
CALL LSLCT (N, A, LDA, B, IPATH, X)
```

Arguments

N — Number of equations. (Input)

A — Complex N by N matrix containing the coefficient matrix of the triangular linear system. (Input)
For a lower triangular system, only the lower triangle of A is referenced. For an upper triangular system, only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means solve AX = B, A lower triangular
IPATH = 2 means solve AX = B, A upper triangular
IPATH = 3 means solve $A^H X$ = B, A lower triangular
IPATH = 4 means solve $A^H X$ = B, A upper triangular

X — Complex vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X can share the same storage locations.

Comments

Informational error

Type	Code	
4	1	The input triangular matrix is singular. Some of its diagonal elements are near zero.

Algorithm

Routine LSLCT solves a system of linear algebraic equations with a complex triangular coefficient matrix. LSLCT fails if the matrix A has a zero diagonal element, in which case A is singular. LSLCT is based on the LINPACK routine CTRSL; see Dongarra et al. (1979).

Example

A system of three linear equations is solved. The coefficient matrix has lower triangular form and the right-hand-side vector, *b*, has three elements.

```
C                                  Declare variables
      INTEGER IPATH, LDA, N
      PARAMETER  (LDA=3, N=3)
      COMPLEX    A(LDA,LDA), B(LDA), X(LDA)
C                                 Set values for A and B
C
C                                 A = ( -3.0+2.0i                    )
C                                     ( -2.0-1.0i  0.0+6.0i          )
C                                     ( -1.0+3.0i  1.0-5.0i -4.0+0.0i )
C
C                                 B = (-13.0+0.0i -10.0-1.0i -11.0+3.0i)
```

```
C
      DATA A/(-3.0,2.0), (-2.0,-1.0), (-1.0, 3.0), (0.0,0.0), (0.0,6.0),
     &       (1.0,-5.0), (0.0,0.0), (0.0,0.0), (-4.0,0.0)/
      DATA B/(-13.0,0.0), (-10.0,-1.0), (-11.0,3.0)/
C
C                                Solve AX = B
      IPATH = 1
      CALL LSLCT (N, A, LDA, B, IPATH, X)
C                                Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                          X
              1                 2                 3
( 3.000, 2.000)  ( 1.000, 1.000)  ( 2.000, 0.000)
```

LFCCT/DLFCCT (Single/Double precision)

Estimate the condition number of a complex triangular matrix.

Usage

`CALL LFCCT (N, A, LDA, IPATH, RCOND)`

Arguments

N — Order of the matrix.   (Input)

A — Complex `N` by `N` matrix containing the triangular matrix.   (Input)
For a lower triangular system, only the lower triangle of `A` is referenced. For an upper triangular system, only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

IPATH — Path indicator.   (Input)
`IPATH` = 1 means `A` is lower triangular.
`IPATH` = 2 means `A` is upper triangular.

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`.   (Output)

Comments

1. Automatic workspace usage is

   `LFCCT`   2`N` units, or
   `DLFCCT` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CCT/DL2CCT`. The reference is

```
CALL L2CCT (N, A, LDA, IPATH, RCOND, CWK)
```

The additional argument is

CWK — Complex work vector of length `N`.

2. Informational error

Type	Code	
3	1	The input triangular matrix is algorithmically singular.

Algorithm

Routine `LFCCT` estimates the condition number of a complex triangular matrix. The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979). If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. `LFCCT` is based on the LINPACK routine `CTRCO`; see Dongarra et al. (1979).

Example

An estimate of the reciprocal condition number is computed for a 3 × 3 lower triangular coefficient matrix.

```
C                                  Declare variables
      INTEGER IPATH, LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    NOUT
      REAL       RCOND
      COMPLEX    A(LDA,LDA)
C                                 Set values for A
C
C                                 A = ( -3.0+2.0i                     )
C                                     ( -2.0-1.0i  0.0+6.0i           )
C                                     ( -1.0+3.0i  1.0-5.0i -4.0+0.0i )
C
      DATA A/(-3.0,2.0), (-2.0,-1.0), (-1.0, 3.0), (0.0,0.0), (0.0,6.0),
     &       (1.0,-5.0), (0.0,0.0), (0.0,0.0), (-4.0,0.0)/
C
C                                  Compute the reciprocal condition
C                                  number
      IPATH = 1
      CALL LFCCT (N, A, LDA, IPATH, RCOND)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.191
L1 Condition number = 5.223
```

LFDCT/DLFDCT (Single/Double precision)

Compute the determinant of a complex triangular matrix.

Usage

```
CALL LFDCT (N, A, LDA, DET1, DET2)
```

Arguments

N — Order of the matrix `A`. (Input)

A — Complex `N` by `N` matrix containing the triangular matrix.(Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

DET1 — Complex scalar containing the mantissa of the determinant. (Output)
The value `DET1` is normalized so that $1.0 \le |\mathrm{DET1}| < 10.0$ or `DET1` = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form $\det(\mathrm{A}) = \mathrm{DET1} * 10^{\mathrm{DET2}}$.

Comments

Informational error

Type	Code	
3	1	The input triangular matrix is singular.

Algorithm

Routine `LFDCT` computes the determinant of a complex triangular coefficient matrix. The determinant of a triangular matrix is the product of the diagonal elements

$$\det A = \prod_{i=1}^{N} A_{ii}$$

`LFDCT` is based on the LINPACK routine `CTRDI`; see Dongarra et al. (1979).

Example

The determinant is computed for a 3 × 3 complex lower triangular matrix.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    NOUT
      REAL       DET2
      COMPLEX    A(LDA,LDA), DET1
```

```
C                                 Set values for A
C
C                                 A = ( -3.0+2.0i                    )
C                                     ( -2.0-1.0i  0.0+6.0i          )
C                                     ( -1.0+3.0i  1.0-5.0i -4.0+0.0i )
C
      DATA A/(-3.0,2.0), (-2.0,-1.0), (-1.0, 3.0), (0.0,0.0), (0.0,6.0),
     &       (1.0,-5.0), (0.0,0.0), (0.0,0.0), (-4.0,0.0)/
C
C                                  Compute the determinant of A
      CALL LFDCT (N, A, LDA, DET1, DET2)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
99999 FORMAT (’ The determinant of A is (’,F4.1,’,’,F4.1,’) * 10**’,
     &        F2.0)
      END
```

Output

```
The determinant of A is ( 0.5, 0.7) * 10**2.
```

LINCT/DLINCT (Single/Double precision)

Compute the inverse of a complex triangular matrix.

Usage

```
CALL LINCT (N, A, LDA, IPATH, AINV, LDAINV)
```

Arguments

N — Order of the matrix. (Input)

A — Complex `N` by `N` matrix containing the triangular matrix to be inverted. (Input)
For a lower triangular matrix, only the lower triangle of `A` is referenced. For an upper triangular matrix, only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means `A` is lower triangular.
`IPATH` = 2 means `A` is upper triangular.

AINV — Complex `N` by `N` matrix containing the inverse of `A`. (Output)
If `A` is lower triangular, `AINV` is also lower triangular. If `A` is upper triangular, `AINV` is also upper triangular. If `A` is not needed, `A` and `AINV` can share the same storage locations.

LDAINV — Leading dimension of `AINV` exactly as specified in the dimension statement of the calling program. (Input)

Comments

Informational error

Type	Code	
4	1	The input triangular matrix is singular. Some of its diagonal elements are close to zero.

Algorithm

Routine `LINCT` computes the inverse of a complex triangular matrix. It fails if `A` has a zero diagonal element.

Example

The inverse is computed for a 3 × 3 lower triangular matrix.

```
C                                 Declare variables
      INTEGER IPATH, LDA, LDAINV, N
      PARAMETER  (LDA=3, N=3, LDAINV=3)
      COMPLEX    A(LDA,LDA), AINV(LDA,LDA)
C                                Set values for A
C
C                                A = ( -3.0+2.0i                    )
C                                    ( -2.0-1.0i  0.0+6.0i          )
C                                    ( -1.0+3.0i  1.0-5.0i -4.0+0.0i )
C
      DATA A/(-3.0,2.0), (-2.0,-1.0), (-1.0, 3.0), (0.0,0.0), (0.0,6.0),
     &       (1.0,-5.0), (0.0,0.0), (0.0,0.0), (-4.0,0.0)/
C
C                                 Compute the inverse of A
      IPATH = 1
      CALL LINCT (N, A, LDA, IPATH, AINV, LDAINV)
C                                 Print results
      CALL WRCRN ('AINV', N, N, AINV, LDAINV, 0)
      END
```

Output

```
                           AINV
                 1                   2                   3
1  (-0.2308,-0.1538)  ( 0.0000, 0.0000)  ( 0.0000, 0.0000)
2  (-0.0897, 0.0513)  ( 0.0000,-0.1667)  ( 0.0000, 0.0000)
3  ( 0.2147,-0.0096)  (-0.2083,-0.0417)  (-0.2500, 0.0000)
```

LSADS/DLSADS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations with iterative refinement.

Usage

`CALL LSADS (N, A, LDA, B, X)`

Arguments

N — Number of equations. (Input)

A — N by N matrix containing the coefficient matrix of the symmetric positive definite linear system. (Input)
Only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

X — Vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSADS $N^2 + N$ units, or
   DLSADS $2N^2 + 2N$ units.

   Workspace may be explicitly provided, if desired, by use of L2ADS/DL2ADS. The reference is

   ```
   CALL L2ADS (N, A, LDA, B, X, FAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length N^2 containing the $R^T R$ factorization of A on output.

   WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is not positive definite.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2ADS the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSADS. Additional memory allocation for FAC and option value restoration are done automatically in LSADS. Users directly calling L2ADS can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSADS or L2ADS.

Default values for the option are
IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSADS temporarily replaces IVAL(2) by IVAL(1). The routine L2CDS computes the condition number if IVAL(2) = 2. Otherwise L2CDS skips this computation. LSADS restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSADS solves a system of linear algebraic equations having a real symmetric positive definite coefficient matrix. It first uses the routine LFCDS, page 61, to compute an $R^T R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. The matrix *R* is upper triangular. The solution of the linear system is then found using the iterative refinement routine LFIDS, page 67. LSADS fails if any submatrix of *R* is not positive definite, if *R* has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if *A* is either very close to a singular matrix or a matrix which is not positive definite. If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSADS solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of three linear equations is solved. The coefficient matrix has real positive definite form and the right-hand-side vector *b* has three elements.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                  Set values for A and B
C
C                                  A = (  1.0  -3.0   2.0)
C                                      ( -3.0  10.0  -5.0)
C                                      (  2.0  -5.0   6.0)
C
C                                  B = ( 27.0 -78.0  64.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
      DATA B/27.0, -78.0, 64.0/
C
      CALL LSADS (N, A, LDA, B, X)
C                                  Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
C
```

```
END
```

Output

```
            X
    1       2       3
1.000  -4.000   7.000
```

LSLDS/DLSLDS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations without iterative refinement.

Usage

```
CALL LSLDS (N, A, LDA, B, X)
```

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the coefficient matrix of the symmetric positive definite linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Comments

1. Automatic workspace usage is

   `LSLDS` $N^2 + N$ units, or
   `DLSLDS` $2N^2 + 2N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2LDS/DL2LDS`. The reference is

   ```
   CALL L2LDS (N, A, LDA, B, X, FAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length N^2 containing the $R^T R$ factorization of `A` on output. If `A` is not needed, `A` can share the same storage locations as `FAC`.

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is not positive definite.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LDS the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSLDS. Additional memory allocation for FAC and option value restoration are done automatically in LSLDS. Users directly calling L2LDS can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLDS or L2LDS. Default values for the option are
IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLDS temporarily replaces IVAL(2) by IVAL(1). The routine L2CDS computes the condition number if IVAL(2) = 2. Otherwise L2CDS skips this computation. LSLDS restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLDS solves a system of linear algebraic equations having a real symmetric positive definite coefficient matrix. It first uses the routine LFCDS, page 61, to compute an $R^T R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. The matrix R is upper triangular. The solution of the linear system is then found using the routine LFSDS, page 65. LSLDS fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A either is very close to a singular matrix or to a matrix which is not positive definite. If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. If the coefficient matrix is ill-conditioned, it is recommended that LSADS, page 56, be used.

Example

A system of three linear equations is solved. The coefficient matrix has real positive definite form and the right-hand-side vector b has three elements.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                  Set values for A and B
C
C                                  A = (  1.0  -3.0   2.0)
C                                      ( -3.0  10.0  -5.0)
C                                      (  2.0  -5.0   6.0)
C
C                                  B = ( 27.0 -78.0  64.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
      DATA B/27.0, -78.0, 64.0/
C
      CALL LSLDS (N, A, LDA, B, X)
C                                  Print results
      CALL WRRRN ('X', 1, N, X, 1, 0)
C
      END
```

Output

```
          X
    1       2       3
1.000  -4.000   7.000
```

LFCDS/DLFCDS (Single/Double precision)

Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix and estimate its L_1 condition number.

Usage

`CALL LFCDS (N, A, LDA, FAC, LDFAC, RCOND)`

Arguments

N — Order of the matrix. (Input)

A — `N` by `N` symmetric positive definite matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — `N` by `N` matrix containing the upper triangular matrix *R* of the factorization of `A` in the upper triangular part. (Output)
Only the upper triangle of `FAC` will be used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCDS` `N` units, or
   `DLFCDS` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CDS`/`DL2CDS`. The reference is

   ```
   CALL L2CDS (N, A, LDA, FAC, LDFAC, RCOND, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is not positive definite.

Algorithm

Routine `LSADS` computes an R^TR Cholesky factorization and estimates the condition number of a real symmetric positive definite coefficient matrix. The matrix *R* is upper triangular.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system.

`LFCDS` fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* is very close to a singular matrix or to a matrix which is not positive definite.

The R^TR factors are returned in a form that is compatible with routines `LFIDS`, page 67, `LFSDS`, page 65, and `LFDDS`, page 69. To solve systems of equations with multiple right-hand-side vectors, use `LFCDS` followed by either `LFIDS` or `LFSDS` called once for each right-hand side. The routine `LFDDS` can be called to compute the determinant of the coefficient matrix after `LFCDS` has performed the factorization.

`LFCDS` is based on the LINPACK routine `SPOCO`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 matrix is computed. `LFCDS` is called to factor the matrix and to check for nonpositive definiteness or ill-conditioning. `LFIDS` (page 67) is called to determine the columns of the inverse.

```
C                                  Declare variables
      INTEGER    LDA, LDFAC, N, NOUT
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), AINV(LDA,LDA), COND, FAC(LDFAC,LDFAC),
     &           RES(N), RJ(N)
C
C                                  Set values for A
C                                  A = (  1.0  -3.0   2.0)
C                                      ( -3.0  10.0  -5.0)
C                                      (  2.0  -5.0   6.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C                                  Factor the matrix A
      CALL LFCDS (N, A, LDA, FAC, LDFAC, COND)
C                                  Set up the columns of the identity
C                                  matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                  RJ is the J-th column of the identity
C                                  matrix so the following LFIDS
C                                  reference places the J-th column of
C                                  the inverse of A in the J-th column
C                                  of AINV
         CALL LFIDS (N, A, LDA, FAC, LDFAC, RJ, AINV(1,J), RES)
         RJ(J) = 0.0E0
   10 CONTINUE
C                                  Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) COND, 1.0E0/COND
      CALL WRRRN (’AINV’, N, N, AINV, LDA, 0)
99999 FORMAT (’  COND = ’,F5.3,/,’  L1 Condition number = ’,F9.3)
      END
```

Output

```
COND = 0.001
L1 Condition number = 673.839

          AINV
        1       2       3
1   35.00    8.00   -5.00
2    8.00    2.00   -1.00
3   -5.00   -1.00    1.00
```

LFTDS/DLFTDS (Single/Double precision)

Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix.

Usage

```
CALL LFTDS (N, A, LDA, FAC, LDFAC)
```

Arguments

N — Order of the matrix.   (Input)

A — `N` by `N` symmetric positive definite matrix to be factored.   (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

FAC — `N` by `N` matrix containing the upper triangular matrix *R* of the factorization of `A` in the upper triangle.   (Output)
Only the upper triangle of `FAC` will be used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program.   (Input)

Comments

Informational error

Type	Code	
4	2	The input matrix is not positive definite.

Algorithm

Routine `LFTDS` computes an $R^T R$ Cholesky factorization of a real symmetric positive definite coefficient matrix. The matrix *R* is upper triangular.

`LFTDS` fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* is very close to a singular matrix or to a matrix which is not positive definite.

The $R^T R$ factors are returned in a form that is compatible with routines `LFIDS`, page 67, `LFSDS`, page 65, and `LFDDS`, page 69. To solve systems of equations with multiple right-hand-side vectors, use `LFTDS` followed by either `LFIDS` or `LFSDS` called once for each right-hand side. The routine `LFDDS` can be called to compute the determinant of the coefficient matrix after `LFTDS` has performed the factorization.

`LFTDS` is based on the LINPACK routine `SPOFA`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 matrix is computed. LFTDS is called to factor the matrix and to check for nonpositive definiteness. LFSDS (page 65) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), AINV(LDA,LDA), FAC(LDFAC,LDFAC), RJ(N)
C
C                                 Set values for A
C                                 A = (  1.0  -3.0   2.0)
C                                     ( -3.0  10.0  -5.0)
C                                     (  2.0  -5.0   6.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C                                 Factor the matrix A
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSDS
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSDS (N, FAC, LDFAC, RJ, AINV(1,J))
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print the results
      CALL WRRRN (’AINV’, N, N, AINV, LDA, 0)
C
      END
```

Output

```
           AINV
        1       2       3
1   35.00    8.00   -5.00
2    8.00    2.00   -1.00
3   -5.00   -1.00    1.00
```

LFSDS/DLFSDS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations given the $R^T R$ Cholesky factorization of the coefficient matrix.

Usage

```
CALL LFSDS (N, FAC, LDFAC, B, X)
```

Arguments

N — Number of equations. (Input)

FAC — N by N matrix containing the $R^T R$ factorization of the coefficient matrix A as output from routine LFCDS/DLFCDS or LFTDS/DLFTDS. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

X — Vector of length N containing the solution to the linear system. (Output) If B is not needed, B and X can share the same storage locations.

Comments

Informational error

Type	Code	
4	1	The input matrix is singular.

Algorithm

This routine computes the solution for a system of linear algebraic equations having a real symmetric positive definite coefficient matrix. To compute the solution, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either LFCDS, page 61, or LFTDS, page 63. R is an upper triangular matrix.

The solution to $Ax = b$ is found by solving the triangular systems $R^T y = b$ and $Rx = y$.

LFSDS and LFIDS, page 67, both solve a linear system given its $R^T R$ factorization. LFIDS generally takes more time and produces a more accurate answer than LFSDS. Each iteration of the iterative refinement algorithm used by LFIDS calls LFSDS.

LFSDS is based on the LINPACK routine SPOSL; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. LFTDS (page 63) is called to factor the coefficient matrix. LFSDS is called to compute the four solutions for the four right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call LFCDS (page 61) to perform the factorization, and LFIDS (page 67) to compute the solutions.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), B(N,4), FAC(LDFAC,LDFAC), X(N,4)
C
C                                Set values for A and B
C
C                                A = (  1.0  -3.0   2.0)
C                                    ( -3.0  10.0  -5.0)
C                                    (  2.0  -5.0   6.0)
C
C                                B = ( -1.0   3.6  -8.0  -9.4)
C                                    ( -3.0  -4.2  11.0  17.6)
C                                    ( -3.0  -5.2  -6.0 -23.4)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
      DATA B/-1.0, -3.0, -3.0, 3.6, -4.2, -5.2, -8.0, 11.0, -6.0,
     &     -9.4, 17.6, -23.4/
C                                Factor the matrix A
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
C                                Compute the solutions
      DO 10  I=1, 4
         CALL LFSDS (N, FAC, LDFAC, B(1,I), X(1,I))
   10 CONTINUE
C                                Print solutions
      CALL WRRRN ('The solution vectors are', N, 4, X, N, 0)
C
      END
```

Output

```
     The solution vectors are
        1       2       3       4
1   -44.0   118.4  -162.0   -71.2
2   -11.0    25.6   -36.0   -16.6
3     5.0   -19.0    23.0     6.0
```

LFIDS/DLFIDS (Single/Double precision)

Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations.

Usage

```
CALL LFIDS (N, A, LDA, FAC, LDFAC, B, X, RES)
```

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the symmetric positive definite coefficient matrix of the linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — `N` by `N` matrix containing the $R^T R$ factorization of the coefficient matrix `A` as output from routine `LFCDS`/`DLFCDS` or `LFTDS`/`DLFTDS`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)

RES — Vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	2	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIDS` computes the solution of a system of linear algebraic equations having a real symmetric positive definite coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either `LFCDS`, page 61, or `LFTDS`, page 63.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIDS` and `LFSDS`, page 65, both solve a linear system given its $R^T R$ factorization. `LFIDS` generally takes more time and produces a more accurate answer than `LFSDS`. Each iteration of the iterative refinement algorithm used by `LFIDS` calls `LFSDS`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.2 to the second element.

```
C                                Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), B(N), COND, FAC(LDFAC,LDFAC), RES(N,3),
     &           X(N,3)
C
C                                Set values for A and B
```

```
C
C                                 A = (  1.0  -3.0   2.0)
C                                     ( -3.0  10.0  -5.0)
C                                     (  2.0  -5.0   6.0)
C
C                                 B = (  1.0  -3.0   2.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
      DATA B/1.0, -3.0, 2.0/
C                                 Factor the matrix A
      CALL LFCDS (N, A, LDA, FAC, LDFAC, COND)
C                                 Print the estimated condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) COND, 1.0E0/COND
C                                 Compute the solutions
      DO 10  I=1, 3
         CALL LFIDS (N, A, LDA, FAC, LDFAC, B, X(1,I), RES(1,I))
         B(2) = B(2) + .2E0
   10 CONTINUE
C                                 Print solutions and residuals
      CALL WRRRN (’The solution vectors are’, N, 3, X, N, 0)
      CALL WRRRN (’The residual vectors are’, N, 3, RES, N, 0)
C
99999 FORMAT (’  COND = ’,F5.3,/,’  L1 Condition number = ’,F9.3)
      END
```

Output

```
COND = 0.001
L1 Condition number =   673.839

The solution vectors are
        1       2       3
1   1.000   2.600   4.200
2   0.000   0.400   0.800
3   0.000  -0.200  -0.400

The residual vectors are
         1        2        3
1   0.0000   0.0000   0.0000
2   0.0000   0.0000   0.0000
3   0.0000   0.0000   0.0000
```

LFDDS/DLFDDS (Single/Double precision)

Compute the determinant of a real symmetric positive definite matrix given the $R^T R$ Cholesky factorization of the matrix.

Usage

```
CALL LFDDS (N, FAC, LDFAC, DET1, DET2)
```

Arguments

N — Order of the matrix.   (Input)

FAC — N by N matrix containing the $R^T R$ factorization of the coefficient matrix A as output from routine LFCDS/DLFCDS or LFTDS/DLFTDS. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that, 1.0 ≤ |DET1| < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form, det(A) = DET1 * 10^{DET2}.

Algorithm

Routine LFDDS computes the determinant of a real symmetric positive definite coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either LFCDS, page 61, or LFTDS, page 63. The formula det A = det R^T det R = (det R)^2 is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det R = \prod\nolimits_{i=1}^{N} R_{ii}$$

(The matrix R is stored in the upper triangle of FAC.)

LFDDS is based on the LINPACK routine SPODI; see Dongarra et al. (1979).

Example

The determinant is computed for a real positive definite 3 × 3 matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NOUT
      PARAMETER  (LDA=3, N=3, LDFAC=3)
      REAL       A(LDA,LDA), DET1, DET2, FAC(LDFAC,LDFAC)
C
C                                 Set values for A
C                                 A = (  1.0  -3.0   2.0)
C                                     ( -3.0  20.0  -5.0)
C                                     (  2.0  -5.0   6.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 20.0, -5.0, 2.0, -5.0, 6.0/
C                                 Factor the matrix
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
C                                 Compute the determinant
      CALL LFDDS (N, FAC, LDFAC, DET1, DET2)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant of A is ’,F6.3,’ * 10**’,F2.0)
      END
```

Output

```
The determinant of A is 2.100 * 10**1.
```

LINDS/DLINDS (Single/Double precision)

Compute the inverse of a real symmetric positive definite matrix.

Usage

```
CALL LINDS (N, A, LDA, AINV, LDAINV)
```

Arguments

N — Order of the matrix `A`.   (Input)

A — `N` by `N` matrix containing the symmetric positive definite matrix to be inverted.   (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

AINV — `N` by `N` matrix containing the inverse of `A`. (Output)
If `A` is not needed, `A` and `AINV` can share the same storage locations.

LDAINV — Leading dimension of `AINV` exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   `LINDS` `N` units, or
   `DLINDS` `2N` units.

   Workspace may be explicitly provided, if desired, by use of `L2NDS/DL2NDS`. The reference is

   ```
   CALL L2NDS (N, A, LDA, AINV, LDAINV, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is not positive definite.

Algorithm

Routine `LINDS` computes the inverse of a real symmetric positive definite matrix. It first uses the routine `LFCDS`, page 61, to compute an $R^T R$ factorization of the coefficient matrix and to estimate the condition number of the matrix. `LINRT`, page 49, is then used to compute R^{-1}. Finally A^{-1} is computed using $R^{-1} = R^{-1} R^{-T}$.

`LINDS` fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in A^{-1}.

Example

The inverse is computed for a real positive definite 3 × 3 matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDAINV, N
      PARAMETER  (LDA=3, LDAINV=3, N=3)
      REAL       A(LDA,LDA), AINV(LDAINV,LDAINV)
C
C                                  Set values for A
C                                  A = (  1.0  -3.0   2.0)
C                                      ( -3.0  10.0  -5.0)
C                                      (  2.0  -5.0   6.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C
      CALL LINDS (N, A, LDA, AINV, LDAINV)
C                                  Print results
      CALL WRRRN ('AINV', N, N, AINV, LDAINV, 0)
C
      END
```

Output

```
           AINV
        1        2        3
1   35.00     8.00    -5.00
2    8.00     2.00    -1.00
3   -5.00    -1.00     1.00
```

LSASF/DLSASF (Single/Double precision)

Solve a real symmetric system of linear equations with iterative refinement.

Usage

`CALL LSASF (N, A, LDA, B, X)`

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the coefficient matrix of the symmetric linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSASF` $N^2 + 2N$ units, or
   `DLSASF` $2N^2 + 3N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2ASF`/`DL2ASF`. The reference is

   ```
   CALL L2ASF (N, A, LDA, B, X, FAC, IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length `N * N` containing information about the $U\,DU^T$ factorization of `A` on output. If `A` is not needed, `A` and `FAC` can share the same storage location.

   IPVT — Integer work vector of length `N` containing the pivoting information for the factorization of `A` on output.

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine `L2ASF` the leading dimension of `FAC` is increased by `IVAL`(3) when `N` is a multiple of `IVAL`(4). The values `IVAL`(3) and `IVAL`(4) are temporarily replaced by `IVAL`(1) and `IVAL`(2), respectively, in `LSASF`. Additional memory allocation for `FAC` and option value restoration are done automatically in `LSASF`. Users directly calling L2ASF can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSASF or L2ASF. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSASF temporarily replaces IVAL(2) by IVAL(1). The routine L2CSF computes the condition number if IVAL(2) = 2. Otherwise L2CSF skips this computation. LSASF restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSASF solves systems of linear algebraic equations having a real symmetric indefinite coefficient matrix. It first uses the routine LFCSF, page 77, to compute a $U\,DU^T$ factorization of the coefficient matrix and to estimate the condition number of the matrix. D is a block diagonal matrix with blocks of order 1 or 2, and U is a matrix composed of the product of a permutation matrix and a unit upper triangular matrix. The solution of the linear system is then found using the iterative refinement routine LFISF, page 83.

LSASF fails if a block in D is singular or if the iterative refinement algorithm fails to converge. These errors occur only if A is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system. LSASF solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of three linear equations is solved. The coefficient matrix has real symmetric form and the right-hand-side vector b has three elements.

```
C                                 Declare variables
      PARAMETER  (LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                 Set values for A and B
C
C                                 A = (  1.0  -2.0   1.0)
C                                     ( -2.0   3.0  -2.0)
C                                     (  1.0  -2.0   3.0)
C
C                                 B = (  4.1  -4.7   6.5)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
      DATA B/4.1, -4.7, 6.5/
C
      CALL LSASF (N, A, LDA, B, X)
C                                Print results
      CALL WRRRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
            X
     1        2        3
-4.100   -3.500    1.200
```

LSLSF/DLSLSF (Single/Double precision)

Solve a real symmetric system of linear equations without iterative refinement.

Usage

`CALL LSLSF (N, A, LDA, B, X)`

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the coefficient matrix of the symmetric linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSLSF` $N^2 + 2N$ units, or
   `DLSLSF` $2N^2 + 3N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2LSF`/`DL2LSF`. The reference is

   `CALL L2LSF (N, A, LDA, B, X, FAC, IPVT, WK)`

   The additional arguments are as follows:

   FAC — Work vector of length N^2 containing information about the $U\,DU^T$ factorization of `A` on output. If `A` is not needed, `A` and `FAC` can share the same storage locations.

IPVT — Integer work vector of length N containing the pivoting information for the factorization of A on output.

WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine LSLSF the leading dimension of FAC is increased by IVAL(3) when *N* is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSLSF. Additional memory allocation for FAC and option value restoration are done automatically in LSLSF. Users directly calling LSLSF can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLSF or LSLSF. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLSF temporarily replaces IVAL(2) by IVAL(1). The routine L2CSF computes the condition number if IVAL(2) = 2. Otherwise L2CSF skips this computation. LSLSF restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLSF solves systems of linear algebraic equations having a real symmetric indefinite coefficient matrix. It first uses the routine LFCSF, page 77, to compute a $U\,DU^T$ factorization of the coefficient matrix. *D* is a block diagonal matrix with blocks of order 1 or 2, and *U* is a matrix composed of the product of a permutation matrix and a unit upper triangular matrix.

The solution of the linear system is then found using the routine LFSSF, page 81.

LSLSF fails if a block in *D* is singular. This occurs only if *A* either is singular or is very close to a singular matrix.

Example

A system of three linear equations is solved. The coefficient matrix has real symmetric form and the right-hand-side vector *b* has three elements.

```
C                                  Declare variables
      PARAMETER  (LDA=3, N=3)
      REAL       A(LDA,LDA), B(N), X(N)
C
C                                  Set values for A and B
C
C                                  A = (  1.0  -2.0   1.0)
C                                      ( -2.0   3.0  -2.0)
C                                      (  1.0  -2.0   3.0)
C
C                                  B = (  4.1  -4.7   6.5)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
      DATA B/4.1, -4.7, 6.5/
C
      CALL LSLSF (N, A, LDA, B, X)
C                                  Print results
      CALL WRRRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
            X
     1       2       3
-4.100  -3.500   1.200
```

LFCSF/DLFCSF (Single/Double precision)

Compute the $U DU^T$ factorization of a real symmetric matrix and estimate its L_1 condition number.

Usage

`CALL LFCSF (N, A, LDA, FAC, LDFAC, IPVT, RCOND)`

Arguments

N — Order of the matrix. (Input)

A — `N` by `N` symmetric matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — `N` by `N` matrix containing information about the factorization of the symmetric matrix `A`. (Output)
Only the upper triangle of `FAC` is used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCSF` `N` units, or
   `DLFCSF` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CSF`/`DL2CSF`. The reference is

   ```
   CALL L2CSF (N, A, LDA, FAC, LDFAC, IPVT, RCOND, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is singular.

Algorithm

Routine `LFCSF` performs a $U\,DU^T$ factorization of a real symmetric indefinite coefficient matrix. It also estimates the condition number of the matrix. The $U\,DU^T$ factorization is called the diagonal pivoting factorization.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in `A` can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

`LFCSF` fails if `A` is singular or very close to a singular matrix.

The $U\,DU^T$ factors are returned in a form that is compatible with routines `LFISF`, page 83, `LFSSF`, page 81, and `LFDSF`, page 85. To solve systems of equations with multiple right-hand-side vectors, use `LFCSF` followed by either `LFISF` or `LFSSF` called once for each right-hand side. The routine `LFDSF` can be called to compute the determinant of the coefficient matrix after `LFCSF` has performed the factorization.

LFCSF is based on the LINPACK routine SSICO; see Dongarra et al. (1979).

Example

The inverse of a 3×3 matrix is computed. LFCSF is called to factor the matrix and to check for singularity or ill-conditioning. LFISF (page 83) is called to determine the columns of the inverse.

```
C                                 Declare variables
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       A(LDA,LDA), AINV(N,N), FAC(LDA,LDA), RJ(N), RES(N),
     &           COND
C
C                                 Set values for A
C
C                                 A = (  1.0  -2.0   1.0)
C                                     ( -2.0   3.0  -2.0)
C                                     (  1.0  -2.0   3.0)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
C                                 Factor A and return the reciprocal
C                                 condition number estimate
      CALL LFCSF (N, A, LDA, FAC, LDA, IPVT, COND)
C                                 Print the estimate of the condition
C                                 number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) COND, 1.0E0/COND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFISF
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFISF (N, A, LDA, FAC, LDA, IPVT, RJ, AINV(1,J), RES)
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print the inverse
      CALL WRRRN (’AINV’, N, N, AINV, LDA, 0)
99999 FORMAT (’  COND = ’,F5.3,/,’  L1 Condition number = ’,F6.3)
      END
```

Output

```
COND = 0.034
L1 Condition number = 29.750

          AINV
        1       2       3
1  -2.500  -2.000  -0.500
2  -2.000  -1.000   0.000
3  -0.500   0.000   0.500
```

LFTSF/DLFTSF (Single/Double precision)

Compute the $U\,DU^T$ factorization of a real symmetric matrix.

Usage

```
CALL LFTSF (N, A, LDA, FAC, LDFAC, IPVT)
```

Arguments

N — Order of the matrix. (Input)

A — `N` by `N` symmetric matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — `N` by `N` matrix containing the information about the factorization of the symmetric matrix `A`. (Output)
Only the upper triangle of `FAC` is used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization. (Output)

Comments

Informational error

Type	Code	
4	2	The input matrix is singular.

Algorithm

Routine `LFTSF` performs a $U\,DU^T$ factorization of a real symmetric indefinite coefficient matrix. The $U\,DU^T$ factorization is called the diagonal pivoting factorization.

`LFTSF` fails if *A* is singular or very close to a singular matrix.

The $U\,DU^T$ factors are returned in a form that is compatible with routines `LFISF`, page 83, `LFSSF`, page 81, and `LFDSF`, page 85. To solve systems of equations with multiple right-hand-side vectors, use `LFTSF` followed by either `LFISF` or `LFSSF` called once for each right-hand side. The routine `LFDSF` can be called to compute the determinant of the coefficient matrix after `LFTSF` has performed the factorization.

`LFTSF` is based on the LINPACK routine `SSIFA`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 matrix is computed. `LFTSF` is called to factor the matrix and to check for singularity. `LFSSF` (page 81) is called to determine the columns of the inverse.

```
C                                 Declare variables
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N)
      REAL       A(LDA,LDA), AINV(N,N), FAC(LDA,LDA), RJ(N)
C
C                                 Set values for A
C                                 A = (  1.0  -2.0   1.0)
C                                     ( -2.0   3.0  -2.0)
C                                     (  1.0  -2.0   3.0)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
C                                 Factor A
      CALL LFTSF (N, A, LDA, FAC, LDA, IPVT)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSSF
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSSF (N, FAC, LDA, IPVT, RJ, AINV(1,J))
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print the inverse
      CALL WRRRN ('AINV', N, N, AINV, LDA, 0)
      END
```

Output

```
           AINV
        1       2       3
1  -2.500  -2.000  -0.500
2  -2.000  -1.000   0.000
3  -0.500   0.000   0.500
```

LFSSF/DLFSSF (Single/Double precision)

Solve a real symmetric system of linear equations given the $U DU^T$ factorization of the coefficient matrix.

Usage

`CALL LFSSF (N, FAC, LDFAC, IPVT, B, X)`

Arguments

N — Number of equations. (Input)

FAC — `N` by `N` matrix containing the factorization of the coefficient matrix `A` as output from routine `LFCSF`/`DLFCSF` or `LFTSF`/`DLFTSF`. (Input)
Only the upper triangle of `FAC` is used.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization of `A` as output from routine `LFCSF`/`DLFCSF` or `LFTSF`/`DLFTSF`. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Algorithm

Routine `LFSSF` computes the solution of a system of linear algebraic equations having a real symmetric indefinite coefficient matrix.

To compute the solution, the coefficient matrix must first undergo a $U\,DU^T$ factorization. This may be done by calling either `LFCSF`, page 77, or `LFTSF`, page 80.

`LFSSF` and `LFISF`, page 83, both solve a linear system given its $U\,DU^T$ factorization. `LFISF` generally takes more time and produces a more accurate answer than `LFSSF`. Each iteration of the iterative refinement algorithm used by `LFISF` calls `LFSSF`.

`LFSSF` is based on the LINPACK routine `SSISL`; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. `LFTSF` (page 80) is called to factor the coefficient matrix. `LFSSF` is called to compute the four solutions for the four right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call `LFCSF` (page 77) to perform the factorization, and `LFISF` (page 83) to compute the solutions.

```
C                                  Declare variables
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N)
      REAL       A(LDA,LDA), B(N,4), X(N,4), FAC(LDA,LDA)
C
C                                  Set values for A and B
C
C                                  A = (  1.0  -2.0   1.0)
C                                      ( -2.0   3.0  -2.0)
C                                      (  1.0  -2.0   3.0)
```

```
C
C                                 B = ( -1.0   3.6  -8.0  -9.4)
C                                     ( -3.0  -4.2  11.0  17.6)
C                                     ( -3.0  -5.2  -6.0 -23.4)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
      DATA B/-1.0, -3.0, -3.0, 3.6, -4.2, -5.2, -8.0, 11.0, -6.0,
     &     -9.4, 17.6, -23.4/
C                                 Factor A
      CALL LFTSF (N, A, LDA, FAC, LDA, IPVT)
C                                 Solve for the four right-hand sides
      DO 10  I=1, 4
         CALL LFSSF (N, FAC, LDA, IPVT, B(1,I), X(1,I))
   10 CONTINUE
C                                 Print results
      CALL WRRRN (’X’, N, 4, X, N, 0)
      END
```

Output

```
                 X
        1        2        3        4
1   10.00     2.00     1.00     0.00
2    5.00    -3.00     5.00     1.20
3   -1.00    -4.40     1.00    -7.00
```

LFISF/DLFISF (Single/Double precision)

Use iterative refinement to improve the solution of a real symmetric system of linear equations.

Usage

`CALL LFISF (N, A, LDA, FAC, LDFAC, IPVT, B, X, RES)`

Arguments

N — Number of equations. (Input)

A — `N` by `N` matrix containing the coefficient matrix of the symmetric linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — `N` by `N` matrix containing the factorization of the coefficient matrix `A` as output from routine `LFCSF`/`DLFCSF` or `LFTSF`/`DLFTSF`. (Input)
Only the upper triangle of `FAC` is used.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization of `A` as output from routine `LFCSF`/`DLFCSF` or `LFTSF`/`DLFTSF`. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution. (Input)

RES — Vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	2	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

`LFISF` computes the solution of a system of linear algebraic equations having a real symmetric indefinite coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo a $U\,DU^T$ factorization. This may be done by calling either `LFCSF`, page 77, or `LFTSF`, page 80.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFISF` and `LFSSF`, page 81, both solve a linear system given its $U\,DU^T$ factorization. `LFISF` generally takes more time and produces a more accurate answer than `LFSSF`. Each iteration of the iterative refinement algorithm used by `LFISF` calls `LFSSF`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.2 to the second element.

```
C                                 Declare variables
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       A(LDA,LDA), B(N), X(N), FAC(LDA,LDA), RES(N), COND
C
C                                 Set values for A and B
C                                 A = (  1.0  -2.0   1.0)
C                                     ( -2.0   3.0  -2.0)
C                                     (  1.0  -2.0   3.0)
C
C                                 B = (  4.1  -4.7   6.5)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
      DATA B/4.1, -4.7, 6.5/
C                                 Factor A and compute the estimate
```

```
C                                  of the reciprocal condition number
      CALL LFCSF (N, A, LDA, FAC, LDA, IPVT, COND)
C                                  Print condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) COND, 1.0E0/COND
C                                  Solve, then perturb right-hand side
      DO 10  I=1, 3
         CALL LFISF (N, A, LDA, FAC, LDA, IPVT, B, X, RES)
C                                  Print results
         CALL WRRRN ('X', 1, N, X, 1, 0)
         CALL WRRRN ('RES', 1, N, RES, 1, 0)
         B(2) = B(2) + .20E0
   10 CONTINUE
C
99999 FORMAT ('  COND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
COND = 0.034
L1 Condition number = 29.750

           X
     1       2       3
-4.100  -3.500   1.200

               RES
         1           2           3
-2.384E-07  -2.384E-07   0.000E+00

            X
     1       2       3
-4.500  -3.700   1.200

               RES
         1           2           3
-2.384E-07  -2.384E-07   0.000E+00

            X
     1       2       3
-4.900  -3.900   1.200

               RES
         1           2           3
-2.384E-07  -2.384E-07   0.000E+00
```

LFDSF/DLFDSF (Single/Double precision)

Compute the determinant of a real symmetric matrix given the $U\,DU^T$ factorization of the matrix.

Usage

```
CALL LFDSF (N, FAC, LDFAC, IPVT, DET1, DET2)
```

Arguments

N — Order of the matrix. (Input)

FAC — N by N matrix containing the factored matrix *A* as output from subroutine LFTSF/DLFTSF or LFCSF/DLFCSF. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the $U DU^T$ factorization as output from routine LFTSF/DLFTSF or LFCSF/DLFCSF. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that, 1.0 ≤ |DET1| < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form, det(*A*) = DET1 * 10^{DET2}.

Algorithm

Routine LFDSF computes the determinant of a real symmetric indefinite coefficient matrix. To compute the determinant, the coefficient matrix must first undergo a $U DU^T$ factorization. This may be done by calling either LFCSF, page 77, or LFTSF, page 80. Since det *U* = ±1, the formula det *A* = det *U* det *D* det U^T = det *D* is used to compute the determinant. Next det *D* is computed as the product of the determinants of its blocks.

LFDSF is based on the LINPACK routine SSIDI; see Dongarra et al. (1979).

Example

The determinant is computed for a real symmetric 3 × 3 matrix.

```
C                                 Declare variables
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       A(LDA,LDA), FAC(LDA,LDA), DET1, DET2
C
C                                 Set values for A
C                                 A = (  1.0  -2.0   1.0)
C                                     ( -2.0   3.0  -2.0)
C                                     (  1.0  -2.0   3.0)
C
      DATA A/1.0, -2.0, 1.0, -2.0, 3.0, -2.0, 1.0, -2.0, 3.0/
C                                 Factor A
      CALL LFTSF (N, A, LDA, FAC, LDA, IPVT)
C                                 Compute the determinant
      CALL LFDSF (N, FAC, LDA, IPVT, DET1, DET2)
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
99999 FORMAT (’ The determinant of A is ’, F6.3, ’ * 10**’, F2.0)
      END
```

Output

```
The determinant of A is -2.000 * 10**0.
```

LSADH/DLSADH (Single/Double precision)

Solve a Hermitian positive definite system of linear equations with iterative refinement.

Usage

```
CALL LSADH (N, A, LDA, B, X)
```

Arguments

N — Number of equations. (Input)

A — Complex N by N matrix containing the coefficient matrix of the Hermitian positive definite linear system. (Input)
Only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

X — Complex vector of length N containing the solution of the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSADH $2N^2 + 2N$ units, or
   DLSADH $4N^2 + 4N$ units.

   Workspace may be explicitly provided, if desired, by use of L2ADH/DL2ADH. The reference is

   ```
   CALL L2ADH (N, A, LDA, B, X, FAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length N^2 containing the $R^H R$ factorization of A on output.

   WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.

3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2ADH the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSADH. Additional memory allocation for FAC and option value restoration are done automatically in LSADH. Users directly calling L2ADH can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSADH or L2ADH. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSADH temporarily replaces IVAL(2) by IVAL(1). The routine L2CDH computes the condition number if IVAL(2) = 2. Otherwise L2CDH skips this computation. LSADH restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSADH solves a system of linear algebraic equations having a complex Hermitian positive definite coefficient matrix. It first uses the routine LFCDH, page 92, to compute an $R^H R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. The matrix R is upper triangular. The solution of the linear system is then found using the iterative refinement routine LFIDH, page 99.

LSADH fails if any submatrix of R is not positive definite, if R has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if A either is very close to a singular matrix or is a matrix that is not positive definite.

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system. LSADH solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of five linear equations is solved. The coefficient matrix has complex positive definite form and the right-hand-side vector *b* has five elements.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=5, N=5)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C
C                                 Set values for A and B
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (             4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                       10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                  6.0+0.0i   1.0+1.0i )
C              (                                             9.0+0.0i )
C
C        B =   ( 1.0+5.0i  12.0-6.0i  1.0-16.0i  -3.0-3.0i  25.0+16.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B /(1.0,5.0), (12.0,-6.0), (1.0,-16.0), (-3.0,-3.0),
     &        (25.0,16.0)/
C
      CALL LSADH (N, A, LDA, B, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
                                  X
              1                2                3                4
( 2.000, 1.000)  ( 3.000, 0.000)  (-1.000,-1.000)  ( 0.000,-2.000)
              5
( 3.000, 2.000)
```

LSLDH/DLSLDH (Single/Double precision)

Solve a complex Hermitian positive definite system of linear equations without iterative refinement.

Usage

`CALL LSLDH (N, A, LDA, B, X)`

Arguments

N — Number of equations.   (Input)

A — Complex `N` by `N` matrix containing the coefficient matrix of the Hermitian positive definite linear system.   (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Comments

1. Automatic workspace usage is

   `LSLDH` $2N^2 + 2N$ units, or
   `DLSLDH` $4N^2 + 4N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2LDH`/`DL2LDH`. The reference is

   `CALL L2LDH (N, A, LDA, B, X, FAC, WK)`

   The additional arguments are as follows:

   FAC – Complex work vector of length N^2 containing the $R^H R$ factorization of `A` on output. If `A` is not needed, `A` can share the same storage locations as `FAC`.

   WK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine `L2LDH` the leading dimension of `FAC` is increased by `IVAL`(3) when `N` is a multiple of `IVAL`(4). The values `IVAL`(3) and `IVAL`(4) are temporarily replaced by `IVAL`(1) and `IVAL`(2), respectively, in `LSLDH`. Additional memory allocation for `FAC` and option value restoration are done automatically in `LSLDH`. Users directly calling `L2LDH` can allocate additional space for `FAC` and set `IVAL`(3) and `IVAL`(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLDH or L2LDH. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLDH temporarily replaces IVAL(2) by IVAL(1). The routine L2CDH computes the condition number if IVAL(2) = 2. Otherwise L2CDH skips this computation. LSLDH restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLDH solves a system of linear algebraic equations having a complex Hermitian positive definite coefficient matrix. It first uses the routine LFCDH, page 92, to compute an $R^H R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. The matrix R is upper triangular. The solution of the linear system is then found using the routine LFSDH, page 97.

LSLDH fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that LSADH, page 87, be used.

Example

A system of five linear equations is solved. The coefficient matrix has complex Hermitian positive definite form and the right-hand-side vector b has five elements.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=5, N=5)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C
C                                  Set values for A and B
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (             4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                       10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                   6.0+0.0i   1.0+1.0i )
C              (                                              9.0+0.0i )
C
C        B =   ( 1.0+5.0i  12.0-6.0i  1.0-16.0i  -3.0-3.0i  25.0+16.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B /(1.0,5.0), (12.0,-6.0), (1.0,-16.0), (-3.0,-3.0),
     &        (25.0,16.0)/
C
      CALL LSLDH (N, A, LDA, B, X)
C                                Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
C
      END
```

Output

```
                                      X
              1                 2                 3                 4
( 2.000, 1.000)  ( 3.000, 0.000)  (-1.000,-1.000)  ( 0.000,-2.000)
              5
( 3.000, 2.000)
```

LFCDH/DLFCDH (Single/Double precision)

Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix and estimate its L_1 condition number.

Usage

CALL LFCDH (N, A, LDA, FAC, LDFAC, RCOND)

Arguments

N — Order of the matrix. (Input)

A — Complex N by N Hermitian positive definite matrix to be factored. (Input)
Only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex N by N matrix containing the upper triangular matrix *R* of the factorization of A in the upper triangle. (Output)
Only the upper triangle of FAC will be used. If A is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of A. (Output)

Comments

1. Automatic workspace usage is

   LFCDH 2N units, or
   DLFCDH 4N units.

Workspace may be explicitly provided, if desired, by use of L2CDH/DL2CDH. The reference is

```
CALL L2CDH (N, A, LDA, FAC, LDFAC, RCOND, WK)
```

The additional argument is

WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	4	The input matrix is not Hermitian.
4	2	The input matrix is not positive definite. It has a diagonal entry with an imaginary part.

Algorithm

Routine LFCDH computes an $R^H R$ Cholesky factorization and estimates the condition number of a complex Hermitian positive definite coefficient matrix. The matrix R is upper triangular.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

LFCDH fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

The $R^H R$ factors are returned in a form that is compatible with routines LFIDH, page 99, LFSDH, page 97, and LFDDH, page 101. To solve systems of equations with multiple right-hand-side vectors, use LFCDH followed by either LFIDH or LFSDH called once for each right-hand side. The routine LFDDH can be called to compute the determinant of the coefficient matrix after LFCDH has performed the factorization.

LFCDH is based on the LINPACK routine CPOCO; see Dongarra et al. (1979).

Example

The inverse of a 5 × 5 Hermitian positive definite matrix is computed. `LFCDH` is called to factor the matrix and to check for nonpositive definiteness or ill-conditioning. `LFIDH` (page 99) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NOUT
      PARAMETER  (LDA=5, LDFAC=5, N=5)
      REAL       COND
      COMPLEX    A(LDA,LDA), AINV(LDA,LDA), FAC(LDFAC,LDFAC),
     &           RES(N), RJ(N)
C
C                                 Set values for A
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (             4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                       10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                  6.0+0.0i   1.0+1.0i )
C              (                                             9.0+0.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
C                                 Factor the matrix A
      CALL LFCDH (N, A, LDA, FAC, LDFAC, COND)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIDH
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIDH (N, A, LDA, FAC, LDFAC, RJ, AINV(1,J), RES)
         RJ(J) = (0.0E0,0.0E0)
   10 CONTINUE
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) COND, 1.0E0/COND
      CALL WRCRN ('AINV', N, N, AINV, LDA, 0)
C
99999 FORMAT ('  COND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
COND = 0.067
L1 Condition number = 14.961

                               AINV
                 1                   2                   3                   4
1 ( 0.7166, 0.0000) ( 0.2166,-0.2166)  (-0.0899,-0.0300)  (-0.0207, 0.0622)
2 ( 0.2166, 0.2166) ( 0.4332, 0.0000)  (-0.0599,-0.1198)  (-0.0829, 0.0415)
3 (-0.0899, 0.0300) (-0.0599, 0.1198)  ( 0.1797, 0.0000)  ( 0.0000,-0.1244)
4 (-0.0207,-0.0622) (-0.0829,-0.0415)  ( 0.0000, 0.1244)  ( 0.2592, 0.0000)
5 ( 0.0092, 0.0046) ( 0.0138,-0.0046)  (-0.0138,-0.0138)  (-0.0288, 0.0288)
```

```
                  5
1  ( 0.0092,-0.0046)
2  ( 0.0138, 0.0046)
3  (-0.0138, 0.0138)
4  (-0.0288,-0.0288)
5  ( 0.1175, 0.0000)
```

LFTDH/DLFTDH (Single/Double precision)

Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix.

Usage

```
CALL LFTDH (N, A, LDA, FAC, LDFAC)
```

Arguments

N – Order of the matrix. (Input)

A — Complex `N` by `N` Hermitian positive definite matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex `N` by `N` matrix containing the upper triangular matrix `R` of the factorization of `A` in the upper triangle. (Output)
Only the upper triangle of `FAC` will be used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

Comments

Informational errors

Type	Code	
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `LFTDH` computes an $R^H R$ Cholesky factorization of a complex Hermitian positive definite coefficient matrix. The matrix R is upper triangular.

LFTDH fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

The $R^H R$ factors are returned in a form that is compatible with routines LFIDH, page 99, LFSDH, page 97, and LFDDH, page 101. To solve systems of equations with multiple right-hand-side vectors, use LFCDH followed by either LFIDH or LFSDH called once for each right-hand side. The IMSL routine LFDDH can be called to compute the determinant of the coefficient matrix after LFCDH has performed the factorization.

LFTDH is based on the LINPACK routine CPOFA; see Dongarra et al. (1979).

Example

The inverse of a 5 × 5 matrix is computed. LFTDH is called to factor the matrix and to check for nonpositive definiteness. LFSDH (page 97) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=5, LDFAC=5, N=5)
      COMPLEX    A(LDA,LDA), AINV(LDA,LDA), FAC(LDFAC,LDFAC), RJ(N)
C
C                                 Set values for A
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (             4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                       10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                  6.0+0.0i   1.0+1.0i )
C              (                                             9.0+0.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
C                                 Factor the matrix A
      CALL LFTDH (N, A, LDA, FAC, LDFAC)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSDH
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSDH (N, FAC, LDFAC, RJ, AINV(1,J))
         RJ(J) = (0.0E0,0.0E0)
   10 CONTINUE
C                                 Print the results
      CALL WRCRN (’AINV’, N, N, AINV, LDA, 1)
C
      END
```

Output

```
                                   AINV
                  1                  2                  3                  4
1 ( 0.7166, 0.0000) ( 0.2166,-0.2166)  (-0.0899,-0.0300)  (-0.0207, 0.0622)
2                   ( 0.4332, 0.0000)  (-0.0599,-0.1198)  (-0.0829, 0.0415)
3                                      ( 0.1797, 0.0000)  ( 0.0000,-0.1244)
4                                                         ( 0.2592, 0.0000)
                   5
1  ( 0.0092,-0.0046)
2  ( 0.0138, 0.0046)
3  (-0.0138, 0.0138)
4  (-0.0288,-0.0288)
5  ( 0.1175, 0.0000)
```

LFSDH/DLFSDH (Single/Double precision)

Solve a complex Hermitian positive definite system of linear equations given the $R^H R$ factorization of the coefficient matrix.

Usage

```
CALL LFSDH (N, FAC, LDFAC, B, X)
```

Arguments

N – Number of equations. (Input)

FAC — Complex `N` by `N` matrix containing the factorization of the coefficient matrix `A` as output from routine `LFCDH`/`DLFCDH` or `LFTDH`/`DLFTDH`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Comments

Informational error

Type	Code	
4	1	The input matrix is singular.

Algorithm

This routine computes the solution for a system of linear algebraic equations having a complex Hermitian positive definite coefficient matrix. To compute the solution, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either LFCDH, page 92, or LFTDH, page 95. *R* is an upper triangular matrix.

The solution to $Ax = b$ is found by solving the triangular systems $R^H y = b$ and $Rx = y$.

LFSDH and LFIDH, page 99, both solve a linear system given its R^H R factorization. LFIDH generally takes more time and produces a more accurate answer than LFSDH. Each iteration of the iterative refinement algorithm used by LFIDH calls LFSDH.

LFSDH is based on the LINPACK routine CPOSL; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. LFTDH (page 95) is called to factor the coefficient matrix. LFSDH is called to compute the four solutions for the four right-hand sides. In this case, the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call LFCDH (page 92) to perform the factorization, and LFIDH (page 99) to compute the solutions.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=5, LDFAC=5, N=5)
      COMPLEX    A(LDA,LDA), B(N,3), FAC(LDFAC,LDFAC), X(N,3)
C
C                                 Set values for A and B
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (             4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                       10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                  6.0+0.0i   1.0+1.0i )
C              (                                             9.0+0.0i )
C
C        B =   (  3.0+3.0i    4.0+0.0i    29.0-9.0i )
C              (  5.0-5.0i   15.0-10.0i  -36.0-17.0i )
C              (  5.0+4.0i  -12.0-56.0i  -15.0-24.0i )
C              (  9.0+7.0i  -12.0+10.0i  -23.0-15.0i )
C              (-22.0+1.0i    3.0-1.0i   -23.0-28.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B /(3.0,3.0), (5.0,-5.0), (5.0,4.0), (9.0,7.0), (-22.0,1.0),
     &        (4.0,0.0), (15.0,-10.0), (-12.0,-56.0), (-12.0,10.0),
     &        (3.0,-1.0), (29.0,-9.0), (-36.0,-17.0), (-15.0,-24.0),
     &        (-23.0,-15.0), (-23.0,-28.0)/
C
C                                 Factor the matrix A
      CALL LFTDH (N, A, LDA, FAC, LDFAC)
C                                 Compute the solutions
      DO 10  I=1, 3
         CALL LFSDH (N, FAC, LDFAC, B(1,I), X(1,I))
   10 CONTINUE
```

```
C                                  Print solutions
      CALL WRCRN (’X’, N, 3, X, N, 0)
C
      END
```

Output

```
                          X
                 1                 2                 3
1  (  1.00,  0.00)  (  3.00, -1.00)  ( 11.00, -1.00)
2  (  1.00, -2.00)  (  2.00,  0.00)  ( -7.00,  0.00)
3  (  2.00,  0.00)  ( -1.00, -6.00)  ( -2.00, -3.00)
4  (  2.00,  3.00)  (  2.00,  1.00)  ( -2.00, -3.00)
5  ( -3.00,  0.00)  (  0.00,  0.00)  ( -2.00, -3.00)
```

LFIDH/DLFIDH (Single/Double precision)

Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations.

Usage

```
CALL LFIDH (N, A, LDA, FAC, LDFAC, B, X, RES)
```

Arguments

N — Number of equations. (Input)

A — Complex N by N matrix containing the coefficient matrix of the linear system. (Input)
Only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex N by N matrix containing the factorization of the coefficient matrix A as output from routine LFCDH/DLFCDH or LFTDH/DLFTDH. (Input)
Only the upper triangle of FAC is used.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

X — Complex vector of length N containing the solution. (Input)

RES — Complex vector of length N containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	3	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIDH` computes the solution of a system of linear algebraic equations having a complex Hermitian positive definite coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either `LFCDH`, page 92, or `LFTDH`, page 95.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIDH` and `LFSDH`, page 97, both solve a linear system given its $R^H R$ factorization. `LFIDH` generally takes more time and produces a more accurate answer than `LFSDH`. Each iteration of the iterative refinement algorithm used by `LFIDH` calls `LFSDH`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed by adding (1 + *i*)/2 to the second element after each call to `LFIDH`.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=5, LDFAC=5, N=5)
      REAL       RCOND
      COMPLEX    A(LDA,LDA), B(N), FAC(LDFAC,LDFAC), RES(N,3), X(N,3)
C
C                                 Set values for A and B
C
C        A =   (  2.0+0.0i  -1.0+1.0i   0.0+0.0i   0.0+0.0i   0.0+0.0i )
C              (              4.0+0.0i   1.0+2.0i   0.0+0.0i   0.0+0.0i )
C              (                        10.0+0.0i   0.0+4.0i   0.0+0.0i )
C              (                                   6.0+0.0i   1.0+1.0i )
C              (                                              9.0+0.0i )
C
C        B =   ( 3.0+3.0i  5.0-5.0i  5.0+4.0i  9.0+7.0i  -22.0+1.0i )
C
      DATA A /(2.0,0.0), 4*(0.0,0.0), (-1.0,1.0), (4.0,0.0),
     &        4*(0.0,0.0), (1.0,2.0), (10.0,0.0), 4*(0.0,0.0),
     &        (0.0,4.0), (6.0,0.0), 4*(0.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B /(3.0,3.0), (5.0,-5.0), (5.0,4.0), (9.0,7.0), (-22.0,1.0)/
C                                 Factor the matrix A
      CALL LFCDH (N, A, LDA, FAC, LDFAC, RCOND)
C                                 Print the estimated condition number
```

```
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                  Compute the solutions, then perturb B
      DO 10  I=1, 3
         CALL LFIDH (N, A, LDA, FAC, LDFAC, B, X(1,I), RES(1,I))
         B(2) = B(2) + (0.5E0,0.5E0)
   10 CONTINUE
C                                  Print solutions and residuals
      CALL WRCRN ('X', N, 3, X, N, 0)
      CALL WRCRN ('RES', N, 3, RES, N, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.067
L1 Condition number = 14.961

                       X
                 1                 2                 3
1  ( 1.000, 0.000)  ( 1.217, 0.000)  ( 1.433, 0.000)
2  ( 1.000,-2.000)  ( 1.217,-1.783)  ( 1.433,-1.567)
3  ( 2.000, 0.000)  ( 1.910, 0.030)  ( 1.820, 0.060)
4  ( 2.000, 3.000)  ( 1.979, 2.938)  ( 1.959, 2.876)
5  (-3.000, 0.000)  (-2.991, 0.005)  (-2.982, 0.009)

                                 RES
                        1                         2                         3
1 ( 1.192E-07, 0.000E+00)  ( 6.592E-08, 1.686E-07)  ( 1.318E-07, 2.010E-14)
2 ( 1.192E-07,-2.384E-07)  (-5.329E-08,-5.329E-08)  ( 1.318E-07,-2.258E-07)
3 ( 2.384E-07, 8.259E-08)  ( 2.390E-07,-3.309E-08)  ( 2.395E-07, 1.015E-07)
4 (-2.384E-07, 2.814E-14)  (-8.240E-08,-8.790E-09)  (-1.648E-07,-1.758E-08)
5 (-2.384E-07,-1.401E-08)  (-2.813E-07, 6.981E-09)  (-3.241E-07,-2.795E-08)
```

LFDDH/DLFDDH (Single/Double precision)

Compute the determinant of a complex Hermitian positive definite matrix given the $R^T R$ Cholesky factorization of the matrix.

Usage

`CALL LFDDH (N, FAC, LDFAC, DET1, DET2)`

Arguments

N – Order of the matrix. (Input)

FAC — Complex `N` by `N` matrix containing the $R^T R$ factorization of the coefficient matrix `A` as output from routine `LFCDH`/`DLFCDH` or `LFTDH`/`DLFTDH`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that 1.0 ≤ |DET1| < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det(A) = DET1 * 10^{DET2}.

Algorithm

Routine LFDDH computes the determinant of a complex Hermitian positive definite coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either LFCDH, page 92, or LFTDH, page 95. The formula det A = det R^H det R = (det R)^2 is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det R = \prod\nolimits_{i=1}^{N} R_{ii}$$

(The matrix R is stored in the upper triangle of FAC.)

LFDDH is based on the LINPACK routine CPODI; see Dongarra et al. (1979).

Example

The determinant is computed for a complex Hermitian positive definite 3 × 3 matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NOUT
      PARAMETER  (LDA=3, N=3, LDFAC=3)
      REAL       DET1, DET2
      COMPLEX    A(LDA,LDA), FAC(LDFAC,LDFAC)
C
C                                 Set values for A
C
C         A =   (  6.0+0.0i   1.0-1.0i   4.0+0.0i )
C               (  1.0+1.0i   7.0+0.0i  -5.0+1.0i )
C               (  4.0+0.0i  -5.0-1.0i  11.0+0.0i )
C
      DATA A /(6.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (7.0,0.0),
     &        (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (11.0,0.0)/
C                                 Factor the matrix
      CALL LFTDH (N, A, LDA, FAC, LDFAC)
C                                 Compute the determinant
      CALL LFDDH (N, FAC, LDFAC, DET1, DET2)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant of A is ’,F6.3,’ * 10**’,F2.0)
      END
```

Output

```
The determinant of A is  1.400 * 10**2.
```

LSAHF/DLSAHF (Single/Double precision)

Solve a complex Hermitian system of linear equations with iterative refinement.

Usage

```
CALL LSAHF (N, A, LDA, B, X)
```

Arguments

N – Number of equations. (Input)

A — Complex `N` by `N` matrix containing the coefficient matrix of the Hermitian linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSAHF` $2N^2 + 3N$ units, or
   `DLSAHF` $4N^2 + 5N$ units.

   Workspace may be explicitly provided, if desired, by use of `L2AHF/DL2AHF`. The reference is

   ```
   CALL L2AHF (N, A, LDA, B, X, FAC, IPVT, CWK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length N^2 containing information about the $U\,DU^H$ factorization of `A` on output.

   IPVT — Integer work vector of length `N` containing the pivoting information for the factorization of `A` on output.

   CWK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix singular.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2AHF the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSAHF. Additional memory allocation for FAC and option value restoration are done automatically in LSAHF. Users directly calling L2AHF can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSAHF or L2AHF. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSAHF temporarily replaces IVAL(2) by IVAL(1). The routine L2CHF computes the condition number if IVAL(2) = 2. Otherwise L2CHF skips this computation. LSAHF restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSAHF solves systems of linear algebraic equations having a complex Hermitian indefinite coefficient matrix. It first uses the routine LFCHF, page 108, to compute a $U\,DU^H$ factorization of the coefficient matrix and to estimate the condition number of the matrix. D is a block diagonal matrix with blocks of order 1 or 2 and U is a matrix composed of the product of a permutation matrix and a unit upper triangular matrix. The solution of the linear system is then found using the iterative refinement routine LFIHF, page 114.

LSAHF fails if a block in D is singular or if the iterative refinement algorithm fails to converge. These errors occur only if A is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system. LSAHF solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of three linear equations is solved. The coefficient matrix has complex Hermitian form and the right-hand-side vector *b* has three elements.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C
C                               Set values for A and B
C
C                               A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                   ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                   ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
C                               B = ( 7.0+32.0i -39.0-21.0i 51.0+9.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
      DATA B/(7.0,32.0), (-39.0,-21.0), (51.0,9.0)/
C
      CALL LSAHF (N, A, LDA, B, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
                        X
             1                 2                 3
(  2.00,  1.00)  (-10.00, -1.00)  (  3.00,  5.00)
```

LSLHF/DLSLHF (Single/Double precision)

Solve a complex Hermitian system of linear equations without iterative refinement.

Usage

```
CALL LSLHF (N, A, LDA, B, X)
```

Arguments

N – Number of equations. (Input)

A — Complex `N` by `N` matrix containing the coefficient matrix of the Hermitian linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSLHF` 2`N`^{2} + 3`N` units, or

   `DLSLHF` 4`N`^{2} + 5`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2LHF`/`DL2LHF`. The reference is

   `CALL L2LHF (N, A, LDA, B, X, FAC, IPVT, CWK)`

   The additional arguments are as follows:

   FAC — Complex work vector of length `N`^{2} containing information about the $U\,DU^H$ factorization of `A` on output. If `A` is not needed, `A` can share the same storage locations with `FAC`.

   IPVT — Integer work vector of length `N` containing the pivoting information for the factorization of `A` on output.

   CWK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is singular.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine `L2LHF` the leading dimension of `FAC` is increased by `IVAL`(3) when `N` is a multiple of `IVAL`(4). The values `IVAL`(3) and `IVAL`(4) are temporarily replaced by `IVAL`(1) and `IVAL`(2), respectively, in `LSLHF`. Additional memory allocation for `FAC` and option value restoration are done automatically in `LSLHF`. Users directly calling `L2LHF` can allocate additional space for `FAC` and set `IVAL`(3) and `IVAL`(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use `LSLHF` or `L2LHF`. Default values for the option are `IVAL(*)` = 1, 16, 0, 1.

   17 This option has two values that determine if the L_1 condition number is to be computed. Routine `LSLHF` temporarily replaces IVAL(2) by IVAL(1). The routine L2CHF computes the condition number if IVAL(2) = 2. Otherwise L2CHF skips this computation. LSLHF restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLHF solves systems of linear algebraic equations having a complex Hermitian indefinite coefficient matrix. It first uses the routine LFCHF, page 108, to compute a $U\,DU^H$ factorization of the coefficient matrix. D is a block diagonal matrix with blocks of order 1 or 2 and U is a matrix composed of the product of a permutation matrix and a unit upper triangular matrix.

The solution of the linear system is then found using the routine LFSHF, page 112. LSLHF fails if a block in D is singular. This occurs only if A is singular or very close to a singular matrix. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that LSAHF, page 103 be used.

Example

A system of three linear equations is solved. The coefficient matrix has complex Hermitian form and the right-hand-side vector b has three elements.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      COMPLEX    A(LDA,LDA), B(N), X(N)
C
C                                Set values for A and B
C
C                                A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                    ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                    ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
C                                B = ( 7.0+32.0i -39.0-21.0i 51.0+9.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
      DATA B/(7.0,32.0), (-39.0,-21.0), (51.0,9.0)/
C
      CALL LSLHF (N, A, LDA, B, X)
C                                  Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                        X
              1                 2                 3
(  2.00,  1.00)  (-10.00, -1.00)  (  3.00,  5.00)
```

LFCHF/DLFCHF (Single/Double precision)

Compute the $U\,DU^H$ factorization of a complex Hermitian matrix and estimate its L_1 condition number.

Usage

```
CALL LFCHF (N, A, LDA, FAC, LDFAC, IPVT, RCOND)
```

Arguments

N – Order of the matrix. (Input)

A — Complex `N` by `N` Hermitian matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex `N` by `N` matrix containing the information about the factorization of the Hermitian matrix `A`. (Output)
Only the upper triangle of `FAC` is used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCHF` 2`N` units, or
   `DLFCHF` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CHF`/`DL2CHF`. The reference is

   ```
   CALL L2CHF (N, A, LDA, FAC, LDFAC, IPVT, RCOND, CWK)
   ```

   The additional argument is

   CWK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.

4	2	The input matrix is singular.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `LFCHF` performs a $U\,DU^H$ factorization of a complex Hermitian indefinite coefficient matrix. It also estimates the condition number of the matrix. The $U\,DU^H$ factorization is called the diagonal pivoting factorization.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

`LFCHF` fails if A is singular or very close to a singular matrix.

The $U\,DU^H$ factors are returned in a form that is compatible with routines `LFIHF`, page 114, `LFSHF`, page 112, and `LFDHF`, page 117. To solve systems of equations with multiple right-hand-side vectors, use `LFCHF` followed by either `LFIHF` or `LFSHF` called once for each right-hand side. The routine `LFDHF` can be called to compute the determinant of the coefficient matrix after `LFCHF` has performed the factorization.

`LFCHF` is based on the LINPACK routine `CSICO`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 complex Hermitian matrix is computed. `LFCHF` is called to factor the matrix and to check for singularity or ill-conditioning. `LFIHF` (page 114) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       RCOND
      COMPLEX    A(LDA,LDA), AINV(LDA,N), FAC(LDA,LDA), RJ(N), RES(N)
C                               Set values for A
C
C                               A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                   ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                   ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
C                                 Set output unit number
```

```
      CALL UMACH (2, NOUT)
C                                 Factor A and return the reciprocal
C                                 condition number estimate
      CALL LFCHF (N, A, LDA, FAC, LDA, IPVT, RCOND)
C                                 Print the estimate of the condition
C                                 number
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0, 0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIHF
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIHF (N, A, LDA, FAC, LDA, IPVT, RJ, AINV(1,J), RES)
         RJ(J) = (0.0E0, 0.0E0)
   10 CONTINUE
C                                 Print the inverse
      CALL WRCRN ('AINV', N, N, AINV, LDA, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.240
L1 Condition number =  4.175

                           AINV
                  1                   2                   3
1  ( 0.2000, 0.0000)  ( 0.1200, 0.0400)  ( 0.0800,-0.0400)
2  ( 0.1200,-0.0400)  ( 0.1467, 0.0000)  (-0.1267,-0.0067)
3  ( 0.0800, 0.0400)  (-0.1267, 0.0067)  (-0.0267, 0.0000)
```

LFTHF/DLFTHF (Single/Double precision)

Compute the $U\,DU^H$ factorization of a complex Hermitian matrix.

Usage

```
CALL LFTHF (N, A, LDA, FAC, LDFAC, IPVT)
```

Arguments

N – Order of the matrix. (Input)

A — Complex `N` by `N` Hermitian matrix to be factored. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex `N` by `N` matrix containing information about the factorization of the Hermitian matrix `A`. (Output)
Only the upper triangle of `FAC` is used. If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization. (Output)

Comments

Informational errors

Type	Code	
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is singular.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `LFTHF` performs a $U\,DU^H$ factorization of a complex Hermitian indefinite coefficient matrix. The $U\,DU^H$ factorization is called the diagonal pivoting factorization.

`LFTHF` fails if A is singular or very close to a singular matrix.

The $U\,DU^H$ factors are returned in a form that is compatible with routines `LFIHF`, page 114, `LFSHF`, page 112, and `LFDHF`, page 117. To solve systems of equations with multiple right-hand-side vectors, use `LFTHF` followed by either `LFIHF` or `LFSHF` called once for each right-hand side. The routine `LFDHF` can be called to compute the determinant of the coefficient matrix after `LFTHF` has performed the factorization.

`LFTHF` is based on the LINPACK routine `CSIFA`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 matrix is computed. `LFTHF` is called to factor the matrix and check for singularity. `LFSHF` is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N)
      COMPLEX    A(LDA,LDA), AINV(LDA,N), FAC(LDA,LDA), RJ(N)
C
C                               Set values for A
C
C                               A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
```

```
C                                    ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                    ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
C                                  Factor A
      CALL LFTHF (N, A, LDA, FAC, LDA, IPVT)
C                                  Set up the columns of the identity
C                                  matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0, 0.0E0)
C                                  RJ is the J-th column of the identity
C                                  matrix so the following LFSHF
C                                  reference places the J-th column of
C                                  the inverse of A in the J-th column
C                                  of AINV
         CALL LFSHF (N, FAC, LDA, IPVT, RJ, AINV(1,J))
         RJ(J) = (0.0E0, 0.0E0)
   10 CONTINUE
C                                  Print the inverse
      CALL WRCRN ('AINV', N, N, AINV, LDA, 0)
      END
```

Output

```
                         AINV
                  1                   2                   3
1  ( 0.2000, 0.0000)  ( 0.1200, 0.0400)  ( 0.0800,-0.0400)
2  ( 0.1200,-0.0400)  ( 0.1467, 0.0000)  (-0.1267,-0.0067)
3  ( 0.0800, 0.0400)  (-0.1267, 0.0067)  (-0.0267, 0.0000)
```

LFSHF/DLFSHF (Single/Double precision)

Solve a complex Hermitian system of linear equations given the $U\,DU^H$ factorization of the coefficient matrix.

Usage

CALL LFSHF (N, FAC, LDFAC, IPVT, B, X)

Arguments

N — Number of equations. (Input)

FAC — Complex N by N matrix containing the factorization of the coefficient matrix A as output from routine LFCHF/DLFCHF or LFTHF/DLFTHF. (Input) Only the upper triangle of FAC is used.

***LDFAC* —** Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the factorization of A as output from routine LFCHF/DLFCHF or LFTHF/DLFTHF. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

X — Complex vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X can share the same storage locations.

Algorithm

Routine LFSHF computes the solution of a system of linear algebraic equations having a complex Hermitian indefinite coefficient matrix.

To compute the solution, the coefficient matrix must first undergo a $U DU^H$ factorization. This may be done by calling either LFCHF, page 108, or LFTHF, page 110.

LFSHF and LFIHF, page 114, both solve a linear system given its $U DU^H$ factorization. LFIHF generally takes more time and produces a more accurate answer than LFSHF. Each iteration of the iterative refinement algorithm used by LFIHF calls LFSHF.

LFSHF is based on the LINPACK routine CSISL; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. LFTHF (page 110) is called to factor the coefficient matrix. LFSHF is called to compute the three solutions for the three right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call LFCHF (page 108) to perform the factorization, and LFIHF (page 114) to compute the solutions.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), I
      COMPLEX    A(LDA,LDA), B(N,3), X(N,3), FAC(LDA,LDA)
C
C                                Set values for A and B
C
C                                A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                    ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                    ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
C                                B = (  7.0+32.0i -6.0+11.0i -2.0-17.0i )
C                                    (-39.0-21.0i -5.5-22.5i  4.0+10.0i )
C                                    ( 51.0+ 9.0i 16.0+17.0i -2.0+12.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
```

```
      DATA B/(7.0,32.0), (-39.0,-21.0), (51.0,9.0), (-6.0,11.0),
     &       (-5.5,-22.5), (16.0,17.0), (-2.0,-17.0), (4.0,10.0),
     &       (-2.0,12.0)/
C                                  Factor A
      CALL LFTHF (N, A, LDA, FAC, LDA, IPVT)
C                                  Solve for the three right-hand sides
      DO 10  I=1, 3
         CALL LFSHF (N, FAC, LDA, IPVT, B(1,I), X(1,I))
   10 CONTINUE
C                                  Print results
      CALL WRCRN ('X', N, 3, X, N, 0)
      END
```

Output

```
                           X
                 1                 2                 3
1  (  2.00,  1.00)  (  1.00,  0.00)  (  0.00, -1.00)
2  (-10.00, -1.00)  ( -3.00, -4.00)  (  0.00, -2.00)
3  (  3.00,  5.00)  ( -0.50,  3.00)  (  0.00, -3.00)
```

LFIHF/DLFIHF (Single/Double precision)

Use iterative refinement to improve the solution of a complex Hermitian system of linear equations.

Usage

`CALL LFIHF (N, A, LDA, FAC, LDFAC, IPVT, B, X, RES)`

Arguments

N — Number of equations. (Input)

A — Complex `N` by `N` matrix containing the coefficient matrix of the Hermitian linear system. (Input)
Only the upper triangle of `A` is referenced.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

FAC — Complex `N` by `N` matrix containing the factorization of the coefficient matrix `A` as output from routine `LFCHF`/`DLFCHF` or `LFTHF`/`DLFTHF`. (Input)
Only the upper triangle of `FAC` is used.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization of *A* as output from routine `LFCHF`/`DLFCHF` or `LFTHF`/`DLFTHF`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution.   (Output)

RES — Complex vector of length `N` containing the residual vector at the improved solution.   (Output)

Comments

Informational error

Type	Code	
3	3	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIHF` computes the solution of a system of linear algebraic equations having a complex Hermitian indefinite coefficient matrix.

Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo a $U\,DU^H$ factorization. This may be done by calling either `LFCHF`, page 108, or `LFTHF`, page 110.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIHF` and `LFSHF`, page 112, both solve a linear system given its $U\,DU^H$ factorization. `LFIHF` generally takes more time and produces a more accurate answer than `LFSHF`. Each iteration of the iterative refinement algorithm used by `LFIHF` calls `LFSHF`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.2 + 0.2*i* to the second element.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       RCOND
      COMPLEX    A(LDA,LDA), B(N), X(N), FAC(LDA,LDA), RES(N)
C
C
C                               Set values for A and B
C
C                               A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                   ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                   ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
C                               B = ( 7.0+32.0i -39.0-21.0i 51.0+9.0i )
```

```
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
      DATA B/(7.0,32.0), (-39.0,-21.0), (51.0,9.0)/
C                                  Set output unit number
      CALL UMACH (2, NOUT)
C                                  Factor A and compute the estimate
C                                  of the reciprocal condition number
      CALL LFCHF (N, A, LDA, FAC, LDA, IPVT, RCOND)
      WRITE (NOUT,99998) RCOND, 1.0E0/RCOND
C                                  Solve, then perturb right-hand side
      DO 10  I=1, 3
         CALL LFIHF (N, A, LDA, FAC, LDA, IPVT, B, X, RES)
C                                  Print results
         WRITE (NOUT,99999) I
         CALL WRCRN ('X', 1, N, X, 1, 0)
         CALL WRCRN ('RES', 1, N, RES, 1, 0)
         B(2) = B(2) + (0.2E0, 0.2E0)
   10 CONTINUE
C
99998 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
99999 FORMAT (//,'  For problem ', I1)
      END
```

Output

```
RCOND = 0.240
L1 Condition number =  4.175
For problem 1
                        X
              1                 2                 3
(  2.00,  1.00)  (-10.00, -1.00)  (  3.00,  5.00)

                                    RES
                      1                        2                        3
( 2.384E-07,-4.768E-07)  ( 0.000E+00,-3.576E-07)  (-1.421E-14, 1.421E-14)

For problem 2
                        X
              1                 2                 3
( 2.016, 1.032)  (-9.971,-0.971)  ( 2.973, 4.976)

                                   RES
                      1                        2                        3
( 2.098E-07,-1.764E-07)  ( 6.231E-07,-1.518E-07)  ( 1.272E-07, 4.005E-07)

For problem 3
                        X
              1                 2                 3
( 2.032, 1.064)  (-9.941,-0.941)  ( 2.947, 4.952)

                                   RES
                      1                        2                        3
( 4.196E-07,-3.529E-07)  ( 2.925E-07,-3.632E-07)  ( 2.543E-07, 3.242E-07)
```

LFDHF/DLFDHF (Single/Double precision)

Compute the determinant of a complex Hermitian matrix given the $U\,DU^H$ factorization of the matrix.

Usage

`CALL LFDHF (N, FAC, LDFAC, IPVT, DET1, DET2)`

Arguments

N — Number of equations. (Input)

FAC — Complex `N` by `N` matrix containing the factorization of the coefficient matrix `A` as output from routine `LFCHF`/`DLFCHF` or `LFTHF`/`DLFTHF`. (Input) Only the upper triangle of `FAC` is used.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the factorization of *A* as output from routine `LFCHF`/`DLFCHF` or `LFTHF`/`DLFTHF`. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value `DET1` is normalized so that 1.0 ≤ |`DET1`| < 10.0 or `DET1` = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det(*A*) = `DET1 *` $10^{\mathtt{DET2}}$.

Algorithm

Routine `LFDHF` computes the determinant of a complex Hermitian indefinite coefficient matrix. To compute the determinant, the coefficient matrix must first undergo a $U\,DU^H$ factorization. This may be done by calling either `LFCHF`, page 108, or `LFTHF`, page 110. Since det $U = \pm 1$, the formula det A = det U det D det U^H = det D is used to compute the determinant. det D is computed as the product of the determinants of its blocks.

`LFDHF` is based on the LINPACK routine `CSIDI`; see Dongarra et al. (1979).

Example

The determinant is computed for a complex Hermitian 3 × 3 matrix.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
      INTEGER    IPVT(N), NOUT
      REAL       DET1, DET2
      COMPLEX    A(LDA,LDA), FAC(LDA,LDA)
C
```

```
C                               Set values for A
C
C                               A = ( 3.0+0.0i   1.0-1.0i   4.0+0.0i )
C                                   ( 1.0+1.0i   2.0+0.0i  -5.0+1.0i )
C                                   ( 4.0+0.0i  -5.0-1.0i  -2.0+0.0i )
C
      DATA A/(3.0,0.0), (1.0,1.0), (4.0,0.0), (1.0,-1.0), (2.0,0.0),
     &       (-5.0,-1.0), (4.0,0.0), (-5.0,1.0), (-2.0,0.0)/
C                                 Factor A
      CALL LFTHF (N, A, LDA, FAC, LDA, IPVT)
C                                 Compute the determinant
      CALL LFDHF (N, FAC, LDA, IPVT, DET1, DET2)
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant is’, F5.1, ’ * 10**’, F2.0)
      END
```

Output

```
The determinant is -1.5 * 10**2.
```

LSLTR/DLSLTR (Single/Double precision)

Solve a real tridiagonal system of linear equations.

Usage

CALL LSLTR (N, C, D, E, B)

Arguments

N — Order of the tridiagonal matrix.   (Input)

C — Vector of length N containing the subdiagonal of the tridiagonal matrix in C(2) through C(N).   (Input/Output)
On output C is destroyed.

D — Vector of length N containing the diagonal of the tridiagonal matrix. (Input/Output)
On output D is destroyed.

E — Vector of length N containing the superdiagonal of the tridiagonal matrix in E(1) through E(N – 1).   (Input/Output)
On output E is destroyed.

B — Vector of length N containing the right-hand side of the linear system on entry and the solution vector on return.   (Input/Output)

Comments

Informational error

Type	Code	
4	2	An element along the diagonal became exactly zero during execution.

Algorithm

Routine LSLTR factors and solves the real tridiagonal linear system $Ax = b$. LSLTR is intended just for tridiagonal systems. The coefficient matrix does not have to be symmetric. The algorithm is Gaussian elimination with partial pivoting for numerical stability. See Dongarra (1979), LINPACK subprograms SGTSL/DGTSL, for details. When computing on vector or parallel computers the cyclic reduction algorithm, page 119, should be considered as an alternative method to solve the system.

Example

A system of $n = 4$ linear equations is solved.

```
C                                  Declaration of variables
      INTEGER    N
      PARAMETER  (N=4)
C
      REAL       B(N), C(N), D(N), E(N)
      CHARACTER  CLABEL(1)*6, FMT*8, RLABEL(1)*4
      EXTERNAL   LSLTR, WRRRL
C
      DATA FMT/’(E13.6)’/
      DATA CLABEL/’NUMBER’/
      DATA RLABEL/’NONE’/
C                                  C(*), D(*), E(*), and B(*)
C                                  contain the subdiagonal, diagonal,
C                                  superdiagonal and right hand side.
      DATA C/0.0, 0.0, -4.0, 9.0/, D/6.0, 4.0, -4.0, -9.0/
      DATA E/-3.0, 7.0, -8.0, 0.0/, B/48.0, -81.0, -12.0, -144.0/
C
C
      CALL LSLTR (N, C, D, E, B)
C                                  Output the solution.
      CALL WRRRL (’Solution:’, 1, N, B, 1, 0, FMT, RLABEL, CLABEL)
      END
```

Output

```
Solution:
        1              2              3              4
0.400000E+01  -0.800000E+01  -0.700000E+01   0.900000E+01
```

LSLCR/DLSLCR (Single/Double precision)

Compute the *L DU* factorization of a real tridiagonal matrix *A* using a cyclic reduction algorithm.

Usage

```
CALL LSLCR (N, C, A, B, IJOB, Y, U, IR, IS)
```

Arguments

N — Order of the matrix. (Input)
`N` must be greater than zero.

C — Array of size 2`N` containing the upper codiagonal of the `N` by `N` tridiagonal matrix in the entries `C`(1), …, `C`(`N` – 1). (Input/Output)

A — Array of size 2`N` containing the diagonal of the `N` by `N` tridiagonal matrix in the entries `A`(1), …, `A`(`N`). (Input/Output)

B — Array of size 2`N` containing the lower codiagonal of the `N` by `N` tridiagonal matrix in the entries `B`(1), …, `B`(`N` – 1). (Input/Output)

IJOB — Flag to direct the desired factoring or solving step. (Input)

IJOB	**Action**
1	Factor the matrix *A* and solve the system *Ax* = *y*, where *y* is stored in array `Y`.
2	Do the solve step only. Use *y* from array `Y.` (The factoring step has already been done.)
3	Factor the matrix *A* but do not solve a system.
4, 5, 6	Same meaning as with the value `IJOB` = 3. For efficiency, no error checking is done on the validity of any input value.

Y — Array of size 2`N` containing the right hand side for the system *Ax* = *y* in the order `Y`(1), …, `Y`(`N`). (Input/Output)

The vector x overwrites `Y` in storage.

U — Array of size 2`N` of flags that indicate any singularities of `A`. (Output)
`A` value `U(I)` = 1. means that a divide by zero would have occurred during the factoring. Otherwise `U(I)` = 0.

IR — Array of integers that determine the sizes of loops performed in the cyclic reduction algorithm. (Output)

IS — Array of integers that determine the sizes of loops performed in the cyclic reduction algorithm. (Output)
The sizes of `IR` and `IS` must be at least $\log_2$(`N`) + 3.

Algorithm

Routine LSLCR factors and solves the real tridiagonal linear system $Ax = y$. The matrix is decomposed in the form A = $L\ DU$, where L is unit lower triangular, U is unit upper triangular, and D is diagonal. The algorithm used for the factorization is effectively that described in Kershaw (1982). More details, tests and experiments are reported in Hanson (1990).

LSLCR is intended just for tridiagonal systems. The coefficient matrix does not have to be symmetric. The algorithm amounts to Gaussian elimination, with no pivoting for numerical stability, on the matrix whose rows and columns are permuted to a new order. See Hanson (1990) for details. The expectation is that LSLCR will outperform either LSLTR, page 118, or LSLPB, page 143, on vector or parallel computers. Its performance may be inferior for small values of n, on scalar computers, or high-performance computers with non-optimizing compilers.

Example

A system of n = 1000 linear equations is solved. The coefficient matrix is the symmetric matrix of the second difference operation, and the right-hand-side vector y is the first column of the identity matrix. Note that $a_{n,\ n}$= 1. The solution vector will be the first column of the inverse matrix of A. Then a new system is solved where y is now the last column of the identity matrix. The solution vector for this system will be the last column of the inverse matrix.

```
C                                  Declare variables
      INTEGER    LP, N, N2
      PARAMETER  (LP=12, N=1000, N2=2*N)
C
      INTEGER    I, IJOB, IR(LP), IS(LP), NOUT
      REAL       A(N2), B(N2), C(N2), U(N2), Y1(N2), Y2(N2)
      EXTERNAL   LSLCR, UMACH
C
C                                  Define matrix entries:
      DO 10  I=1, N - 1
         C(I)    = -1.E0
         A(I)    = 2.E0
         B(I)    = -1.E0
         Y1(I+1) = 0.E0
         Y2(I)   = 0.E0
   10 CONTINUE
      A(N)  = 1.E0
      Y1(1) = 1.E0
      Y2(N) = 1.E0
C
C                                  Obtain decomposition of matrix and
C                                  solve the first system:
      IJOB = 1
      CALL LSLCR (N, C, A, B, IJOB, Y1, U, IR, IS)
C
C                                  Solve the second system with the
C                                  decomposition ready:
      IJOB = 2
      CALL LSLCR (N, C, A, B, IJOB, Y2, U, IR, IS)
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ' The value of n is:  ', N
      WRITE (NOUT,*) ' Elements 1, n of inverse matrix columns 1 '//
     &               'and   n:', Y1(1), Y2(N)
      END
```

Output

```
The value of n is:    1000
Elements 1, n of inverse matrix columns 1 and   n:    1.00000   1000.000
```

LSARB/DLSARB (Single/Double precision)

Solve a real system of linear equations in band storage mode with iterative refinement.

Usage

`CALL LSARB (N, A, LDA, NLCA, NUCA, B, IPATH, X)`

Arguments

N — Number of equations.   (Input)

A — (NLCA + NUCA + 1) by N array containing the N by N banded coefficient matrix in band storage mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals of A.   (Input)

NUCA — Number of upper codiagonals of A.   (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator.   (Input)
IPATH = 1 means the system AX= B is solved.
IPATH = 2 means the system $A^T X = B$ is solved.

X — Vector of length N containing the solution to the linear system.   (Output)

Comments

1. Automatic workspace usage is

   LSARB   (2 * NLCA + NUCA + 1) * N+ 2N units, or
   DLSARB 2(2 * NLCA + NUCA + 1) * N+ 3N units.

   Workspace may be explicitly provided, if desired, by use of L2ARB/DL2ARB. The reference is

```
CALL L2ARB (N, A, LDA, NLCA, NUCA, B, IPATH, X, FAC,
            IPVT, WK)
```

The additional arguments are as follows:

FAC — Work vector of length (2 * NLCA + NUCA + 1) * N containing the *LU* factorization of A on output.

IPVT — Work vector of length N containing the pivoting information for the *LU* factorization of A on output.

WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2ARB the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSARB. Additional memory allocation for FAC and option value restoration are done automatically in LSARB. Users directly calling L2ARB can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSARB or L2ARB. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSARB temporarily replaces IVAL(2) by IVAL(1). The routine L2CRB computes the condition number if IVAL(2) = 2. Otherwise L2CRB skips this computation. LSARB restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSARB solves a system of linear algebraic equations having a real banded coefficient matrix. It first uses the routine LFCRB, page 127, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using the iterative refinement routine LFIRB, page 134.

LSARB fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if A is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSARB solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of four linear equations is solved. The coefficient matrix has real banded form with 1 upper and 1 lower codiagonal. The right-hand-side vector *b* has four elements.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, N, NLCA, NUCA
      PARAMETER  (IPATH=1, LDA=3, N=4, NLCA=1, NUCA=1)
      REAL       A(LDA,N), B(N), X(N)
      EXTERNAL   LSARB, WRRRN
C                                 Set values for A in band form, and B
C
C                                 A = (  0.0  -1.0  -2.0   2.0)
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
C                                 B = (  3.0   1.0  11.0  -2.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
      DATA B/3.0, 1.0, 11.0, -2.0/
C
      CALL LSARB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
                X
    1       2       3       4
2.000   1.000  -3.000   4.000
```

LSLRB/DLSLRB (Single/Double precision)

Solve a real system of linear equations in band storage mode without iterative refinement.

Usage

```
CALL LSLRB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
```

Arguments

N — Number of equations.   (Input)

A — (NLCA + NUCA + 1) by N array containing the N by N banded coefficient matrix in band storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX= B is solved.
IPATH = 2 means the system $A^T X$ = B is solved.

X — Vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X can share the same storage locations.

Comments

1. Automatic workspace usage is

   LSLRB (2 * NLCA + NUCA + 1) * N + 2N units, or
   DLSLRB 2(2 * NLCA + NUCA + 1) * N + 3N units.

   Workspace may be explicitly provided, if desired, by use of L2LRB/DL2LRB. The reference is

   ```
   CALL L2LRB (N, A, LDA, NLCA, NUCA, B, IPATH, X, FAC,
               IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length (2 * NLCA + NUCA + 1) * N containing the *LU* factorization of A on output. If A is not needed, A can share the first (NLCA + NUCA + 1) * N storage locations with FAC.

   IPVT — Work vector of length N containing the pivoting information for the *LU* factorization of A on output.

   WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LRB the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSLRB. Additional memory allocation for FAC and option value restoration are done automatically in LSLRB. Users directly calling L2LRB can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLRB or L2LRB. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLRB temporarily replaces IVAL(2) by IVAL(1). The routine L2CRB computes the condition number if IVAL(2) = 2. Otherwise L2CRB skips this computation. LSLRB restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLRB solves a system of linear algebraic equations having a real banded coefficient matrix. It first uses the routine LFCRB, page 127, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using LFSRB, page 132. LSLRB fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This occurs only if *A* is singular or very close to a singular matrix. If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that LSARB, page 122, be used.

Example

A system of four linear equations is solved. The coefficient matrix has real banded form with 1 upper and 1 lower codiagonal. The right-hand-side vector *b* has four elements.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, N, NLCA, NUCA
      PARAMETER  (IPATH=1, LDA=3, N=4, NLCA=1, NUCA=1)
      REAL       A(LDA,N), B(N), X(N)
      EXTERNAL   LSLRB, WRRRN
C                                 Set values for A in band form, and B
C
C                                 A = (  0.0  -1.0  -2.0   2.0)
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
C                                 B = (  3.0   1.0  11.0  -2.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
      DATA B/3.0, 1.0, 11.0, -2.0/
C
      CALL LSLRB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
              X
   1       2       3       4
2.000   1.000  -3.000   4.000
```

LFCRB/DLFCRB (Single/Double precision)

Compute the *LU* factorization of a real matrix in band storage mode and estimate its L_1 condition number.

Usage

`CALL LFCRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)`

Arguments

N — Order of the matrix. (Input)

A — (NLCA + NUCA + 1) by N array containing the N by N matrix in band storage mode to be factored. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

FAC — (2 * NLCA + NUCA + 1) by N array containing the *LU* factorization of the matrix A. (Output)
If A is not needed, A can share the first (NLCA + NUCA + 1) * N locations with FAC
.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of A. (Output)

Comments

1. Automatic workspace usage is

`LFCRB` `N` units, or
`DLFCRB` 2`N` units.

Workspace may be explicitly provided, if desired, by use of `L2CRB`/`DL2CRB`. The reference is

```
CALL L2CRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT,
            RCOND, WK)
```

The additional argument is

WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is singular.

Algorithm

Routine `LFCRB` performs an *LU* factorization of a real banded coefficient matrix. It also estimates the condition number of the matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

`LSCRB` fails if U, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if A is singular or very close to a singular matrix. The *LU* factors are returned in a form that is compatible with routines `LFIRB`, page 134, `LFSRB`, page 132, and `LFDRB`, page 136. To solve systems of equations with multiple right-hand-side vectors, use `LFCRB` followed by either `LFIRB` or `LFSRB` called once for each right-hand side. The routine `LFDRB` can be called to compute the determinant of the coefficient matrix after `LFCRB` has performed the factorization.

Let F be the matrix `FAC`, let m_l= `NLCA` and let m_u = `NUCA`. The first $m_l + m_u + 1$ rows of F contain the triangular matrix U in band storage form. The lower m_l rows of F contain the multipliers needed to reconstruct L^{-1} .

`LFCRB` is based on the LINPACK routine `SGBCO`; see Dongarra et al. (1979). `SGBCO` uses unscaled partial pivoting.

Example

The inverse of a 4 × 4 band matrix with one upper and one lower codiagonal is computed. LFCRB is called to factor the matrix and to check for singularity or ill-conditioning. LFIRB (page 134) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)

      INTEGER    IPVT(N)
      REAL       A(LDA,N), AINV(N,N), FAC(LDFAC,N), RCOND, RJ(N), RES(N)
      EXTERNAL   LFCRB, LFIRB, SSET, UMACH, WRRRN
C                                 Set values for A in band form
C                                 A = (  0.0  -1.0  -2.0   2.0)
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
C
      CALL LFCRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the reciprocal condition number
C                                 and the L1 condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIRB
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RJ, IPATH,
     &               AINV(1,J), RES)
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print results
      CALL WRRRN ('AINV', N, N, AINV, N, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.065
L1 Condition number = 15.351

               AINV
        1       2       3       4
1  -1.000  -1.000   0.400  -0.800
2  -3.000  -2.000   0.800  -1.600
3   0.000   0.000  -0.200   0.400
4   0.000   0.000   0.400   0.200
```

LFTRB/DLFTRB (Single/Double precision)

Compute the *LU* factorization of a real matrix in band storage mode.

Usage

```
CALL LFTRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
```

Arguments

N — Order of the matrix. (Input)

A — (NLCA + NUCA + 1) by N array containing the N by N matrix in band storage mode to be factored. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

FAC — (2 * NLCA + NUCA + 1) by N array containing the *LU* factorization of the matrix A. (Output)
If A is not needed, A can share the first (NLCA + NUCA + 1) * N locations with FAC
.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

   LFTRB N units, or
   DLFTRB 2N units.

   Workspace may be explicitly provided, if desired, by use of L2TRB/DL2TRB. The reference is

   ```
   CALL L2TRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT,
               WK)
   ```

   The additional argument is

   WK — Work vector of length N used for scaling.

2 Informational error

Type	Code	
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

21 The performance of the *LU* factorization may improve on high-performance computers if the blocking factor, NB, is increased. The current version of the routine allows NB to be reset to a value no larger than 32. Default value is NB = 1.

Algorithm

The routine LFTRB performs an *LU* factorization of a real banded coefficient matrix using Gaussian elimination with partial pivoting. *A* failure occurs if *U*, the upper triangular factor, has a zero diagonal element. This can happen if *A* is close to a singular matrix. The *LU* factors are returned in a form that is compatible with routines LFIRB, page 134, LFSRB, page 132, and LFDRB, page 136. To solve systems of equations with multiple right-hand-side vectors, use LFTRB followed by either LFIRB or LFSRB called once for each right-hand side. The routine LFDRB can be called to compute the determinant of the coefficient matrix after LFTRB has performed the factorization

Let m_l = NLCA, and let m_u = NUCA. The first $m_l + m_u + 1$ rows of FAC contain the triangular matrix *U* in band storage form. The next m_l rows of FAC contain the multipliers needed to produce L^{-1}.

The routine LFTRB is based on the the blocked *LU* factorization algorithm for banded linear systems given in Du Croz, et al. (1990). Level-3 BLAS invocations were replaced by in-line loops. The blocking factor *nb* has the default value 1 in LFTRB. It can be reset to any positive value not exceeding 32.

Example

A linear system with multiple right-hand sides is solved. LFTRB is called to factor the coefficient matrix. LFSRB (page 132) is called to compute the two solutions for the two right-hand sides. In this case the coefficient matrix is assumed to be appropriately scaled. Otherwise, it may be better to call routine LFCRB (page 127) to perform the factorization, and LFIRB (page 134) to compute the solutions.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       A(LDA,N), B(N,2), FAC(LDFAC,N), X(N,2)
      EXTERNAL   LFTRB, LFSRB, WRRRN
C                                 Set values for A in band form, and B
C
C                                 A = (  0.0  -1.0  -2.0   2.0)
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
C                                 B = ( 12.0 -17.0)
C                                     (-19.0  23.0)
C                                     (  6.0   5.0)
```

```
C                                    (  8.0   5.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
      DATA B/12.0, -19.0, 6.0, 8.0, -17.0, 23.0, 5.0, 5.0/
C                                Compute factorization
      CALL LFTRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                Solve for the two right-hand sides
      DO 10  J=1, 2
         CALL LFSRB (N, FAC, LDFAC, NLCA, NUCA, IPVT, B(1,J), IPATH,
     &               X(1,J))
   10 CONTINUE
C                                Print results
      CALL WRRRN ('X', N, 2, X, N, 0)
C
      END
```

Output

```
        X
        1       2
1   3.000  -8.000
2  -6.000   1.000
3   2.000   1.000
4   4.000   3.000
```

LFSRB/DLFSRB (Single/Double precision)

Solve a real system of linear equations given the *LU* factorization of the coefficient matrix in band storage mode.

Usage

CALL LFSRB (N, FAC, LDFAC, NLCA, NUCA, IPVT, B, IPATH, X)

Arguments

N — Number of equations.   (Input)

FAC — (2 ∗ NLCA + NUCA + 1) by N array containing the *LU* factorization of the coefficient matrix A as output from routine LFCRB/DLFCRB or LFTRB/DLFTRB. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals of A.   (Input)

NUCA — Number of upper codiagonals of A.   (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization of A as output from routine LFCRB/DLFCRB or LFTRB/DLFTRB. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system `AX` = `B` is solved.

`IPATH` = 2 means the system $\mathtt{A}^T\mathtt{X}$ = `B` is solved.

X — Vector of length `N` containing the solution to the linear system. (Output)
If `B` is not needed, `B` and `X` can share the same storage locations.

Algorithm

Routine `LFSRB` computes the solution of a system of linear algebraic equations having a real banded coefficient matrix. To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCRB`, page 127, or `LFTRB`, page 130. The solution to $Ax = b$ is found by solving the banded triangular systems $Ly = b$ and $Ux = y$. The forward elimination step consists of solving the system $Ly = b$ by applying the same permutations and elimination operations to b that were applied to the columns of A in the factorization routine. The backward substitution step consists of solving the banded triangular system $Ux = y$ for x.

`LFSRB` and `LFIRB`, page 134, both solve a linear system given its *LU* factorization. `LFIRB` generally takes more time and produces a more accurate answer than `LFSRB`. Each iteration of the iterative refinement algorithm used by `LFIRB` calls `LFSRB`.

`LFSRB` is based on the LINPACK routine `SGBSL`; see Dongarra et al. (1979).

Example

The inverse is computed for a real banded 4 × 4 matrix with one upper and one lower codiagonal. The input matrix is assumed to be well-conditioned, hence `LFTRB` (page 130) is used rather than `LFCRB`.

```
C                                  Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       A(LDA,N), AINV(N,N), FAC(LDFAC,N), RJ(N)
      EXTERNAL   LFSRB, LFTRB, SSET, WRRRN
C                                  Set values for A in band form
C                                  A = (  0.0  -1.0  -2.0   2.0)
C                                      (  2.0   1.0  -1.0   1.0)
C                                      ( -3.0   0.0   2.0   0.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
C
      CALL LFTRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                  Set up the columns of the identity
C                                  matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
```

```
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSRB
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSRB (N, FAC, LDFAC, NLCA, NUCA, IPVT, RJ, IPATH,
     &               AINV(1,J))
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print results
      CALL WRRRN (’AINV’, N, N, AINV, N, 0)
C
      END
```

Output

```
              AINV
        1       2       3       4
1  -1.000  -1.000   0.400  -0.800
2  -3.000  -2.000   0.800  -1.600
3   0.000   0.000  -0.200   0.400
4   0.000   0.000   0.400   0.200
```

LFIRB/DLFIRB (Single/Double precision)

Use iterative refinement to improve the solution of a real system of linear equations in band storage mode.

Usage

```
CALL LFIRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, B,
            IPATH, X,RES)
```

Arguments

N — Number of equations.   (Input)

A — (NUCA +NLCA +1) by N array containing the N by N banded coefficient matrix in band storage mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals of A.   (Input)

NUCA — Number of upper codiagonals of A.   (Input)

FAC — (2 * NLCA +NUCA +1) by N array containing the *LU* factorization of the matrix A as output from routines LFCRB/DLFCRB or LFTRB/DLFTRB.   (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization of `A` as output from routine `LFCRB`/`DLFCRB` or `LFTRB`/`DLFTRB`. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system `AX` = `B` is solved.

`IPATH` = 2 means the system $A^T X$ = `B` is solved.

X — Vector of length `N` containing the solution to the linear system. (Output)

RES — Vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	2	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIRB` computes the solution of a system of linear algebraic equations having a real banded coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCRB`, page 127, or `LFTRB`, page 130.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIRB` and `LFSRB`, page 132, both solve a linear system given its *LU* factorization. `LFIRB` generally takes more time and produces a more accurate answer than `LFSRB`. Each iteration of the iterative refinement algorithm used by `LFIRB` calls `LFSRB`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.5 to the second element.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       A(LDA,N), B(N), FAC(LDFAC,N), RCOND, RES(N), X(N)
      EXTERNAL   LFCRB, LFIRB, UMACH, WRRRN
```

```
C                                 Set values for A in band form, and B
C
C                                 A = (  0.0  -1.0  -2.0   2.0)
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
C                                 B = (  3.0   5.0   7.0  -9.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
      DATA B/3.0, 5.0, 7.0, -9.0/
C
      CALL LFCRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the reciprocal condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Solve the three systems
      DO 10  J=1, 3
         CALL LFIRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, B,
     &               IPATH, X, RES)
C                                 Print results
         CALL WRRRN ('X', 1, N, X, 1, 0)
C                                 Perturb B by adding 0.5 to B(2)
         B(2) = B(2) + 0.5E0
   10 CONTINUE
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.065
L1 Condition number = 15.351
                X
    1       2       3       4
2.000   1.000  -5.000   1.000

                X
    1       2       3       4
1.500   0.000  -5.000   1.000

                X
    1       2       3       4
1.000  -1.000  -5.000   1.000
```

LFDRB/DLFDRB (Single/Double precision)

Compute the determinant of a real matrix in band storage mode given the *LU* factorization of the matrix.

Usage

`CALL LFDRB (N, FAC, LDFAC, NLCA, NUCA, IPVT, DET1, DET2)`

Arguments

N — Order of the matrix. (Input)

FAC — (2 * NLCA + NUCA + 1) by N array containing the *LU* factorization of the matrix A as output from routine LFTRB/DLFTRB or LFCRB/DLFCRB. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals in matrix A. (Input)

NUCA — Number of upper codiagonals in matrix A. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization as output from routine LFTRB/DLFTRB or LFCRB/DLFCRB. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that 1.0 ≤ |DET1| < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det(A) = DET1 * 10^{DET2}.

Algorithm

Routine LFDRB computes the determinant of a real banded coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either LFCRB, page 127, or LFTRB, page 130. The formula det A = det L det U is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det U = \prod\nolimits_{i=1}^{N} U_{ii}$$

(The matrix U is stored in the upper NUCA + NLCA + 1 rows of FAC as a banded matrix.) Since L is the product of triangular matrices with unit diagonals and of permutation matrices, det $L = (-1)^k$, where k is the number of pivoting interchanges.

LFDRB is based on the LINPACK routine CGBDI; see Dongarra et al. (1979).

Example

The determinant is computed for a real banded 4 × 4 matrix with one upper and one lower codiagonal.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       A(LDA,N), DET1, DET2, FAC(LDFAC,N)
      EXTERNAL   LFTRB, LFDRB, UMACH
C                                 Set values for A in band form
C                                 A = (  0.0  -1.0  -2.0   2.0)
```

```
C                                     (  2.0   1.0  -1.0   1.0)
C                                     ( -3.0   0.0   2.0   0.0)
C
      DATA A/0.0, 2.0, -3.0, -1.0, 1.0, 0.0, -2.0, -1.0, 2.0,
     &       2.0, 1.0, 0.0/
C
      CALL LFTRB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                 Compute the determinant
      CALL LFDRB (N, FAC, LDFAC, NLCA, NUCA, IPVT, DET1, DET2)
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
99999 FORMAT (' The determinant of A is ', F6.3, ' * 10**', F2.0)
      END
```

Output

```
The determinant of A is  5.000 * 10**0.
```

LSAQS/DLSAQS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations in band symmetric storage mode with iterative refinement.

Usage

`CALL LSAQS (N, A, LDA, NCODA, B, X)`

Arguments

N — Number of equations. (Input)

A — `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band coefficient matrix in band symmetric storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of `A`. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSAQS` `N(NCODA` + 1) + `N` units, or
   `DLSAQS` 2`N(NCODA` + 1) + 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2AQS/DL2AQS`. The reference is

   `CALL L2AQS (N, A, LDA, NCODA, B, X, FAC, WK)`

The additional arguments are as follows:

FAC — Work vector of length NCODA + 1 by N containing the $R^T R$ factorization of A in band symmetric storage form on output.

WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is not positive definite.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2AQS the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSAQS. Additional memory allocation for FAC and option value restoration are done automatically in LSAQS.

Users directly calling L2AQS can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSAQS or L2AQS. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSAQS temporarily replaces IVAL(2) by IVAL(1). The routine L2CQS computes the condition number if IVAL(2) = 2. Otherwise L2CQS skips this computation. LSAQS restores the option. Default values for the option are IVAL(*) = 1,2.

Algorithm

Routine LSAQS solves a system of linear algebraic equations having a real symmetric positive definite band coefficient matrix. It first uses the routine LFCQS, page 145, to compute an $R^T R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. R is an upper triangular band matrix. The solution of the linear system is then found using the iterative refinement routine LFIQS, page 151.

LSAQS fails if any submatrix of R is not positive definite, if R has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSAQS solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of four linear equations is solved. The coefficient matrix has real positive definite band form, and the right-hand-side vector *b* has four elements.

```
C                                  Declare variables
      INTEGER    LDA, N, NCODA
      PARAMETER  (LDA=3, N=4, NCODA=2)
      REAL       A(LDA,N), B(N), X(N)
C
C                        Set values for A in band symmetric form, and B
C
C                                  A = (  0.0   0.0  -1.0   1.0 )
C                                      (  0.0   0.0   2.0  -1.0 )
C                                      (  2.0   4.0   7.0   3.0 )
C
C                                  B = (  6.0 -11.0 -11.0  19.0 )
C
      DATA A/2*0.0, 2.0, 2*0.0, 4.0, -1.0, 2.0, 7.0, 1.0, -1.0, 3.0/
      DATA B/6.0, -11.0, -11.0, 19.0/
C                                  Solve A*X = B
      CALL LSAQS (N, A, LDA, NCODA, B, X)
C                                  Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
                X
      1        2        3        4
     4.000   -6.000    2.000    9.000
```

LSLQS/DLSLQS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations in band symmetric storage mode without iterative refinement.

Usage

CALL LSLQS (N, A, LDA, NCODA, B, X)

Arguments

N — Number of equations. (Input)

A — NCODA + 1 by N array containing the N by N positive definite band symmetric coefficient matrix in band symmetric storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of A. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

X — Vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSLQS N(NCODA + 1) + N units, or
   DLSLQS 2N(NCODA + 1) + 2N units.

   Workspace may be explicitly provided, if desired, by use of L2LQS/DL2LQS. The reference is

   ```
   CALL L2LQS (N, A, LDA, NCODA, B, X, FAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length NCODA + 1 by N containing the $R^T R$ factorization of A in band symmetric form on output. If A is not needed, A and FAC can share the same storage locations.

   WK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is not positive definite.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LQS the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSLQS. Additional memory allocation for FAC and option value restoration are done automatically in LSLQS. Users directly calling L2LQS can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLQS or L2LQS. Default values for the option are IVAL(*) = 1,16,0,1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLQS temporarily replaces IVAL(2) by IVAL(1). The routine L2CQS computes the condition number if IVAL(2) = 2. Otherwise L2CQS skips this computation. LSLQS restores the option. Default values for the option are IVAL(*) = 1,2.

Algorithm

Routine LSLQS solves a system of linear algebraic equations having a real symmetric positive definite band coefficient matrix. It first uses the routine LFCQS, page 145, to compute an R^TR Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. *R* is an upper triangular band matrix. The solution of the linear system is then found using the routine LFSQS, page 149.

LSLQS fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* is very close to a singular matrix or to a matrix which is not positive definite.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in *A* can cause very large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that LSAQS, page 138, be used.

Example

A system of four linear equations is solved. The coefficient matrix has real positive definite band form and the right-hand-side vector *b* has four elements.

```
C                                 Declare variables
      INTEGER    LDA, N, NCODA
      PARAMETER  (LDA=3, N=4, NCODA=2)
      REAL       A(LDA,N), B(N), X(N)
C
C                       Set values for A in band symmetric form, and B
C
C                                 A = (  0.0   0.0  -1.0   1.0 )
C                                     (  0.0   0.0   2.0  -1.0 )
C                                     (  2.0   4.0   7.0   3.0 )
C
C                                 B = (  6.0 -11.0 -11.0  19.0 )
C
      DATA A/2*0.0, 2.0, 2*0.0, 4.0, -1.0, 2.0, 7.0, 1.0, -1.0, 3.0/
      DATA B/6.0, -11.0, -11.0, 19.0/
C                                 Solve A*X = B
      CALL LSLQS (N, A, LDA, NCODA, B, X)
C                                 Print results
      CALL WRRRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                   X
     1        2         3         4
   4.000  -6.000   2.000   9.000
```

LSLPB/DLSLPB (Single/Double precision)

Compute the $R^T DR$ Cholesky factorization of a real symmetric positive definite matrix *A* in codiagonal band symmetric storage mode. Solve a system *Ax* = *b*.

Usage

```
CALL LSLPB (N, A, LDA, NCODA, IJOB, U)
```

Arguments

N — Order of the matrix. (Input)
Must satisfy `N` > 0.

A — Array containing the `N` by `N` positive definite band coefficient matrix and right hand side in codiagonal band symmetric storage mode. (Input/Output)
The number of array columns must be at least `NCODA` + 2. The number of columns is not an input to this subprogram.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)
Must satisfy `LDA` ≥ `N` + `NCODA`.

NCODA — Number of upper codiagonals of matrix `A`. (Input)
Must satisfy `NCODA` ≥ 0 and `NCODA` < `N`.

IJOB — Flag to direct the desired factorization or solving step. (Input)

`IJOB`	**Meaning**
1	factor the matrix `A` and solve the system `A`*x* = *b*, where *b* is stored in column `NCODA` + 2 of array `A`. The vector *x* overwrites *b* in storage.
2	solve step only. Use *b* as column `NCODA` + 2 of `A`. (The factorization step has already been done.) The vector *x* overwrites *b* in storage.
3	factor the matrix `A` but do not solve a system.
4,5,6	same meaning as with the value `IJOB` = 3. For efficiency, no error checking is done on values `LDA`, `N`, `NCODA`, and `U(*)`.

U — Array of flags that indicate any singularities of `A`, namely loss of positive-definiteness of a leading minor. (Output)
A value `U(I)` = 0. means that the leading minor of dimension `I` is not positive-definite. Otherwise, `U(I)` = 1.

Comments

1. Automatic workspace usage is

   `LSLPB` `NCODA` units, or
   `DLSLPB` 2 * `NCODA` units.

   Workspace may be explicitly provided, if desired, by use of `L2LPB/DL2LPB` The reference is

   `CALL L2LPB (N, A, LDA, NCODA, IJOB, U, WK)`

   The additional argument is

   WK — Work vector of length `NCODA`.

2. Informational error

Type	Code	
4	2	The input matrix is not positive definite.

Algorithm

Routine `LSLPB` factors and solves the symmetric positive definite banded linear system $Ax = b$. The matrix is factored so that $A = R^T DR$, where R is unit upper triangular and D is diagonal. The reciprocals of the diagonal entries of D are computed and saved to make the solving step more efficient. Errors will occur if D has a non-positive diagonal element. Such events occur only if A is very close to a singular matrix or is not positive definite.

`LSLPB` is efficient for problems with a small band width. The particular cases `NCODA` = 0, 1, 2 are done with special loops within the code. These cases will give good performance. See Hanson (1989) for details. When solving tridiagonal systems, `NCODA` = 1 , the cyclic reduction code `LSLCR`, page 119, should be considered as an alternative. The expectation is that `LSLCR` will outperform `LSLPB` on vector or parallel computers. It may be inferior on scalar computers or even parallel computers with non-optimizing compilers.

Example

A system of four linear equations is solved. The coefficient matrix has real positive definite codiagonal band form and the right-hand-side vector b has four elements.

```
C                                 Declare variables
      INTEGER    LDA, N, NCODA
      PARAMETER  (N=4, NCODA=2, LDA=N+NCODA)
C
      INTEGER    IJOB
      REAL       A(LDA,NCODA+2), U(N)
      EXTERNAL   LSLPB, WRRRN
C
C                                 Set values for A and right side in
C                                 codiagonal band symmetric form:
C
C                            A    =   (  *     *      *       * )
```

```
C                                     (  *      *       *        * )
C                                     (2.0      *       *      6.0)
C                                     (4.0    0.0       *    -11.0)
C                                     (7.0    2.0    -1.0    -11.0)
C                                     (3.0   -1.0     1.0     19.0)
C
      DATA ((A(I+NCODA,J),I=1,N),J=1,NCODA+2)/2.0, 4.0, 7.0, 3.0, 0.0,
     &     0.0, 2.0, -1.0, 0.0, 0.0, -1.0, 1.0, 6.0, -11.0, -11.0,
     &     19.0/
C                                 Factor and solve A*x = b.
      IJOB = 1
      CALL LSLPB (N, A, LDA, NCODA, IJOB, U)
C                                 Print results
      CALL WRRRN (’X’, 1, N, A(NCODA+1,NCODA+2), 1, 0)
      END
```

Output

```
                 X
      1       2        3       4
     4.000  -6.000   2.000   9.000
```

LFCQS/DLFCQS (Single/Double precision)

Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode and estimate its L_1 condition number.

Usage

`CALL LFCQS (N, A, LDA, NCODA, FAC, LDFAC, RCOND)`

Arguments

N — Order of the matrix. (Input)

A — `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band coefficient matrix in band symmetric storage mode to be factored. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of `A`. (Input)

FAC — `NCODA` + 1 by `N` array containing the $R^T R$ factorization of the matrix `A` in band symmetric form. (Output)
If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCQS` `N` units, or
   `DLFCQS` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CQS/DL2CQS.` The reference is

   ```
   CALL L2CQS (N, A, LDA, NCODA, FAC, LDFAC, RCOND, WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. Informational errors

Type	Code	
3	3	The input matrix is algorithmically singular.
4	2	The input matrix is not positive definite.

Algorithm

Routine `LFCQS` computes an R^TR Cholesky factorization and estimates the condition number of a real symmetric positive definite band coefficient matrix. R is an upper triangular band matrix.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

`LFCQS` fails if any submatrix of R is not positive definite or if R has a zero diagonal element. These errors occur only if A is very close to a singular matrix or to a matrix which is not positive definite.

The R^TR factors are returned in a form that is compatible with routines `LFIQS`, page 151, `LFSQS`, page 149, and `LFDQS`, page 153. To solve systems of equations with multiple right-hand-side vectors, use `LFCQS` followed by either `LFIQS` or `LFSQS` called once for each right-hand side. The routine `LFDQS` can be called to compute the determinant of the coefficient matrix after `LFCQS` has performed the factorization.

`LFCQS` is based on the LINPACK routine `SPBCO`; see Dongarra et al. (1979).

Example

The inverse of a 4 × 4 symmetric positive definite band matrix with one codiagonal is computed. LFCQS is called to factor the matrix and to check for nonpositive definiteness or ill-conditioning. LFIQS (page 151) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA, NOUT
      PARAMETER  (LDA=2, LDFAC=2, N=4, NCODA=1)
      REAL       A(LDA,N), AINV(N,N), RCOND, FAC(LDFAC,N),
     &           RES(N), RJ(N)
C
C                              Set values for A in band symmetric form
C
C                                 A = (  0.0   1.0   1.0   1.0 )
C                                     (  2.0   2.5   2.5   2.0 )
C
      DATA A/0.0, 2.0, 1.0, 2.5, 1.0, 2.5, 1.0, 2.0/
C                                 Factor the matrix A
      CALL LFCQS (N, A, LDA, NCODA, FAC, LDFAC, RCOND)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIQS
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIQS (N, A, LDA, NCODA, FAC, LDFAC, RJ, AINV(1,J), RES)
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
      CALL WRRRN ('AINV', N, N, AINV, N, 0)
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.160
L1 Condition number =  6.248
                 AINV
          1         2         3         4
    1   0.6667  -0.3333   0.1667  -0.0833
    2  -0.3333   0.6667  -0.3333   0.1667
    3   0.1667  -0.3333   0.6667  -0.3333
    4  -0.0833   0.1667  -0.3333   0.6667
```

LFTQS/DLFTQS (Single/Double precision)

Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode.

Usage

```
CALL LFTQS (N, A, LDA, NCODA, FAC, LDFAC)
```

Arguments

N — Order of the matrix. (Input)

A — `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band coefficient matrix in band symmetric storage mode to be factored. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of `A`. (Input)

FAC — `NCODA` + 1 by `N` array containing the $R^T R$ factorization of the matrix `A`. (Output)
If `A` s not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

Comments

Informational error

Type	Code	
4	2	The input matrix is not positive definite.

Algorithm

Routine `LFTQS` computes an $R^T R$ Cholesky factorization of a real symmetric positive definite band coefficient matrix. *R* is an upper triangular band matrix.

`LFTQS` fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* is very close to a singular matrix or to a matrix which is not positive definite.

The $R^T R$ factors are returned in a form that is compatible with routines `LFIQS`, page 151, `LFSQS`, page 149, and `LFDQS`, page 153. To solve systems of equations with multiple right hand-side vectors, use `LFTQS` followed by either `LFIQS` or `LFSQS` called once for each right-hand side. The routine `LFDQS` can be called to compute the determinant of the coefficient matrix after `LFTQS` has performed the factorization.

`LFTQS` is based on the LINPACK routine `CPBFA`; see Dongarra et al. (1979).

Example

The inverse of a 3 × 3 matrix is computed. `LFTQS` is called to factor the matrix and to check for nonpositive definiteness. `LFSQS` (page 149) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA
      PARAMETER  (LDA=2, LDFAC=2, N=4, NCODA=1)
      REAL       A(LDA,N), AINV(N,N), FAC(LDFAC,N), RJ(N)
C
C                              Set values for A in band symmetric form
C
C                                 A = (  0.0   1.0   1.0   1.0 )
C                                     (  2.0   2.5   2.5   2.0 )
C
      DATA A/0.0, 2.0, 1.0, 2.5, 1.0, 2.5, 1.0, 2.0/
C                                 Factor the matrix A
      CALL LFTQS (N, A, LDA, NCODA, FAC, LDFAC)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL SSET (N, 0.0E0, RJ, 1)
      DO 10  J=1, N
         RJ(J) = 1.0E0
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSQS
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSQS (N, FAC, LDFAC, NCODA, RJ, AINV(1,J))
         RJ(J) = 0.0E0
   10 CONTINUE
C                                 Print the results
      CALL WRRRN ('AINV', N, N, AINV, N, 1)
      END
```

Output

```
                 AINV
         1         2         3         4
1   0.6667   -0.3333    0.1667   -0.0833
2             0.6667   -0.3333    0.1667
3                       0.6667   -0.3333
4                                 0.6667
```

LFSQS/DLFSQS (Single/Double precision)

Solve a real symmetric positive definite system of linear equations given the factorization of the coefficient matrix in band symmetric storage mode.

Usage

`CALL LFSQS (N, FAC, LDFAC, NCODA, B, X)`

Arguments

N — Number of equations. (Input)

FAC — NCODA + 1 by N array containing the $R^T R$ factorization of the positive definite band matrix A in band symmetric storage mode as output from subroutine LFCQS/DLFCQS or LFTQS/DLFTQS. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of A. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

X — Vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X an share the same storage locations.

Comments

Informational error

Type	Code	
4	1	The factored matrix is singular.

Algorithm

This routine computes the solution for a system of linear algebraic equations having a real symmetric positive definite band coefficient matrix. To compute the solution, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either LFCQS, page 145, or LFTQS, page 148. R is an upper triangular band matrix.

The solution to $Ax = b$ is found by solving the triangular systems $R^T y = b$ and $Rx = y$.

LFSQS and LFIQS, page 151, both solve a linear system given its $R^T R$ factorization. LFIQS generally takes more time and produces a more accurate answer than LFSQS. Each iteration of the iterative refinement algorithm used by LFIQS calls LFSQS.

LFSQS is based on the LINPACK routine SPBSL; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. LFTQS (page 148) is called to factor the coefficient matrix. LFSQS is called to compute the four solutions for the four right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call LFCQS (page 145) to perform the factorization, and LFIQS (page 151) to compute the solutions.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA
      PARAMETER  (LDA=3, LDFAC=3, N=4, NCODA=2)
      REAL       A(LDA,N), B(N,4), FAC(LDFAC,N), X(N,4)
C
C
C                       Set values for A in band symmetric form, and B
C
C                                 A = (  0.0   0.0  -1.0   1.0 )
C                                     (  0.0   0.0   2.0  -1.0 )
C                                     (  2.0   4.0   7.0   3.0 )
C
C                                 B = (  4.0  -3.0   9.0  -1.0 )
C                                     (  6.0  10.0  29.0   3.0 )
C                                     ( 15.0  12.0  11.0   6.0 )
C                                     ( -7.0   1.0  14.0   2.0 )
C
      DATA A/2*0.0, 2.0, 2*0.0, 4.0, -1.0, 2.0, 7.0, 1.0, -1.0, 3.0/
      DATA B/4.0, 6.0, 15.0, -7.0, -3.0, 10.0, 12.0, 1.0, 9.0, 29.0,
     &       11.0, 14.0, -1.0, 3.0, 6.0, 2.0/
C                                 Factor the matrix A
      CALL LFTQS (N, A, LDA, NCODA, FAC, LDFAC)
C                                 Compute the solutions
      DO 10  I=1, 4
         CALL LFSQS (N, FAC, LDFAC, NCODA, B(1,I), X(1,I))
   10 CONTINUE
C                                 Print solutions
      CALL WRRRN ('X', N, 4, X, N, 0)
C
      END
```

Output

```
                X
        1       2       3       4
1   3.000  -1.000   5.000   0.000
2   1.000   2.000   6.000   0.000
3   2.000   1.000   1.000   1.000
4  -2.000   0.000   3.000   1.000
```

LFIQS/DLFIQS (Single/Double precision)

Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations in band symmetric storage mode.

Usage

CALL LFIQS (N, A, LDA, NCODA, FAC, LDFAC, B, X, RES)

Arguments

N — Number of equations. (Input)

A — NCODA + 1 by N array containing the N by N positive definite band coefficient matrix in band symmetric storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of `A`. (Input)

FAC — `NCODA` + 1 by `N` array containing the $R^T R$ factorization of the matrix `A` as output from routine `LFCQS`/`DLFCQS` or `LFTQS`/`DLFTQS`. (Input)

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

X — Vector of length `N` containing the solution to the system. (Output)

RES — Vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	4	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIQS` computes the solution of a system of linear algebraic equations having a real symmetric positive-definite band coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either IMSL routine `LFCQS`, page 145, or `LFTQS`, page 148.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIQS` and `LFSQS`, page 149, both solve a linear system given its $R^T R$ factorization. `LFIQS` generally takes more time and produces a more accurate answer than `LFSQS`. Each iteration of the iterative refinement algorithm used by `LFIQS` calls `LFSQS`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding 0.5 to the second element.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA, NOUT
      PARAMETER  (LDA=2, LDFAC=2, N=4, NCODA=1)
      REAL       A(LDA,N), B(N), RCOND, FAC(LDFAC,N), RES(N,3),
     &           X(N,3)
C
C                       Set values for A in band symmetric form, and B
C
C                                 A = (  0.0   1.0   1.0   1.0 )
C                                     (  2.0   2.5   2.5   2.0 )
C
C                                 B = (  3.0   5.0   7.0   4.0 )
C
      DATA A/0.0, 2.0, 1.0, 2.5, 1.0, 2.5, 1.0, 2.0/
      DATA B/3.0, 5.0, 7.0, 4.0/
C                                 Factor the matrix A
      CALL LFCQS (N, A, LDA, NCODA, FAC, LDFAC, RCOND)
C                                 Print the estimated condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Compute the solutions
      DO 10  I=1, 3
         CALL LFIQS (N, A, LDA, NCODA, FAC, LDFAC, B, X(1,I), RES(1,I))
         B(2) = B(2) + 0.5E0
   10 CONTINUE
C                                 Print solutions and residuals
      CALL WRRRN ('X', N, 3, X, N, 0)
      CALL WRRRN ('RES', N, 3, RES, N, 0)
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.160
L1 Condition number =  6.248
             X
        1       2       3
1   1.167   1.000   0.833
2   0.667   1.000   1.333
3   2.167   2.000   1.833
4   0.917   1.000   1.083

                 RES
            1            2            3
1   7.947E-08   0.000E+00   9.934E-08
2   7.947E-08   0.000E+00   3.974E-08
3   7.947E-08   0.000E+00   1.589E-07
4  -3.974E-08   0.000E+00  -7.947E-08
```

LFDQS/DLFDQS (Single/Double precision)

Compute the determinant of a real symmetric positive definite matrix given the R^TR Cholesky factorization of the band symmetric storage mode.

Usage

```
CALL LFDQS (N, FAC, LDFAC, NCODA, DET1, DET2)
```

Arguments

N — Number of equations. (Input)

FAC — NCODA + 1 by N array containing the $R^T R$ factorization of the positive definite band matrix, A, in band symmetric storage mode as output from subroutine LFCQS/DLFCQS or LFTQS/DLFTQS. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper codiagonals of A. (Input)

DET1 — Scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that $1.0 \le |\mathrm{DET1}| < 10.0$ or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form $\det(A) = \mathrm{DET1} * 10^{\mathrm{DET2}}$.

Algorithm

Routine LFDQS computes the determinant of a real symmetric positive-definite band coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an $R^T R$ factorization. This may be done by calling either IMSL routine LFCQS, page 145, or LFTQS, page 148. The formula $\det A = \det R^T \det R = (\det R)^2$ is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det R = \prod_{i=1}^{N} R_{ii}$$

LFDQS is based on the LINPACK routine SPBDI; see Dongarra et al. (1979).

Example

The determinant is computed for a real positive definite 4 × 4 matrix with 2 codiagonals.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA, NOUT
      PARAMETER  (LDA=3, N=4, LDFAC=3, NCODA=2)
      REAL       A(LDA,N), DET1, DET2, FAC(LDFAC,N)
C
C                       Set values for A in band symmetric form
C
C                                 A = (  0.0   0.0   1.0  -2.0 )
C                                     (  0.0   2.0   1.0   3.0 )
C                                     (  7.0   6.0   6.0   8.0 )
C
      DATA A/2*0.0, 7.0, 0.0, 2.0, 6.0, 1.0, 1.0, 6.0, -2.0, 3.0, 8.0/
C                                 Factor the matrix
      CALL LFTQS (N, A, LDA, NCODA, FAC, LDFAC)
C                                 Compute the determinant
      CALL LFDQS (N, FAC, LDFAC, NCODA, DET1, DET2)
C                                 Print results
```

```
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (' The determinant of A is ',F6.3,' * 10**',F2.0)
      END
```

Output

```
The determinant of A is  1.186 * 10**3.
```

LSLTQ/DLSLTQ (Single/Double precision)

Solve a complex tridiagonal system of linear equations.

Usage

`CALL LSLTQ (N, C, D, E, B)`

Arguments

N — Order of the tridiagonal matrix. (Input)

C — Complex vector of length `N` containing the subdiagonal of the tridiagonal matrix in `C`(2) through `C`(`N`). (Input/Output)
On output `C` is destroyed.

D — Complex vector of length `N` containing the diagonal of the tridiagonal matrix. (Input/Output)
On output `D` is destroyed.

E — Complex vector of length `N` containing the superdiagonal of the tridiagonal matrix in `E`(1) through `E`(`N` – 1). (Input/Output)
On output `E` is destroyed.

B — Complex vector of length `N` containing the right-hand side of the linear system on entry and the solution vector on return. (Input/Output)

Comments

Informational error

Type	Code	
4	2	An element along the diagonal became exactly zero during execution.

Algorithm

Routine `LSLTQ` factors and solves the complex tridiagonal linear system $Ax = b$. `LSLTQ` is intended just for tridiagonal systems. The coefficient matrix does not have to be symmetric. The algorithm is Gaussian elimination with pivoting for numerical stability. See Dongarra et al. (1979), LINPACK subprograms `CGTSL`/`ZGTSL`, for details. When computing on vector or parallel computers the cyclic reduction algorithm, page 156, should be considered as an alternative method to solve the system.

Example

A system of $n = 4$ linear equations is solved.

```
C                                 Declaration of variables
      INTEGER    N
      PARAMETER  (N=4)
C
      COMPLEX    B(N), C(N), D(N), E(N)
      CHARACTER  CLABEL(1)*6, FMT*8, RLABEL(1)*4
      EXTERNAL   LSLTQ, WRCRL
C
      DATA FMT/’(E13.6)’/
      DATA CLABEL/’NUMBER’/
      DATA RLABEL/’NONE’/
C                                C(*), D(*), E(*) and B(*)
C                                contain the subdiagonal,
C                                diagonal, superdiagonal and
C                                right hand side.
      DATA C/(0.0,0.0), (-9.0,3.0), (2.0,7.0), (7.0,-4.0)/
      DATA D/(3.0,-5.0), (4.0,-9.0), (-5.0,-7.0), (-2.0,-3.0)/
      DATA E/(-9.0,8.0), (1.0,8.0), (8.0,3.0), (0.0,0.0)/
      DATA B/(-16.0,-93.0), (128.0,179.0), (-60.0,-12.0), (9.0,-108.0)/
C
C
      CALL LSLTQ (N, C, D, E, B)
C                                 Output the solution.
      CALL WRCRL (’Solution:’, 1, N, B, 1, 0, FMT, RLABEL, CLABEL)
      END
```

Output

```
Solution:
                           1                              2
(-0.400000E+01,-0.700000E+01)  (-0.700000E+01, 0.400000E+01)
                           3                              4
( 0.700000E+01,-0.700000E+01)  ( 0.900000E+01, 0.200000E+01)
```

LSLCQ/DLSLCQ (Single/Double precision)

Compute the *LDU* factorization of a complex tridiagonal matrix *A* using a cyclic reduction algorithm.

Usage

CALL LSLCQ (N, C, A, B, IJOB, Y, U, IR, IS)

Arguments

N — Order of the matrix. (Input)
N must be greater than zero.

C — Complex array of size 2N containing the upper codiagonal of the N by N tridiagonal matrix in the entries C(1), …, C(N − 1). (Input/Output)

A — Complex array of size 2N containing the diagonal of the N by N tridiagonal matrix in the entries A(1), …, A(N − 1). (Input/Output)

B — Complex array of size 2N containing the lower codiagonal of the N by N tridiagonal matrix in the entries B(1), …, B(N − 1). (Input/Output)

IJOB — Flag to direct the desired factoring or solving step. (Input)

IJOB	**Action**
1	Factor the matrix *A* and solve the system *Ax* = *y*, where *y* is stored in array *Y*.
2	Do the solve step only. Use *y* from array *Y*. (The factoring step has already been done.)
3	Factor the matrix *A* but do not solve a system.
4	Same meaning as with the value IJOB = 3. For efficiency, no error checking is done on the validity of any input value.

Y — Complex array of size 2N containing the right-hand side of the system *Ax* = *y* in the order Y(1),…,Y(N). (Input/Output)
The vector *x* overwrites *Y* in storage.

U — Real array of size 2N of flags that indicate any singularities of A. (Output)
A value U(I) = 1. means that a divide by zero would have occurred during the factoring. Otherwise U(I) = 0.

IR — Array of integers that determine the sizes of loops performed in the cyclic reduction algorithm. (Output)

IS — Array of integers that determine the sizes of loops performed in the cyclic reduction algorithm. (Output)
The sizes of these arrays must be at least $\log_2$(N) + 3.

Algorithm

Routine LSLCQ factors and solves the complex tridiagonal linear system *Ax* = *y*. The matrix is decomposed in the form *A* = *LDU*, where *L* is unit lower triangular, *U* is unit upper triangular, and *D* is diagonal. The algorithm used for the factorization is effectively that described in Kershaw (1982). More details, tests and experiments are reported in Hanson (1990).

LSLCQ is intended just for tridiagonal systems. The coefficient matrix does not have to be Hermitian. The algorithm amounts to Gaussian elimination, with no pivoting for numerical stability, on the matrix whose rows and columns are permuted to a new order. See Hanson (1990) for details. The expectation is that LSLCQ will outperform either LSLTQ, page 155, or LSLQB, page 181, on vector or parallel computers. Its performance may be inferior for small values of *n*, on scalar computers, or high-performance computers with non-optimizing compilers.

Example

A real skew-symmetric tridiagonal matrix, A, of dimension $n = 1000$ is given by $\mathrm{c}_k = -k$, $a_k = 0$, and $b_k = k$, $k = 1, \ldots, n - 1$, $a_n = 0$. This matrix will have eigenvalues that are purely imaginary. The eigenvalue closest to the imaginary unit is required. This number is obtained by using inverse iteration to approximate a complex eigenvector y. The eigenvalue is approximated by $\lambda = y^H Ay/y^H y$. (This example is contrived in the sense that the given tridiagonal skew-symmetric matrix eigenvalue problem is essentially equivalent to the tridiagonal symmetic eigenvalue problem where the $c_k = k$ and the other data are unchanged.)

```
C                                 Declare variables
      INTEGER    LP, N, N2
      PARAMETER  (LP=12, N=1000, N2=2*N)
C
      INTEGER    I, IJOB, IR(LP), IS(LP), K, NOUT
      REAL       AIMAG, U(N2)
      COMPLEX    A(N2), B(N2), C(N2), CMPLX, CONJG, S, T, Y(N2)
      INTRINSIC  AIMAG, CMPLX, CONJG
      EXTERNAL   LSLCQ, UMACH
C                                 Define entries of skew-symmetric
C                                 matrix, A:
      DO 10  I=1, N - 1
         C(I) = -I
C                                 This amounts to subtracting the
C                                 positive imaginary unit from the
C                                 diagonal.  (The eigenvalue closest
C                                 to this value is desired.)
         A(I) = CMPLX(0.E0,-1.0E0)
         B(I) = I
C                                 This initializes the approximate
C                                 eigenvector.
         Y(I) = 1.E0
   10 CONTINUE
      A(N) = CMPLX(0.E0,-1.0E0)
      Y(N) = 1.E0
C                                 First step of inverse iteration
C                                 follows. Obtain decomposition of
C                                 matrix and solve the first system:
      IJOB = 1
      CALL LSLCQ (N, C, A, B, IJOB, Y, U, IR, IS)
C
C                                 Next steps of inverse iteration
C                                 follow. Solve the system again with
C                                 the decomposition ready:
      IJOB = 2
      DO 20  K=1, 3
         CALL LSLCQ (N, C, A, B, IJOB, Y, U, IR, IS)
   20 CONTINUE
C
C                                 Compute the Raleigh quotient to
C                                 estimate the eigenvalue closest to
C                                 the positive imaginary unit. After
C                                 the approximate eigenvector, y, is
C                                 computed, the estimate of the
```

```
C                                 eigenvalue is ctrans(y)*A*y/t,
C                                 where t = ctrans(y)*y.
      S = -CONJG(Y(1))*Y(2)
      T = CONJG(Y(1))*Y(1)
      DO 30  I=2, N - 1
         S = S + CONJG(Y(I))*((I-1)*Y(I-1)-I*Y(I+1))
         T = T + CONJG(Y(I))*Y(I)
   30 CONTINUE
      S = S + CONJG(Y(N))*(N-1)*Y(N-1)
      T = T + CONJG(Y(N))*Y(N)
      S = S/T
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The value of n is:  ’, N
      WRITE (NOUT,*) ’ Value of approximate imaginary eigenvalue:’,
     &              AIMAG(S)
      STOP
      END
```

Output

```
The value of n is:    1000
Value of approximate imaginary eigenvalue:    1.03811
```

LSACB/DLSACB (Single/Double precision)

Solve a complex system of linear equations in band storage mode with iterative refinement.

Usage

CALL LSACB (N, A, LDA, NLCA, NUCA, B, IPATH, X)

Arguments

N — Number of equations. (Input)

A — Complex NLCA + NUCA + 1 by N array containing the N by N banded coefficient matrix in band storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX = B is solved.

IPATH = 2 means the system $A^H X = B$ is solved.

X — Complex vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSACB 2N(2 * NLCA + NUCA + 1) + 3N units, or
   DLSACB 4N(2 * NLCA + NUCA + 1) + 5N units.

   Workspace may be explicitly provided, if desired, by use of L2ACB/DL2ACB The reference is

   ```
   CALL L2ACB (N, A, LDA, NLCA, NUCA, B, IPATH, X, FAC,
              IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length (2 * NLCA + NUCA + 1) * N containing the *LU* factorization of A on output.

   IPVT — Integer work vector of length N containing the pivoting information for the *LU* factorization of A on output.

   WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	3	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2ACB the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSACB. Additional memory allocation for FAC and option value restoration are done automatically in LSACB. Users directly calling L2ACB can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSACB or L2ACB. Default values for the option are IVAL(*) = 1,16,0,1.

   17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSACB temporarily replaces IVAL(2) by IVAL(1). The routine L2CCB computes the condition number if IVAL(2) = 2. Otherwise L2CCB skips this computation. LSACB restores the option. Default values for the option are IVAL(*) = 1,2.

Algorithm

Routine LSACB solves a system of linear algebraic equations having a complex banded coefficient matrix. It first uses the routine LFCCB, page 164, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using the iterative refinement routine LFICB, page 172.

LSACB fails if *U*, the upper triangular part of the factorization, has a zero diagonal element or if the iterative refinement algorithm fails to converge. These errors occur only if A is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution *x*. Iterative refinement can sometimes find the solution to such a system. LSACB solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of four linear equations is solved. The coefficient matrix has complex banded form with one upper and one lower codiagonal. The right-hand-side vector *b* has four elements.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, N, NLCA, NUCA
      PARAMETER  (LDA=3, N=4, NLCA=1, NUCA=1)
      COMPLEX    A(LDA,N), B(N), X(N)
C
C                Set values for A in band form, and B
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    ( -2.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  1.0+1.0i  0.0+2.0i  0.0+0.0i )
C
C                B = ( -10.0-5.0i  9.5+5.5i  12.0-12.0i  0.0+8.0i )
C
      DATA A/(0.0,0.0), (-2.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (1.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
      DATA B/(-10.0,-5.0), (9.5,5.5), (12.0,-12.0), (0.0,8.0)/
C                                 Solve A*X = B
      IPATH = 1
      CALL LSACB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
                                  X
             1                 2                 3                 4
( 3.000, 0.000)  (-1.000, 1.000)  ( 3.000, 0.000)  (-1.000, 1.000)
```

LSLCB/DLSLCB (Single/Double precision)

Solve a complex system of linear equations in band storage mode without iterative refinement.

Usage

```
CALL LSLCB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
```

Arguments

N — Number of equations. (Input)

A — Complex NLCA + NUCA + 1 by N array containing the N by N banded coefficient matrix in band storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX = B is solved.
IPATH = 2 means the system $A^H X$ = B is solved.

X — Complex vector of length N containing the solution to the linear system. (Output)
If B is not needed, then B and X may share the same storage locations.

Comments

1. Automatic workspace usage is

   LSLCB 2N(2 * NLCA + NUCA + 1) + 3N units, or
   DLSLCB 4N(2 * NLCA + NUCA + 1) + 5N units.

   Workspace may be explicitly provided, if desired, by use of L2LCB/DL2LCB The reference is

   ```
   CALL L2LCB (N, A, LDA, NLCA, NUCA, B, IPATH, X, FAC,
               IPVT, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length (2 * NLCA + NUCA + 1) * N containing the *LU* factorization of A on output. If A is not needed, A can share the first (NLCA + NUCA + 1) * N locations with FAC.

   IPVT — Integer work vector of length N containing the pivoting information for the *LU* factorization of A on output.

WK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	3	The input matrix is too ill-conditioned. The solution might not be accurate.
4	2	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine `L2LCB` the leading dimension of `FAC` is increased by `IVAL`(3) when `N` is a multiple of `IVAL`(4). The values `IVAL`(3) and `IVAL`(4) are temporarily replaced by `IVAL`(1) and `IVAL`(2), respectively, in `LSLCB`. Additional memory allocation for `FAC` and option value restoration are done automatically in `LSLCB`. Users directly calling `L2LCB` can allocate additional space for `FAC` and set `IVAL`(3) and `IVAL`(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use `LSLCB` or `L2LCB`. Default values for the option are `IVAL(*)` = 1,16,0,1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine `LSLCB` temporarily replaces `IVAL`(2) by `IVAL`(1). The routine `L2CCB` computes the condition number if `IVAL`(2) = 2. Otherwise `L2CCB` skips this computation. `LSLCB` restores the option. Default values for the option are `IVAL(*)` = 1,2.

Algorithm

Routine `LSLCB` solves a system of linear algebraic equations having a complex banded coefficient matrix. It first uses the routine `LFCCB`, page 164, to compute an *LU* factorization of the coefficient matrix and to estimate the condition number of the matrix. The solution of the linear system is then found using `LFSCB`, page 170.

`LSLCB` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This occurs only if *A* is singular or very close to a singular matrix.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in `A` can cause very large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly scaled, it is recommended that `LSACB`, page 159, be used.

Example

A system of four linear equations is solved. The coefficient matrix has complex banded form with one upper and one lower codiagonal. The right-hand-side vector *b* has four elements.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, N, NLCA, NUCA
      PARAMETER  (LDA=3, N=4, NLCA=1, NUCA=1)
      COMPLEX    A(LDA,N), B(N), X(N)
C
C                Set values for A in band form, and B
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    ( -2.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  1.0+1.0i  0.0+2.0i  0.0+0.0i )
C
C                B = ( -10.0-5.0i  9.5+5.5i  12.0-12.0i  0.0+8.0i )
C
      DATA A/(0.0,0.0), (-2.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (1.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
      DATA B/(-10.0,-5.0), (9.5,5.5), (12.0,-12.0), (0.0,8.0)/
C                                 Solve A*X = B
      IPATH = 1
      CALL LSLCB (N, A, LDA, NLCA, NUCA, B, IPATH, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
                                  X
             1                 2                 3                 4
( 3.000, 0.000)  (-1.000, 1.000)  ( 3.000, 0.000)  (-1.000, 1.000)
```

LFCCB/DLFCCB (Single/Double precision)

Compute the *LU* factorization of a complex matrix in band storage mode and estimate its L_1 condition number.

Usage

`CALL LFCCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)`

Arguments

N — Order of the matrix. (Input)

A — Complex `NLCA` + `NUCA` + 1 by `N` array containing the `N` by `N` matrix in band storage mode to be factored. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of `A`. (Input)

NUCA — Number of upper codiagonals of `A`. (Input)

FAC — Complex 2 `*` `NLCA` + `NUCA` + 1 by `N` array containing the *LU* factorization of the matrix `A`. (Output)
If `A` is not needed, `A` can share the first (`NLCA` + `NUCA` + 1) `*` `N` locations with `FAC`
.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length `N` containing the pivoting information for the *LU* factorization. (Output)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCCB` 2`N` units, or
   `DLFCCB` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CCB`/`DL2CCB`. The reference is

   ```
   CALL L2CCB  (N, A, LDA, NLCA, NUCA, FAC, LDFAC,
                IPVT, RCOND, WK)
   ```

   The additional argument is

   WK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
4	2	The input matrix is singular.

Algorithm

Routine `LFCCB` performs an *LU* factorization of a complex banded coefficient matrix. It also estimates the condition number of the matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

LFCCB fails if U, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if A is singular or very close to a singular matrix.

The LU factors are returned in a form that is compatible with IMSL routines LFICB, page 172, LFSCB, page 170, and LFDCB, page 175. To solve systems of equations with multiple right-hand-side vectors, use LFCCB followed by either LFICB or LFSCB called once for each right-hand side. The routine LFDCB can be called to compute the determinant of the coefficient matrix after LFCCB has performed the factorization.

Let F be the matrix FAC, let m_l = NLCA and let m_u = NUCA. The first $m_l + m_u + 1$ rows of F contain the triangular matrix U in band storage form. The lower m_l rows of F contain the multipliers needed to reconstruct L^{-1}.

LFCCB is based on the LINPACK routine CGBCO; see Dongarra et al. (1979). CGBCO uses unscaled partial pivoting.

Example

The inverse of a 4 × 4 band matrix with one upper and one lower codiagonal is computed. LFCCB is called to factor the matrix and to check for singularity or ill-conditioning. LFICB is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       RCOND
      COMPLEX    A(LDA,N), AINV(N,N), FAC(LDFAC,N), RJ(N), RES(N)
C
C                Set values for A in band form
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    (  0.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  4.0+1.0i  0.0+2.0i  0.0+0.0i )
C
      DATA A/(0.0,0.0), (0.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (4.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
C
      CALL LFCCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the reciprocal condition number
C                                 and the L1 condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0),  RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                  RJ is the J-th column of the identity
C                                  matrix so the following LFICB
C                                  reference places the J-th column of
C                                  the inverse of A in the J-th column
C                                  of AINV
         CALL LFICB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RJ, IPATH,
     &               AINV(1,J), RES)
         RJ(J) = (0.0E0,0.0E0)
   10 CONTINUE
C                                  Print results
      CALL WRCRN ('AINV', N, N, AINV, N, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.022
L1 condition number = 45.933
                                 AINV
                  1                 2                 3                 4
    1  ( 0.562, 0.170)  ( 0.125, 0.260)  (-0.385,-0.135)  (-0.239,-1.165)
    2  ( 0.122, 0.421)  (-0.195, 0.094)  ( 0.101,-0.289)  ( 0.874,-0.179)
    3  ( 0.034, 0.904)  (-0.437, 0.090)  (-0.153,-0.527)  ( 1.087,-1.172)
    4  ( 0.938, 0.870)  (-0.347, 0.527)  (-0.679,-0.374)  ( 0.415,-1.759)
```

LFTCB/DLFTCB (Single/Double precision)

Compute the *LU* factorization of a complex matrix in band storage mode.

Usage

`CALL LFTCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)`

Arguments

N — Order of the matrix. (Input)

A — Complex `NLCA` + `NUCA` + 1 by `N` array containing the `N` by `N` matrix in band storage mode to be factored. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of `A`. (Input)

NUCA — Number of upper codiagonals of `A`. (Input)

FAC — Complex 2 * `NLCA` + `NUCA` + 1 by `N` array containing the *LU* factorization of the matrix `A`. (Output)
If `A` is not needed, `A` can share the first (`NLCA` + `NUCA` + 1) ∗ `N` locations with `FAC`.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Integer vector of length `N` containing the pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

   `LFTCB` 2`N` units, or
   `DLFTCB` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2TCB`/`DL2TCB` The reference is

   ```
   CALL L2TCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT,
               WK)
   ```

   The additional argument is

   WK — Complex work vector of length `N` used for scaling.

2. Informational error

Type	Code	
4	2	The input matrix is singular.

Algorithm

Routine `LFTCB` performs an *LU* factorization of a complex banded coefficient matrix. The *LU* factorization is done using scaled partial pivoting. Scaled partial pivoting differs from partial pivoting in that the pivoting strategy is the same as if each row were scaled to have the same ∞-norm.

`LFTCB` fails if *U*, the upper triangular part of the factorization, has a zero diagonal element. This can occur only if *A* is singular or very close to a singular matrix.

The *LU* factors are returned in a form that is compatible with routines `LFICB`, page 172, `LFSCB`, page 170, and `LFDCB`, page 175. To solve systems of equations with multiple right-hand-side vectors, use `LFTCB` followed by either `LFICB` or `LFSCB` called once for each right-hand side. The routine `LFDCB` can be called to compute the determinant of the coefficient matrix after `LFTCB` has performed the factorization.

Let F be the matrix `FAC`, let m_l = `NLCA` and let m_u = `NUCA`. The first $m_l + m_u + 1$ rows of F contain the triangular matrix U in band storage form. The lower m_l rows of F contain the multipliers needed to reconstruct L^{-1}. `LFTCB` is based on the LINPACK routine `CGBFA`; see Dongarra et al. (1979). `CGBFA` uses unscaled partial pivoting.

Example

A linear system with multiple right-hand sides is solved. `LFTCB` is called to factor the coefficient matrix. `LFSCB` (page 170) is called to compute the two solutions for the two right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call `LFCCB` (page 164) to perform the factorization, and `LFICB` (page 172) to compute the solutions.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      COMPLEX    A(LDA,N), B(N,2), FAC(LDFAC,N), X(N,2)
C
C                Set values for A in band form, and B
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    (  0.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  4.0+1.0i  0.0+2.0i  0.0+0.0i )
C
C                B = (  -4.0-5.0i  16.0-4.0i )
C                    (   9.5+5.5i  -9.5+19.5i )
C                    (   9.0-9.0i  12.0+12.0i )
C                    (   0.0+8.0i  -8.0-2.0i  )
C
      DATA A/(0.0,0.0), (0.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (4.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
      DATA B/(-4.0,-5.0), (9.5,5.5), (9.0,-9.0), (0.0,8.0),
     &       (16.0,-4.0), (-9.5,19.5), (12.0,12.0), (-8.0,-2.0)/
C
      CALL LFTCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                 Solve for the two right-hand sides
      DO 10  J=1, 2
         CALL LFSCB (N, FAC, LDFAC, NLCA, NUCA, IPVT, B(1,J), IPATH,
     &               X(1,J))
   10 CONTINUE
C                                 Print results
      CALL WRCRN (’X’, N, 2, X, N, 0)
C
      END
```

Output

```
                   X
                1                2
1  ( 3.000, 0.000)  ( 0.000, 4.000)
2  (-1.000, 1.000)  ( 1.000,-1.000)
3  ( 3.000, 0.000)  ( 0.000, 4.000)
4  (-1.000, 1.000)  ( 1.000,-1.000)
```

LFSCB/DLFSCB (Single/Double precision)

Solve a complex system of linear equations given the *LU* factorization of the coefficient matrix in band storage mode.

Usage

```
CALL LFSCB (N, FAC, LDFAC, NLCA, NUCA, IPVT, B, IPATH, X)
```

Arguments

N — Number of equations. (Input)

FAC — Complex 2 * NLCA + NUCA + 1 by N array containing the *LU* factorization of the coefficient matrix A as output from subroutine LFCCB/DLFCCB or LFTCB/DLFTCB. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization of A as output from subroutine LFCCB/DLFCCB or LFTCB/DLFTCB. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX = B is solved.
IPATH = 2 means the system $A^H X$ = B is solved.

X — Complex vector of length N containing the solution to the linear system. (Output)
If B is not needed, B and X can share the same storage locations.

Algorithm

Routine LFSCB computes the solution of a system of linear algebraic equations having a complex banded coefficient matrix. To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either LFCCB, page 164, or LFTCB, page 167. The solution to $Ax = b$ is found by solving the banded triangular systems $Ly = b$ and $Ux = y$. The forward elimination step consists of solving the system $Ly = b$ by applying the same permutations and elimination operations to b that were applied to the columns of A in the factorization routine. The backward substitution step consists of solving the banded triangular system $Ux = y$ for x.

LFSCB and LFICB, page 172, both solve a linear system given its *LU* factorization. LFICB generally takes more time and produces a more accurate answer than LFSCB. Each iteration of the iterative refinement algorithm used by LFICB calls LFSCB.

LFSCB is based on the LINPACK routine CGBSL; see Dongarra et al. (1979).

Example

The inverse is computed for a real banded 4 × 4 matrix with one upper and one lower codiagonal. The input matrix is assumed to be well-conditioned; hence LFTCB (page 167) is used rather than LFCCB.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA
      PARAMETER  (LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      COMPLEX    A(LDA,N), AINV(N,N), FAC(LDFAC,N), RJ(N)
C
C                Set values for A in band form
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    ( -2.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  1.0+1.0i  0.0+2.0i  0.0+0.0i )
C
      DATA A/(0.0,0.0), (-2.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (1.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
C
      CALL LFTCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSCB
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         IPATH = 1
         CALL LFSCB (N, FAC, LDFAC, NLCA, NUCA, IPVT, RJ, IPATH,
     &               AINV(1,J))
         RJ(J) = (0.0E0,0.0E0)
   10 CONTINUE
C                                 Print results
      CALL WRCRN (’AINV’, N, N, AINV, N, 0)
C
      END
```

Output

```
                              AINV
                  1                   2                   3                   4
1  ( 0.165,-0.341)  ( 0.376,-0.094)  (-0.282, 0.471)  (-1.600, 0.000)
2  ( 0.588,-0.047)  ( 0.259, 0.235)  (-0.494, 0.024)  (-0.800,-1.200)
3  ( 0.318, 0.271)  ( 0.012, 0.247)  (-0.759,-0.235)  (-0.550,-2.250)
4  ( 0.588,-0.047)  ( 0.259, 0.235)  (-0.994, 0.524)  (-2.300,-1.200)
```

LFICB/DLFICB (Single/Double precision)

Use iterative refinement to improve the solution of a complex system of linear equations in band storage mode.

Usage

```
CALL LFICB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, B,
            IPATH,X, RES)
```

Arguments

N — Number of equations. (Input)

A — Complex NLCA + NUCA + 1 by N array containing the N by N coefficient matrix in band storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of A. (Input)

NUCA — Number of upper codiagonals of A. (Input)

FAC — Complex 2 * NLCA + NUCA + 1 by N array containing the *LU* factorization of the matrix A as output from routine LFCCB/DLFCCB or LFTCB/DLFTCB. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization of A as output from routine LFCCB/DLFCCB or LFTCB/DLFTCB. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system AX = B is solved.

IPATH = 2 means the system $A^H X = B$ is solved.

X — Complex vector of length N containing the solution. (Output)

RES — Complex vector of length N containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	3	The input matrix is too ill-conditioned for iterative refinement be effective.

Algorithm

Routine `LFICB` computes the solution of a system of linear algebraic equations having a complex banded coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either `LFCCB`, page 164, or `LFTCB`, page 167.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFICB` and `LFSCB`, page 170, both solve a linear system given its *LU* factorization. `LFICB` generally takes more time and produces a more accurate answer than `LFSCB`. Each iteration of the iterative refinement algorithm used by `LFICB` calls `LFSCB`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding (1 + *i*)/2 to the second element.

```
C                                 Declare variables
      INTEGER    IPATH, LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (IPATH=1, LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       RCOND
      COMPLEX    A(LDA,N), B(N), FAC(LDFAC,N), RES(N), X(N)
C
C                Set values for A in band form, and B
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    ( -2.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  1.0+1.0i  0.0+2.0i  0.0+0.0i )
C
C                B = ( -10.0-5.0i  9.5+5.5i  12.0-12.0i  0.0+8.0i )
C
      DATA A/(0.0,0.0), (-2.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (1.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
      DATA B/(-10.0,-5.0), (9.5,5.5), (12.0,-12.0), (0.0,8.0)/
C
      CALL LFCCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, RCOND)
C                                 Print the reciprocal condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998) RCOND, 1.0E0/RCOND
```

```
C                                 Solve the three systems
      DO 10  J=1, 3
         CALL LFICB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT, B,
     &               IPATH, X, RES)
C                                 Print results
         WRITE (NOUT, 99999) J
         CALL WRCRN ('X', 1, N, X, 1, 0)
         CALL WRCRN ('RES', 1, N, RES, 1, 0)
C                                 Perturb B by adding 0.5+0.5i to B(2)
         B(2) = B(2) + (0.5E0,0.5E0)
   10 CONTINUE
C
99998 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
99999 FORMAT (//,'  For system ',I1)
      END
```

Output

```
RCOND = 0.014
L1 Condition number = 72.414

For system 1
                                  X
            1                 2                 3                 4
( 3.000, 0.000)  (-1.000, 1.000)  ( 3.000, 0.000)  (-1.000, 1.000)

                                    RES
                      1                         2                         3
( 0.000E+00, 0.000E+00)  ( 0.000E+00, 0.000E+00)  ( 0.000E+00, 5.684E-14)
                      4
( 3.494E-22,-6.698E-22)

For system 2
                                  X
              1                 2                 3                 4
( 3.235, 0.141)  (-0.988, 1.247)  ( 2.882, 0.129)  (-0.988, 1.247)

                                    RES
                      1                         2                         3
(-1.402E-08, 6.486E-09)  (-7.012E-10, 4.488E-08)  (-1.122E-07, 7.188E-09)
                      4
(-7.012E-10, 4.488E-08)

For system 3
                                  X
              1                 2                 3                 4
( 3.471, 0.282)  (-0.976, 1.494)  ( 2.765, 0.259)  (-0.976, 1.494)

                                    RES
                      1                         2                         3
(-2.805E-08, 1.297E-08)  (-1.402E-09,-2.945E-08)  ( 1.402E-08, 1.438E-08)
                      4
(-1.402E-09,-2.945E-08)
```

LFDCB/DLFDCB (Single/Double precision)

Compute the determinant of a complex matrix given the *LU* factorization of the matrix in band storage mode.

Usage

```
CALL LFDCB (N, FAC, LDFAC, NLCA, NUCA, IPVT, DET1, DET2)
```

Arguments

N — Order of the matrix. (Input)

FAC — Complex (2 * NLCA + NUCA + 1) by N array containing the *LU* factorization of the matrix A as output from routine LFTCB/DLFTCB or LFCCB/DLFCCB. (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals in matrix A. (Input)

NUCA — Number of upper codiagonals in matrix A. (Input)

IPVT — Vector of length N containing the pivoting information for the *LU* factorization as output from routine LFTCB/DLFTCB or LFCCB/DLFCCB. (Input)

DET1 — Complex scalar containing the mantissa of the determinant. (Output)
The value DET1 is normalized so that 1.0 ≤ |DET1 | < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant. (Output)
The determinant is returned in the form det (*A*) = DET1 * 10^{DET2}.

Algorithm

Routine LFDCB computes the determinant of a complex banded coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an *LU* factorization. This may be done by calling either LFCCB, page 164, or LFTCB, page 167. The formula det *A* = det *L* det *U* is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det U = \prod_{i=1}^{N} U_{ii}$$

(The matrix *U* is stored in the upper NUCA + NLCA + 1 rows of FAC as a banded matrix.) Since *L* is the product of triangular matrices with unit diagonals and of permutation matrices, det *L* = $(-1)^k$, where *k* is the number of pivoting interchanges.

LFDCB is based on the LINPACK routine CGBDI; see Dongarra et al. (1979).

Example

The determinant is computed for a complex banded 4 × 4 matrix with one upper and one lower codiagonal.

```
C                                  Declare variables
      INTEGER    LDA, LDFAC, N, NLCA, NUCA, NOUT
      PARAMETER  (LDA=3, LDFAC=4, N=4, NLCA=1, NUCA=1)
      INTEGER    IPVT(N)
      REAL       DET2
      COMPLEX    A(LDA,N), DET1, FAC(LDFAC,N)
C
C                Set values for A in band form
C
C                A = (  0.0+0.0i  4.0+0.0i -2.0+2.0i -4.0-1.0i )
C                    ( -2.0-3.0i -0.5+3.0i  3.0-3.0i  1.0-1.0i )
C                    (  6.0+1.0i  1.0+1.0i  0.0+2.0i  0.0+0.0i )
C
      DATA A/(0.0,0.0), (-2.0,-3.0), (6.0,1.0), (4.0,0.0), (-0.5,3.0),
     &       (1.0,1.0), (-2.0,2.0), (3.0,-3.0), (0.0,2.0), (-4.0,-1.0),
     &       (1.0,-1.0), (0.0,0.0)/
C
      CALL LFTCB (N, A, LDA, NLCA, NUCA, FAC, LDFAC, IPVT)
C                                  Compute the determinant
      CALL LFDCB (N, FAC, LDFAC, NLCA, NUCA, IPVT, DET1, DET2)
C                                  Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (' The determinant of A is (', F6.3, ',', F6.3, ') * 10**',
     &        F2.0)
      END
```

Output

```
The determinant of A is ( 2.500,-1.500) * 10**1.
```

LSAQH/DLSAQH (Single/Double precision)

Solve a complex Hermitian positive definite system of linear equations in band Hermitian storage mode with iterative refinement.

Usage

`CALL LSAQH (N, A, LDA, NCODA, B, X)`

Arguments

N — Number of equations. (Input)

A — Complex `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band Hermitian coefficient matrix in band Hermitian storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA – Number of upper or lower codiagonals of A. (Input)

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

X — Complex vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSAQH 2N(NCODA + 2) units, or
   DLSAQH 4N(NCODA + 2) units.

   Workspace may be explicitly provided, if desired, by use of L2AQH/DL2AQH The reference is

   ```
   CALL L2AQH (N, A, LDA, NCODA, B, X, FAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Complex work vector of length (NCODA + 1) * N containing the $R^H R$ factorization of A in band Hermitian storage form on output.

   WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	3	The input matrix is too ill-conditioned. The solution might not be accurate.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2AQH the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSAQH. Additional memory allocation for FAC and option value restoration are done automatically in LSAQH. Users directly calling L2AQH can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSAQH or L2AQH. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSAQH temporarily replaces IVAL(2) by IVAL(1). The routine L2CQH computes the condition number if IVAL(2) = 2. Otherwise L2CQH skips this computation. LSAQH restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSAQH solves a system of linear algebraic equations having a complex Hermitian positive definite band coefficient matrix. It first uses the IMSL routine LFCQH, page 184, to compute an $R^H R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. R is an upper triangular band matrix. The solution of the linear system is then found using the iterative refinement IMSL routine LFIQH, page 191.

LSAQH fails if any submatrix of R is not positive definite, if R has a zero diagonal element, or if the iterative refinement agorithm fails to converge. These errors occur only if the matrix A either is very close to a singular matrix or is a matrix that is not positive definite.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system. LSAQH solves the problem that is represented in the computer; however, this problem may differ from the problem whose solution is desired.

Example

A system of five linear equations is solved. The coefficient matrix has complex Hermitian positive definite band form with one codiagonal and the right-hand-side vector b has five elements.

```
C                                 Declare variables
      INTEGER    LDA, N, NCODA
      PARAMETER  (LDA=2, N=5, NCODA=1)
      COMPLEX    A(LDA,N), B(N), X(N)
C
C            Set values for A in band Hermitian form, and B
C
C            A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
C            B = ( 1.0+5.0i 12.0-6.0i  1.0-16.0i -3.0-3.0i 25.0+16.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B/(1.0,5.0), (12.0,-6.0), (1.0,-16.0), (-3.0,-3.0),
     &       (25.0,16.0)/
C                                 Solve A*X = B
      CALL LSAQH (N, A, LDA, NCODA, B, X)
```

```
C                                  Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
C
      END
```

Output

```
                                    X
              1                  2                  3                  4
( 2.000, 1.000)  ( 3.000, 0.000)  (-1.000,-1.000)  ( 0.000,-2.000)
              5
( 3.000, 2.000)
```

LSLQH/DLSLQH (Single/Double precision)

Solve a complex Hermitian positive definite system of linear equations in band Hermitian storage mode without iterative refinement.

Usage

`CALL LSLQH (N, A, LDA, NCODA, B, X)`

Arguments

N — Number of equations. (Input)

A — Complex `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band Hermitian coefficient matrix in band Hermitian storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper or lower codiagonals of `A`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

X — Complex vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSLQH` 2`N`(`NCODA` + 2) units, or
   `DLSLQH` 4`N`(`NCODA` + 2) units.

   Workspace may be explicitly provided, if desired, by use of `L2LQH`/`DL2LQH` The reference is

   `CALL L2LQH (N, A, LDA, NCODA, B, X, FAC, WK)`

   The additional arguments are as follows:

FAC — Complex work vector of length (NCODA + 1) * N containing the $R^H R$ factorization of A in band Hermitian storage form on output. If A is not needed, A and FAC can share the same storage locations.

WK — Complex work vector of length N.

2. Informational errors

Type	Code	
3	3	The input matrix is too ill-conditioned. The solution might not be accurate.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2LQH the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSLQH. Additional memory allocation for FAC and option value restoration are done automatically in LSLQH. Users directly calling L2LQH can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSLQH or L2LQH. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSLQH temporarily replaces IVAL(2) by IVAL(1). The routine L2CQH computes the condition number if IVAL(2) = 2. Otherwise L2CQH skips this computation. LSLQH restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSLQH solves a system of linear algebraic equations having a complex Hermitian positive definite band coefficient matrix. It first uses the routine LFCQH, page 184, to compute an $R^H R$ Cholesky factorization of the coefficient matrix and to estimate the condition number of the matrix. *R* is an upper triangular band matrix. The solution of the linear system is then found using the routine LFSQH, page 189.

`LSLQH` fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* either is very close to a singular matrix or is a matrix that is not positive definite.

If the estimated condition number is greater than 1/ε (where ε is machine precision), a warning error is issued. This indicates that very small changes in `A` can cause very large changes in the solution *x*. If the coefficient matrix is ill-conditioned or poorly sealed, it is recommended that `LSAQH`, page 176, be used.

Example

A system of five linear equations is solved. The coefficient matrix has complex Hermitian positive definite band form with one codiagonal and the right-hand-side vector *b* has five elements.

```
C                                  Declare variables
      INTEGER    N, NCODA, LDA
      PARAMETER  (N=5, NCODA=1, LDA=NCODA+1)
      COMPLEX    A(LDA,N), B(N), X(N)
C
C            Set values for A in band Hermitian form, and B
C
C            A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
C            B = ( 1.0+5.0i 12.0-6.0i  1.0-16.0i -3.0-3.0i 25.0+16.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B/(1.0,5.0), (12.0,-6.0), (1.0,-16.0), (-3.0,-3.0),
     &       (25.0,16.0)/
C                                  Solve A*X = B
      CALL LSLQH (N, A, LDA, NCODA, B, X)
C                                  Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
C
      END
```

Output

```
                                  X
              1                 2                 3                 4
( 2.000, 1.000)  ( 3.000, 0.000)  (-1.000,-1.000)  ( 0.000,-2.000)

              5
( 3.000, 2.000)
```

LSLQB/DLSLQB (Single/Double precision)

Compute the $R^H DR$ Cholesky factorization of a complex Hermitian positive-definite matrix A in codiagonal band Hermitian storage mode. Solve a system $Ax = b$.

Usage

```
CALL LSLQB (N, A, LDA, NCODA, IJOB, U)
```

Arguments

N — Order of the matrix. (Input)
Must satisfy `N` > 0.

A — Array containing the `N` by `N` positive-definite band coefficient matrix and the right hand side in codiagonal band Hermitian storage mode. (Input/Output)
The number of array columns must be at least 2 * `NCODA` + 3. The number of columns is not an input to this subprogram.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)
Must satisfy `LDA` ≥ `N` + `NCODA`.

NCODA — Number of upper codiagonals of matrix `A`. (Input)
Must satisfy `NCODA` ≥ 0 and `NCODA` < `N`.

IJOB — flag to direct the desired factorization or solving step. (Input)

`IJOB`	**Meaning**
1	factor the matrix `A` and solve the system `A`x = b; where the real part of b is stored in column 2 * `NCODA` + 2 and the imaginary part of b is stored in column 2 * `NCODA` + 3 of array `A`. The real and imaginary parts of b are overwritten by the real and imaginary parts of x.
2	solve step only. Use the real part of b as column 2 * `NCODA` + 2 and the imaginary part of b as column 2 * `NCODA` + 3 of `A`. (The factorization step has already been done.) The real and imaginary parts of b are overwritten by the real and imaginary parts of x.
3	factor the matrix `A` but do not solve a system.
4,5,6	same meaning as with the value `IJOB` = 3. For efficiency, no error checking is done on values `LDA`, `N`, `NCODA`, and `U(*)`.

U — Array of flags that indicate any singularities of `A`, namely loss of positive-definiteness of a leading minor. (Output)
A value `U(I)` = 0. means that the leading minor of dimension `I` is not positive-definite. Otherwise, `U(I)` = 1.

Comments

1. Automatic workspace usage is

   `LSLQB` 2 * `NCODA` units, or
   `DLSLQB` 4 * `NCODA` units.

   Workspace may be explicitly provided, if desired, by use of `L2LQB`/`DL2LQB` The reference is

   `CALL L2LQB (N, A, LDA, NCODA, IJOB, U, WK1, WK2)`

   The additional arguments are as follows:

   WK1 — Work vector of length `NCODA`.

   WK2 — Work vector of length `NCODA`.

2. Informational error

Type	Code	
4	2	The input matrix is not positive definite.

Algorithm

Routine `LSLQB` factors and solves the Hermitian positive definite banded linear system $Ax = b$. The matrix is factored so that $A = R^H DR$, where R is unit upper triangular and D is diagonal and real. The reciprocals of the diagonal entries of D are computed and saved to make the solving step more efficient. Errors will occur if D has a nonpositive diagonal element. Such events occur only if A is very close to a singular matrix or is not positive definite.

`LSLQB` is efficient for problems with a small band width. The particular cases `NCODA` = 0, 1 are done with special loops within the code. These cases will give good performance. See Hanson (1989) for more on the algorithm. When solving tridiagonal systems, `NCODA` = 1, the cyclic reduction code `LSLCQ` (page 156) should be considered as an alternative. The expectation is that `LSLCQ` will outperform `LSLQB` on vector or parallel computers. It may be inferior on scalar computers or even parallel computers with non-optimizing compilers.

Example

A system of five linear equations is solved. The coefficient matrix has real positive definite codiagonal Hermitian band form and the right-hand-side vector b has five elements.

```
      INTEGER    LDA, N, NCODA
      PARAMETER  (N=5, NCODA=1, LDA=N+NCODA)
C
      INTEGER    I, IJOB, J
      REAL       A(LDA,2*NCODA+3), U(N)
      EXTERNAL   LSLQB, WRRRN
C
C                                 Set values for A and right hand side
C                                 in codiagonal band Hermitian form:
C
```

```
C                            (   *      *      *      *      * )
C                            ( 2.0      *      *    1.0   5.0)
C                  A   =     ( 4.0  -1.0   1.0  12.0  -6.0)
C                            (10.0   1.0   2.0   1.0 -16.0)
C                            ( 6.0   0.0   4.0  -3.0  -3.0)
C                            ( 9.0   1.0   1.0  25.0  16.0)
C
      DATA ((A(I+NCODA,J),I=1,N),J=1,2*NCODA+3)/2.0, 4.0, 10.0, 6.0,
     &     9.0, 0.0, -1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 2.0, 4.0, 1.0,
     &     1.0, 12.0, 1.0, -3.0, 25.0, 5.0, -6.0, -16.0, -3.0, 16.0/
C
C                                  Factor and solve A*x = b.
C
      IJOB = 1
      CALL LSLQB (N, A, LDA, NCODA, IJOB, U)
C
C                                  Print results
C
      CALL WRRRN ('REAL(X)', 1, N, A(NCODA+1,2*NCODA+2), 1, 0)
      CALL WRRRN ('IMAG(X)', 1, N, A(NCODA+1,2*NCODA+3), 1, 0)
      END
```

Output

```
                  REAL(X)
    1       2        3       4       5
2.000   3.000  -1.000   0.000   3.000

                 IMAG(X)
    1       2        3       4       5
1.000   0.000  -1.000  -2.000   2.000
```

LFCQH/DLFCQH (Single/Double precision)

Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode and estimate its L_1 condition number.

Usage

```
CALL LFCQH (N, A, LDA, NCODA, FAC, LDFAC, RCOND)
```

Arguments

N — Order of the matrix.   (Input)

A — Complex `NCODA` + 1 by `N` array containing the `N` by `N` positive definite band Hermitian matrix to be factored in band Hermitian storage mode.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NCODA — Number of upper or lower codiagonals of `A`.   (Input)

FAC — Complex `NCODA` + 1 by `N` array containing the $R^H R$ factorization of the matrix `A`. (Output)
If `A` is not needed, `A` and `FAC` can share the same storage locations.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

RCOND — Scalar containing an estimate of the reciprocal of the L_1 condition number of `A`. (Output)

Comments

1. Automatic workspace usage is

   `LFCQH` 2`N` units, or
   `DLFCQH` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `L2CQH`/`DL2CQH`. The reference is

   ```
   CALL L2CQH (N, A, LDA, NCODA, FAC, LDFAC, RCOND, WK)
   ```

   The additional argument is

   WK — Complex work vector of length `N`.

2. Informational errors

Type	Code	
3	1	The input matrix is algorithmically singular.
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.
4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `LFCQH` computes an $R^H R$ Cholesky factorization and estimates the condition number of a complex Hermitian positive definite band coefficient matrix. R is an upper triangular band matrix.

The L_1 condition number of the matrix A is defined to be $\kappa(A) = \|A\|_1 \|A^{-1}\|_1$. Since it is expensive to compute $\|A^{-1}\|_1$, the condition number is only estimated. The estimation algorithm is the same as used by LINPACK and is described by Cline et al. (1979).

If the estimated condition number is greater than $1/\varepsilon$ (where ε is machine precision), a warning error is issued. This indicates that very small changes in A can cause very large changes in the solution x. Iterative refinement can sometimes find the solution to such a system.

LFCQH fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* either is very close to a singular matrix or is a matrix which is not positive definite.

The $R^H R$ factors are returned in a form that is compatible with routines LFIQH, page 191, LFSQH, page 189, and LFDQH, page 193. To solve systems of equations with multiple right-hand-side vectors, use LFCQH followed by either LFIQH or LFSQH called once for each right-hand side. The routine LFDQH can be called to compute the determinant of the coefficient matrix after LFCQH has performed the factorization.

LFCQH is based on the LINPACK routine CPBCO; see Dongarra et al. (1979).

Example

The inverse of a 5 × 5 band Hermitian matrix with one codiagonal is computed. LFCQH is called to factor the matrix and to check for nonpositive definiteness or ill-conditioning. LFIQH (page 191) is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    N, NCODA, LDA, LDFAC, NOUT
      PARAMETER  (N=5, NCODA=1, LDA=NCODA+1, LDFAC=LDA)
      REAL       RCOND
      COMPLEX    A(LDA,N), AINV(N,N), FAC(LDFAC,N), RES(N), RJ(N)
C
C            Set values for A in band Hermitian form
C
C            A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
C                                 Factor the matrix A
      CALL LFCQH (N, A, LDA, NCODA, FAC, LDFAC, RCOND)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFIQH
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFIQH (N, A, LDA, NCODA, FAC, LDFAC, RJ, AINV(1,J), RES)
         RJ(J) = (0.0E0,0.0E0)
   10 CONTINUE
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
      CALL WRCRN ('AINV', N, N, AINV, N, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.067
L1 Condition number = 14.961

                                    AINV
                  1                   2                   3                  4
1 ( 0.7166, 0.0000)  ( 0.2166,-0.2166)  (-0.0899,-0.0300) (-0.0207, 0.0622)
2 ( 0.2166, 0.2166)  ( 0.4332, 0.0000)  (-0.0599,-0.1198) (-0.0829, 0.0415)
3 (-0.0899, 0.0300)  (-0.0599, 0.1198)  ( 0.1797, 0.0000) ( 0.0000,-0.1244)
4 (-0.0207,-0.0622)  (-0.0829,-0.0415)  ( 0.0000, 0.1244) ( 0.2592, 0.0000)
5 ( 0.0092, 0.0046)  ( 0.0138,-0.0046)  (-0.0138,-0.0138) (-0.0288, 0.0288)
                   5
1  ( 0.0092,-0.0046)
2  ( 0.0138, 0.0046)
3  (-0.0138, 0.0138)
4  (-0.0288,-0.0288)
5  ( 0.1175, 0.0000)
```

LFTQH/DLFTQH (Single/Double precision)

Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode.

Usage

```
CALL LFTQH (N, A, LDA, NCODA, FAC, LDFAC)
```

Arguments

N — Order of the matrix. (Input)

A — Complex NCODA + 1 by N array containing the N by N positive definite band Hermitian matrix to be factored in band Hermitian storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper or lower codiagonals of A. (Input)

FAC — Complex NCODA + 1 by N array containing the $R^H R$ factorization of the matrix A. (Output)
If A is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

Informational errors

Type	Code	
3	4	The input matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The input matrix is not positive definite.

4	4	The input matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `LFTQH` computes an $R^H R$ Cholesky factorization of a complex Hermitian positive definite band coefficient matrix. *R* is an upper triangular band matrix.

`LFTQH` fails if any submatrix of *R* is not positive definite or if *R* has a zero diagonal element. These errors occur only if *A* either is very close to a singular matrix or is a matrix which is not positive definite.

The $R^H R$ factors are returned in a form that is compatible with routines `LFIQH`, page 191, `LFSQH`, page 189, and `LFDQH`, page 193. To solve systems of equations with multiple right-hand-side vectors, use `LFTQH` followed by either `LFIQH` or `LFSQH` called once for each right-hand side. The routine `LFDQH` can be called to compute the determinant of the coefficient matrix after `LFTQH` has performed the factorization.

`LFTQH` is based on the LINPACK routine `SPBFA`; see Dongarra et al. (1979).

Example

The inverse of a 5 × 5 band Hermitian matrix with one codiagonal is computed. `LFTQH` is called to factor the matrix and to check for nonpositive definiteness. `LFSQH` is called to determine the columns of the inverse.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA
      PARAMETER  (LDA=2, LDFAC=2, N=5, NCODA=1)
      COMPLEX    A(LDA,N), AINV(N,N), FAC(LDFAC,N), RJ(N)
C
C            Set values for A in band Hermitian form
C
C            A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
C                                 Factor the matrix A
      CALL LFTQH (N, A, LDA, NCODA, FAC, LDFAC)
C                                 Set up the columns of the identity
C                                 matrix one at a time in RJ
      CALL CSET (N, (0.0E0,0.0E0), RJ, 1)
      DO 10  J=1, N
         RJ(J) = (1.0E0,0.0E0)
C                                 RJ is the J-th column of the identity
C                                 matrix so the following LFSQH
C                                 reference places the J-th column of
C                                 the inverse of A in the J-th column
C                                 of AINV
         CALL LFSQH (N, FAC, LDFAC, NCODA, RJ, AINV(1,J))
         RJ(J) = (0.0E0,0.0E0)
```

```
   10 CONTINUE
C                                  Print the results
      CALL WRCRN ('AINV', N, N, AINV, N, 0)
C
      END
```

Output

```
                                          AINV
                  1                   2                   3                   4
1 ( 0.7166, 0.0000) ( 0.2166,-0.2166)  (-0.0899,-0.0300)  (-0.0207, 0.0622)
2 ( 0.2166, 0.2166) ( 0.4332, 0.0000)  (-0.0599,-0.1198)  (-0.0829, 0.0415)
3 (-0.0899, 0.0300) (-0.0599, 0.1198)  ( 0.1797, 0.0000)  ( 0.0000,-0.1244)
4 (-0.0207,-0.0622) (-0.0829,-0.0415)  ( 0.0000, 0.1244)  ( 0.2592, 0.0000)
5 ( 0.0092, 0.0046) ( 0.0138,-0.0046)  (-0.0138,-0.0138)  (-0.0288, 0.0288)
                   5
1  ( 0.0092,-0.0046)
2  ( 0.0138, 0.0046)
3  (-0.0138, 0.0138)
4  (-0.0288,-0.0288)
5  ( 0.1175, 0.0000)
```

LFSQH/DLFSQH (Single/Double precision)

Solve a complex Hermitian positive definite system of linear equations given the factorization of the coefficient matrix in band Hermitian storage mode.

Usage

CALL LFSQH (N, FAC, LDFAC, NCODA, B, X)

Arguments

N — Number of equations.   (Input)

FAC — Complex NCODA + 1 by N array containing the $R^H R$ factorization of the Hermitian positive definite band matrix A.   (Input)
FAC is obtained as output from routine LFCQH/DLFCQH or LFTQH/DLFTQH .

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

NCODA — Number of upper or lower codiagonals of A.   (Input)

B — Complex vector of length N containing the right-hand-side of the linear system.   (Input)

X — Complex vector of length N containing the solution to the linear system.   (Output)
If B is not needed, B and X can share the same storage locations.

Comments

Informational error

Type	Code	
4	1	The factored matrix has a diagonal element close to zero.

Algorithm

This routine computes the solution for a system of linear algebraic equations having a complex Hermitian positive definite band coefficient matrix. To compute the solution, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either IMSL routine `LFCQH`, page 184, or `LFTQH`, page 187. R is an upper triangular band matrix.

The solution to $Ax = b$ is found by solving the triangular systems $R^H y = b$ and $Rx = y$.

`LFSQH` and `LFIQH`, page 191, both solve a linear system given its $R^H R$ factorization. `LFIQH` generally takes more time and produces a more accurate answer than `LFSQH`. Each iteration of the iterative refinement algorithm used by `LFIQH` calls `LFSQH`.

`LFSQH` is based on the LINPACK routine `CPBSL`; see Dongarra et al. (1979).

Example

A set of linear systems is solved successively. `LFTQH`, page 187, is called to factor the coefficient matrix. `LFSQH` is called to compute the three solutions for the three right-hand sides. In this case the coefficient matrix is assumed to be well-conditioned and correctly scaled. Otherwise, it would be better to call `LFCQH`, page 184, to perform the factorization, and `LFIQH`, page 191, to compute the solutions.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA
      PARAMETER  (LDA=2, LDFAC=2, N=5, NCODA=1)
      COMPLEX    A(LDA,N), B(N,3), FAC(LDFAC,N), X(N,3)
C
C             Set values for A in band Hermitian form, and B
C
C             A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                 ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
C             B = (  3.0+3.0i   4.0+0.0i   29.0-9.0i  )
C                 (  5.0-5.0i  15.0-10.0i -36.0-17.0i )
C                 (  5.0+4.0i -12.0-56.0i -15.0-24.0i )
C                 (  9.0+7.0i -12.0+10.0i -23.0-15.0i )
C                 (-22.0+1.0i   3.0-1.0i  -23.0-28.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B/(3.0,3.0), (5.0,-5.0), (5.0,4.0), (9.0,7.0), (-22.0,1.0),
     &       (4.0,0.0), (15.0,-10.0), (-12.0,-56.0), (-12.0,10.0),
     &       (3.0,-1.0), (29.0,-9.0), (-36.0,-17.0), (-15.0,-24.0),
     &       (-23.0,-15.0), (-23.0,-28.0)/
C                                  Factor the matrix A
      CALL LFTQH (N, A, LDA, NCODA, FAC, LDFAC)
C                                  Compute the solutions
      DO 10  I=1, 3
         CALL LFSQH (N, FAC, LDFAC, NCODA, B(1,I), X(1,I))
   10 CONTINUE
C                                  Print solutions
      CALL WRCRN ('X', N, 3, X, N, 0)
      END
```

Output

```
                          X
                 1                 2                 3
1  (  1.00,  0.00)  (  3.00, -1.00)  ( 11.00, -1.00)
2  (  1.00, -2.00)  (  2.00,  0.00)  ( -7.00,  0.00)
3  (  2.00,  0.00)  ( -1.00, -6.00)  ( -2.00, -3.00)
4  (  2.00,  3.00)  (  2.00,  1.00)  ( -2.00, -3.00)
5  ( -3.00,  0.00)  (  0.00,  0.00)  ( -2.00, -3.00)
```

LFIQH/DLFIQH (Single/Double precision)

Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations in band Hermitian storage mode.

Usage

CALL LFIQH (N, A, LDA, NCODA, FAC, LDFAC, B, X, RES)

Arguments

N — Number of equations.   (Input)

A — Complex NCODA + 1 by N array containing the N by N positive definite band Hermitian coefficient matrix in band Hermitian storage mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NCODA — Number of upper or lower codiagonals of A.   (Input)

FAC — Complex NCODA + 1 by N array containing the $R^H R$ factorization of the matrix A as output from routine LFCQH/DLFCQH or LFTQH/DLFTQH.   (Input)

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program.   (Input)

B — Complex vector of length N containing the right-hand side of the linear system.   (Input)

X — Complex vector of length N containing the solution to the linear system. (Output)

RES — Complex vector of length `N` containing the residual vector at the improved solution. (Output)

Comments

Informational error

Type	Code	
3	3	The input matrix is too ill-conditioned for iterative refinement to be effective.

Algorithm

Routine `LFIQH` computes the solution of a system of linear algebraic equations having a complex Hermitian positive definite band coefficient matrix. Iterative refinement is performed on the solution vector to improve the accuracy. Usually almost all of the digits in the solution are accurate, even if the matrix is somewhat ill-conditioned.

To compute the solution, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either `LFCQH`, page 184, or `LFTQH`, page 187.

Iterative refinement fails only if the matrix is very ill-conditioned.

`LFIQH` and `LFSQH`, page 189, both solve a linear system given its $R^H R$ factorization. `LFIQH` generally takes more time and produces a more accurate answer than `LFSQH`. Each iteration of the iterative refinement algorithm used by `LFIQH` calls `LFSQH`.

Example

A set of linear systems is solved successively. The right-hand-side vector is perturbed after solving the system each of the first two times by adding (1 + *i*)/2 to the second element.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA, NOUT
      PARAMETER  (LDA=2, LDFAC=2, N=5, NCODA=1)
      REAL       RCOND
      COMPLEX    A(LDA,N), B(N), FAC(LDFAC,N), RES(N,3), X(N,3)
C
C            Set values for A in band Hermitian form, and B
C
C            A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
C            B = ( 3.0+3.0i  5.0-5.0i  5.0+4.0i  9.0+7.0i -22.0+1.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
      DATA B/(3.0,3.0), (5.0,-5.0), (5.0,4.0), (9.0,7.0), (-22.0,1.0)/
C                                 Factor the matrix A
      CALL LFCQH (N, A, LDA, NCODA, FAC, LDFAC, RCOND)
```

```
C                                  Print the estimated condition number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) RCOND, 1.0E0/RCOND
C                                  Compute the solutions
      DO 10  I=1, 3
         CALL LFIQH (N, A, LDA, NCODA, FAC, LDFAC, B, X(1,I), RES(1,I))
         B(2) = B(2) + (0.5E0,0.5E0)
   10 CONTINUE
C                                  Print solutions and residuals
      CALL WRCRN ('X', N, 3, X, N, 0)
      CALL WRCRN ('RES', N, 3, RES, N, 0)
C
99999 FORMAT ('  RCOND = ',F5.3,/,'  L1 Condition number = ',F6.3)
      END
```

Output

```
RCOND = 0.067
L1 Condition number = 14.961

                           X
                 1                 2                 3
1  ( 1.000, 0.000)  ( 1.217, 0.000)  ( 1.433, 0.000)
2  ( 1.000,-2.000)  ( 1.217,-1.783)  ( 1.433,-1.567)
3  ( 2.000, 0.000)  ( 1.910, 0.030)  ( 1.820, 0.060)
4  ( 2.000, 3.000)  ( 1.979, 2.938)  ( 1.959, 2.876)
5  (-3.000, 0.000)  (-2.991, 0.005)  (-2.982, 0.009)

                                     RES
                        1                         2                         3
1 ( 1.192E-07, 0.000E+00)  ( 6.592E-08, 1.686E-07)  ( 1.318E-07, 2.010E-14)
2 ( 1.192E-07,-2.384E-07)  (-5.329E-08,-5.329E-08)  ( 1.318E-07,-2.258E-07)
3 ( 2.384E-07, 8.259E-08)  ( 2.390E-07,-3.309E-08)  ( 2.395E-07, 1.015E-07)
4 (-2.384E-07, 2.814E-14)  (-8.240E-08,-8.790E-09)  (-1.648E-07,-1.758E-08)
5 (-2.384E-07,-1.401E-08)  (-2.813E-07, 6.981E-09)  (-3.241E-07,-2.795E-08)
```

LFDQH/DLFDQH (Single/Double precision)

Compute the determinant of a complex Hermitian positive definite matrix given the $R^T R$ Cholesky factorization in band Hermitian storage mode.

Usage

`CALL LFDQH (N, FAC, LDFAC, NCODA, DET1, DET2)`

Arguments

N — Number of equations. (Input)

FAC — Complex `NCODA` + 1 by `N` array containing the $R^H R$ factorization of the Hermitian positive definite band matrix `A`. (Input)
`FAC` is obtained as output from routine `LFCQH`/`DLFCQH` or `LFTQH`/`DLFTQH`.

LDFAC — Leading dimension of `FAC` exactly as specified in the dimension statement of the calling program. (Input)

NCODA — Number of upper or lower codiagonals of A.   (Input)

DET1 — Scalar containing the mantissa of the determinant.   (Output)
The value DET1 is normalized so that 1.0 ≤ |DET1| < 10.0 or DET1 = 0.0.

DET2 — Scalar containing the exponent of the determinant.   (Output)
The determinant is returned in the form det (*A*) = DET1 * 10^{DET2}.

Algorithm

Routine LFDQH computes the determinant of a complex Hermitian positive definite band coefficient matrix. To compute the determinant, the coefficient matrix must first undergo an $R^H R$ factorization. This may be done by calling either LFCQH, page 184, or LFTQH, page 187. The formula det A = det R^H det R = $(\det R)^2$ is used to compute the determinant. Since the determinant of a triangular matrix is the product of the diagonal elements,

$$\det R = \prod_{i=1}^{N} R_{ii}$$

LFDQH is based on the LINPACK routine CPBDI; see Dongarra et al. (1979).

Example

The determinant is computed for a 5 × 5 complex Hermitian positive definite band matrix with one codiagonal.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N, NCODA, NOUT
      PARAMETER  (LDA=2, N=5, LDFAC=2, NCODA=1)
      REAL       DET1, DET2
      COMPLEX    A(LDA,N), FAC(LDFAC,N)
C
C             Set values for A in band Hermitian form
C
C             A = ( 0.0+0.0i -1.0+1.0i  1.0+2.0i  0.0+4.0i  1.0+1.0i )
C                 ( 2.0+0.0i  4.0+0.0i 10.0+0.0i  6.0+0.0i  9.0+0.0i )
C
      DATA A/(0.0,0.0), (2.0,0.0), (-1.0,1.0), (4.0, 0.0), (1.0,2.0),
     &       (10.0,0.0), (0.0,4.0), (6.0,0.0), (1.0,1.0), (9.0,0.0)/
C                                 Factor the matrix
      CALL LFTQH (N, A, LDA, NCODA, FAC, LDFAC)
C                                 Compute the determinant
      CALL LFDQH (N, FAC, LDFAC, NCODA, DET1, DET2)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) DET1, DET2
C
99999 FORMAT (’ The determinant of A is ’,F6.3,’ * 10**’,F2.0)
      END
```

Output

```
The determinant of A is  1.736 * 10**3.
```

LSLXG/DLSLXG (Single/Double precision)

Solve a sparse system of linear algebraic equations by Gaussian elimination.

Usage

```
CALL LSLXG (N, NZ, A, IROW, JCOL, B, IPATH, IPARAM, RPARAM,
            X)
```

Arguments

N — Number of equations. (Input)

NZ — The number of nonzero coefficients in the linear system. (Input)

A — Vector of length NZ containing the nonzero coefficients of the linear system. (Input)

IROW — Vector of length NZ containing the row numbers of the corresponding elements in A. (Input)

JCOL — Vector of length NZ containing the column numbers of the corresponding elements in A. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
IPATH = 1 means the system $Ax = b$ is solved.

IPATH = 2 means the system $A^T x = b$ is solved.

IPARAM — Parameter vector of length 6. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM.
See Comment 3.

RPARAM — Parameter vector of length 5. (Input/Output)
See Comment 3.

X — Vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is at least

   LSLXG 19N + 5 * MAXNZ units, or
   DLSLXG 21N + 6 * MAXNZ units.

   MAXNZ is the maximal number of nonzero elements at any stage of the Gaussian elimination. In the absence of other information, setting MAXNZ equal to 3 * NZ is recommended. Higher or lower values may be used depending on fill-in, see IPARAM(5) in Comment 3.

   Workspace may be explicitly provided, if desired, by use of L2LXG/DL2LXG. The reference is

```
CALL L2LXG (N, NZ, A, IROW, JCOL, B, IPATH,
            IPARAM,RPARAM, X, WK, LWK, IWK, LIWK)
```

The additional arguments are as follows:

WK — Real work vector of length LWK.

LWK — The length of WK, LWK should be at least 2N + MAXNZ.

IWK — Integer work vector of length LIWK.

LIWK — The length of IWK, LIWK should be at least 17N + 4 * MAXNZ.

The workspace limit is determined by MAXNZ, where

```
MAXNZ = MIN0(LWK-2N, INT(0.25(LIWK-17N)))
```

2. Informational errors

Type	Code	
3	1	The coefficient matrix is numerically singular.
3	2	The growth factor is too large to continue.
3	3	The matrix is too ill-conditioned for iterative refinement.

3. If the default parameters are desired for LSLXG, then set IPARAM(1) to zero and call the routine LSLXG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM. then the following steps should be taken before calling LSLXG.

```
CALL L4LXG (IPARAM, RPARAM)
```
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to L4LXG will set IPARAM and RPARAM to their default values, so only nondefault values need to be set above.

IPARAM — Integer vector of length 6.
IPARAM(1) = Initialization flag.

IPARAM(2) = The pivoting strategy

IPARAM(2)	**Action**
1	Markowitz row search
2	Markowitz column search
3	Symmetric Markowitz search

Default: 3.

IPARAM(3) = The number of rows which have least numbers of nonzero elements that will be searched for a pivotal element.
Default: 3.

IPARAM(4) = The maximal number of nonzero elements in A at any stage of the Gaussian elimination. (Output)

IPARAM(5) = The workspace limit.

IPARAM(5)	**Action**
0	Default limit, see Comment 1.
integer	This integer value replaces the default workspace limit. When L2LXG is called, the values of LWK and LIWK are used instead of IPARAM(5).

Default: 0.

IPARAM(6) = Iterative refinement is done when this is nonzero.
Default: 0.

RPARAM — Real vector of length 5.
RPARAM(1) = The upper limit on the growth factor. The computation stops when the growth factor exceeds the limit.
Default: 10^{16}.

RPARAM(2) = The stability factor. The absolute value of the pivotal element must be bigger than the largest element in absolute value in its row divided by RPARAM(2).
Default: 10.0.

RPARAM(3) = Drop-tolerance. Any element in the lower triangular factor L will be removed if its absolute value becomes smaller than the drop-tolerance at any stage of the Gaussian elimination.
Default: 0.0.

RPARAM(4) = The growth factor. It is calculated as the largest element in absolute value in A at any stage of the Gaussian elimination divided by the largest element in absolute value in the original A matrix. (Output)
Large value of the growth factor indicates that an appreciable error in the computed solution is possible.

RPARAM(5) = The value of the smallest pivotal element in absolute value. (Output)

If double precision is required, then DL4LXG is called and RPARAM is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is a $n \times n$ sparse matrix. The sparse coordinate format for the matrix A requires one real and two integer vectors. The real array a contains all the nonzeros in A. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column numbers for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in *A* zero.

The routine `LSLXG` solves a system of linear algebraic equations having a real sparse coefficient matrix. It first uses the routine `LFTXG` (page 199) to perform an *LU* factorization of the coefficient matrix. The solution of the linear system is then found using `LFSXG` (page 204).

The routine `LFTXG` by default uses a *symmetric Markowitz strategy* (Crowe et al. 1990) to choose pivots that most likely would reduce fill-ins while maintaining numerical stability. Different strategies are also provided as options for row oriented or column oriented problems. The algorithm can be expressed as

$$P\,AQ = LU$$

where *P* and *Q* are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and *L* and *U* are lower and upper triangular matrices, respectively.

Finally, the solution *x* is obtained by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

Example

As an example consider the 6 × 6 linear system:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let $x^T = (1, 2, 3, 4, 5, 6)$ so that $Ax = (10, 7, 45, 33, -34, 31)^T$. The number of nonzeros in *A* is `nz` = 15. The sparse coordinate form for *A* is given by:

irow	6	2	3	2	4	4	5	5	5	5	1	6	6	2	4
jcol	6	2	3	3	4	5	1	6	4	5	1	1	2	4	1
a	6	10	15	–3	10	–1	–1	–3	–5	1	10	–1	–2	–1	–2

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
C
      INTEGER    IPARAM(6), IPATH, IROW(NZ), JCOL(NZ)
      REAL       A(NZ), B(N), RPARAM(5), X(N)
      EXTERNAL   L4LXG, LSLXG, WRRRN
C
      DATA A/6., 10., 15., -3., 10., -1., -1., -3., -5., 1., 10., -1.,
     &     -2., -1., -2./
      DATA B/10., 7., 45., 33., -34., 31./
      DATA IROW/6, 2, 3, 2, 4, 4, 5, 5, 5, 5, 1, 6, 6, 2, 4/
      DATA JCOL/6, 2, 3, 3, 4, 5, 1, 6, 4, 5, 1, 1, 2, 4, 1/
C
      IPATH = 1
C                                 Change a default parameter
      CALL L4LXG (IPARAM, RPARAM)
      IPARAM(5) = 203
C                                 Solve for X
      CALL LSLXG (N, NZ, A, IROW, JCOL, B, IPATH, IPARAM, RPARAM, X)
C
      CALL WRRRN (’ x ’, 1, N, X, 1, 0)
      END
```

Output

```
                           x
    1       2       3       4       5       6
1.000   2.000   3.000   4.000   5.000   6.000
```

LFTXG/DLFTXG (Single/Double precision)

Compute the *LU* factorization of a real general sparse matrix.

Usage

```
CALL LFTXG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL,
            FAC, IRFAC, JCFAC, IPVT, JPVT)
```

Arguments

N — Number of equations.   (Input)

NZ — The number of nonzero coefficients in the linear system.   (Input)

A — Vector of length NZ containing the nonzero coefficients of the linear system.   (Input)

IROW — Vector of length NZ containing the row numbers of the corresponding elements in A.   (Input)

JCOL — Vector of length NZ containing the column numbers of the corresponding elements in A.   (Input)

IPARAM — Parameter vector of length 6.   (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 3.

RPARAM — Parameter vector of length 5.   (Input/Output)
See Comment 3.

NFAC — On input, the dimension of vector `FAC`. (Input/Output)
On output, the number of nonzero coefficients in the triangular matrix *L* and *U*.

NL — The number of nonzero coefficients in the triangular matrix *L* excluding the diagonal elements. (Output)

FAC — Vector of length `NFAC` containing the nonzero elements of *L* (excluding the diagonals) in the first `NL` locations and the nonzero elements of *U* in `NL` + 1 to `NFAC` locations. (Output)

IRFAC — Vector of length `NFAC` containing the row numbers of the corresponding elements in `FAC`. (Output)

JCFAC — Vector of length `NFAC` containing the column numbers of the corresponding elements in `FAC`. (Output)

IPVT — Vector of length `N` containing the row pivoting information for the *LU* factorization. (Output)

JPVT — Vector of length `N` containing the column pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

   `LFTXG` 15`N` + 5 * `MAXNZ` units, or
   `DLFTXG` 15`N` + 6 `MAXNZ` units.

   `MAXNZ` is the maximal number of nonzero elements at any stage of the Gaussian elimination. In the absence of other information, setting `MAXNZ` equal to 3`NZ` is recommended. Higher or lower values may be used depending on fill-in, see `IPARAM`(5) in Comment 3.

   Workspace may be explicitly provided, if desired, by use of `L2TXG`/`DL2TXG`. The reference is

   ```
   CALL L2TXG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM,
               NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT,
               WK, LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length `LWK`.

   LWK — The length of `WK`, `LWK` should be at least `MAXNZ`.

   IWK — Integer work vector of length `LIWK`.

   LIWK — The length of `IWK`, `LIWK` should be at least 15`N` + 4 ∗ `MAXNZ`.

   The workspace limit is determined by `MAXNZ`, where

   ```
   MAXNZ = MIN0(LWK, INT(0.25(LIWK-15N)))
   ```

2. Informational errors

Type	Code	
3	1	The coefficient matrix is numerically singular.
3	2	The growth factor is too large to continue.

3. If the default parameters are desired for LFTXG, then set IPARAM(1) to zero and call the routine LFTXG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling LFTXG.

   CALL L4LXG (IPARAM, RPARAM)
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to L4LXG will set IPARAM and RPARAM to their default values, so only nondefault values need to be set above.

   The arguments are as follows:

   IPARAM — Integer vector of length 6.

   IPARAM(1) = Initialization flag.

   IPARAM(2) = The pivoting strategy.

IPARAM(2)	**Action**
1	Markowitz row search
2	Markowitz column search
3	Symmetric Markowitz search

   Default: 3.

   IPARAM(3) = The number of rows which have least numbers of nonzero elements that will be searched for a pivotal element.
   Default: 3.

   IPARAM(4) = The maximal number of nonzero elements in A at any stage of the Gaussian elimination. (Output)

   IPARAM(5) = The workspace limit.

IPARAM(5)	**Action**
0	Default limit, see Comment 1.
integer	This integer value replaces the default workspace limit. When L2TXG is called, the values of LWK and LIWK are used instead of IPARAM(5).

   IPARAM(6) = Not used in LFTXG.

   RPARAM — Real vector of length 5.

   RPARAM(1) = The upper limit on the growth factor. The computation stops when the growth factor exceeds the limit.
   Default: 10^{16}.

RPARAM(2) = The stability factor. The absolute value of the pivotal element must be bigger than the largest element in absolute value in its row divided by RPARAM(2).
Default: 10.0.

RPARAM(3) = Drop-tolerance. Any element in the lower triangular factor L will be removed if its absolute value becomes smaller than the drop-tolerance at any stage of the Gaussian elimination.
Default: 0.0.

RPARAM(4) = The growth factor. It is calculated as the largest element in absolute value in A at any stage of the Gaussian elimination divided by the largest element in absolute value in the original A matrix. (Output) Large value of the growth factor indicates that an appreciable error in the computed solution is possible.

RPARAM(5) = The value of the smallest pivotal element in absolute value. (Output)

If double precision is required, then DL4LXG is called and RPARAM is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is a $n \times n$ sparse matrix. The sparse coordinate format for the matrix A requires one real and two integer vectors. The real array a contains all the nonzeros in A. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column numbers for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in A zero.

The routine LFTXG performs an LU factorization of the coefficient matrix A. It by default uses a *symmetric Markowitz strategy* (Crowe et al. 1990) to choose pivots that most likely would reduce fillins while maintaining numerical stability. Different strategies are also provided as options for row oriented or column oriented problems. The algorithm can be expressed as

$$P\,AQ = LU$$

where P and Q are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and L and U are lower and upper triangular matrices, respectively.

Finally, the solution *x* is obtained using `LFSXG` (page 204) by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

Example

As an example, consider the 6 × 6 matrix of a linear system:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

The sparse coordinate form for *A* is given by:

irow	6	2	3	2	4	4	5	5	5	5	1	6	6	2	4
jcol	6	2	3	3	4	5	1	6	4	5	1	1	2	4	1
a	6	10	15	–3	10	–1	–1	–3	–5	1	10	–1	–2	–1	–2

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
      INTEGER    IPARAM(6), IROW(NZ), JCOL(NZ), NFAC, NL,
     &           IRFAC(3*NZ), JCFAC(3*NZ), IPVT(N), JPVT(N)
      REAL       RPARAM(5), A(NZ), FAC(3*NZ)
C
      DATA A/6., 10., 15., -3., 10., -1., -1., -3., -5., 1., 10., -1.,
     &       -2., -1., -2./
      DATA IROW/6, 2, 3, 2, 4, 4, 5, 5, 5, 5, 1, 6, 6, 2, 4/
      DATA JCOL/6, 2, 3, 3, 4, 5, 1, 6, 4, 5, 1, 1, 2, 4, 1/
C
      NFAC = 3*NZ
C                                 Use default options
      IPARAM(1) = 0
      CALL LFTXG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL,
     &            FAC, IRFAC, JCFAC, IPVT, JPVT)
C
      CALL WRRRN (’ fac ’, 1, NFAC, FAC, 1, 0)
      CALL WRIRN (’ irfac ’, 1, NFAC, IRFAC, 1, 0)
      CALL WRIRN (’ jcfac ’, 1, NFAC, JCFAC, 1, 0)
      CALL WRIRN (’ p ’, 1, N, IPVT, 1, 0)
      CALL WRIRN (’ q ’, 1, N, JPVT, 1, 0)
C
      END
```

Output

```
                                              fac
    1       2       3       4       5       6       7       8       9      10
-0.10   -5.00   -0.20   -0.10   -0.10   -1.00   -0.20    4.90   -5.10    1.00
   11      12      13      14      15      16
-1.00   30.00    6.00   -2.00   10.00   15.00

                              irfac
1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
3   4   4   5   5   6   6   6   5   5   4   4   3   3   2   1

                              jcfac
1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
2   3   1   4   2   5   2   6   6   5   6   4   4   3   2   1

            p
1   2   3   4   5   6
3   1   6   2   5   4

            q
1   2   3   4   5   6
3   1   2   6   5   4
```

LFSXG/DLFSXG (Single/Double precision)

Solve a sparse system of linear equations given the *LU* factorization of the coefficient matrix.

Usage

```
CALL LFSXG (N, NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT, B,
            IPATH, X)
```

Arguments

N — Number of equations. (Input)

NFAC — The number of nonzero coefficients in FAC as output from subroutine LFTXG/DLFTXG. (Input)

NL — The number of nonzero coefficients in the triangular matrix *L* excluding the diagonal elements as output from subroutine LFTXG/DLFTXG. (Input)

FAC — Vector of length NFAC containing the nonzero elements of *L* (excluding the diagonals) in the first NL locations and the nonzero elements of *U* in NL + 1 to NFAC locations as output from subroutine LFTXG/DLFTXG. (Input)

IRFAC — Vector of length NFAC containing the row numbers of the corresponding elements in FAC as output from subroutine LFTXG/DLFTXG. (Input)

JCFAC — Vector of length NFAC containing the column numbers of the corresponding elements in FAC as output from subroutine LFTXG/DLFTXG. (Input)

IPVT — Vector of length `N` containing the row pivoting information for the *LU* factorization as output from subroutine `LFTXG`/`DLFTXG`. (Input)

JPVT — Vector of length `N` containing the column pivoting information for the *LU* factorization as output from subroutine `LFTXG`/`DLFTXG`. (Input)

B — Vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system $Ax = B$ is solved.
`IPATH` = 2 means the system $A^T x = B$ is solved,

X — Vector of length `N` containing the solution to the linear system. (Output)

Algorithm

Consider the linear equation

$$Ax = b$$

where A is a $n \times n$ sparse matrix. The sparse coordinate format for the matrix A requires one real and two integer vectors. The real array `a` contains all the nonzeros in A. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column numbers for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in A zero. The routine `LFSXG` computes the solution of the linear equation given its *LU* factorization. The factorization is performed by calling `LFTXG` (page 199). The solution of the linear system is then found by the forward and backward substitution. The algorithm can be expressed as

$$P\,AQ = LU$$

where P and Q are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and L and U are lower and upper triangular matrices, respectively. Finally, the solution x is obtained by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

For more details, see Crowe et al. (1990).

Example

As an example, consider the 6 × 6 linear system:

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

Let

$$x_1^T = (1, 2, 3, 4, 5, 6)$$

so that $Ax_1 = (10, 7, 45, 33, –34, 31)^T$, and

$$x_2^T = (6, 5, 4, 3, 2, 1)$$

so that $Ax_2 = (60, 35, 60, 16, –22, 10)^T$. The sparse coordinate form for A is given by:

irow	6	2	3	2	4	4	5	5	5	5	1	6	6	2	4
jcol	6	2	3	3	4	5	1	6	4	5	1	1	2	4	1
a	6	10	15	–3	10	–1	–1	–3	–5	1	10	–1	–2	–1	–2

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
      INTEGER    IPARAM(8), IPATH, IROW(NZ), JCOL(NZ), NFAC,
     &           NL, IRFAC(3*NZ), JCFAC(3*NZ), IPVT(N), JPVT(N)
      REAL       RPARAM(10), X(N), A(NZ), B(N,2), FAC(3*NZ)
      CHARACTER  TITLE(2)*2
C
      DATA A/6., 10., 15., -3., 10., -1., -1., -3., -5., 1., 10., -1.,
     &       -2., -1., -2./
      DATA B/10., 7., 45., 33., -34., 31.,
     &       60., 35., 60., 16., -22., -10./
      DATA IROW/6, 2, 3, 2, 4, 4, 5, 5, 5, 5, 1, 6, 6, 2, 4/
      DATA JCOL/6, 2, 3, 3, 4, 5, 1, 6, 4, 5, 1, 1, 2, 4, 1/
      DATA TITLE/'x1', 'x2'/
C
      NFAC = 3*NZ
C                                 Use default options
      IPARAM(1) = 0
C                                 Perform LU factorization
      CALL LFTXG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL,
     &            FAC, IRFAC, JCFAC, IPVT, JPVT)
C
      IPATH = 1
      DO 10 I = 1, 2
C                                 Solve A * X(i) = B(i)
         CALL LFSXG (N, NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT,
     &               B(1,I), IPATH, X)
C
```

```
         CALL WRRRL (TITLE(I), 1, N, X, 1, 0, '(f4.1)', 'NONE',
     &               'NUMBER')
   10 CONTINUE
      END
```

Output

```
                  x1
  1     2     3     4     5     6
1.0   2.0   3.0   4.0   5.0   6.0

                 x2
  1     2     3     4     5     6
6.0   5.0   4.0   3.0   2.0   1.0
```

LSLZG/DLSLZG (Single/Double precision)

Solve a complex sparse system of linear equations by Gaussian elimination.

Usage

```
CALL LSLZG (N, NZ, A, IROW, JCOL, B, IPATH, IPARAM, RPARAM,
            X)
```

Arguments

N — Number of equations. (Input)

NZ — The number of nonzero coefficients in the linear system. (Input)

A — Complex vector of length `NZ` containing the nonzero coefficients of the linear system. (Input)

IROW — Vector of length `NZ` containing the row numbers of the corresponding elements in `A`. (Input)

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in `A`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system $Ax = b$ is solved.
`IPATH` = 2 means the system $A^H x = b$ is solved.

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 3.

RPARAM — Parameter vector of length 5. (Input/Output)
See Comment 3.

X — Complex vector of length `N` containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is at least

   LSLZG 21N + 6 * MAXNZ units, or
   DLSLZG 23N + 8 * MAXNZ units.

   MAXNZ is the maximal number of nonzero elements at any stage of the Gaussian elimination. In the absence of other information, setting MAXNZ equal to 3NZ is recommended. Higher or lower values may be used depending on fill-in, see IPARAM(5) in Comment 3.

   Workspace may be explicitly provided, if desired, by use of L2LZG/DL2LZG. The reference is

   ```
   CALL L2LZG (N, NZ, A, IROW, JCOL, B, IPATH, IPARAM,
               RPARAM, X, WK, LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   WK — Complex work vector of length LWK.

   LWK — The length of WK, LWK should be at least 2N+ MAXNZ.

   IWK — Integer work vector of length LIWK.

   LIWK — The length of IWK, LIWK should be at least 17N + 4 * MAXNZ.

   The workspace limit is determined by MAXNZ, where

   MAXNZ = MIN0(LWK-2N, INT(0.25(LIWK-17N)))

2. Informational errors

Type	Code	
3	1	The coefficient matrix is numerically singular.
3	2	The growth factor is too large to continue.
3	3	The matrix is too ill-conditioned for iterative refinement.

3. If the default parameters are desired for LSLZG, then set IPARAM(1) to zero and call the routine LSLZG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM. then the following steps should be taken before calling LSLZG.

   CALL L4LZG (IPARAM, RPARAM)
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to L4LZG will set IPARAM and RPARAM to their default values, so only nondefault values need to be set above. The arguments are as follows:

   IPARAM — Integer vector of length 6.

   IPARAM(1) = Initialization flag.

IPARAM(2) = The pivoting strategy.

IPARAM(2)	**Action**
1	Markowitz row search
2	Markowitz column search
3	Symmetric Markowitz search

Default: 3.

IPARAM(3) = The number of rows which have least numbers of nonzero elements that will be searched for a pivotal element.
Default: 3.

IPARAM(4) = The maximal number of nonzero elements in *A* at any stage of the Gaussian elimination. (Output)

IPARAM(5) = The workspace limit.

IPARAM(5)	**Action**
0	Default limit, see Comment 1.
integer	This integer value replaces the default workspace limit. When L2LZG is called, the values of LWK and LIWK are used instead of IPARAM(5).

Default: 0.

IPARAM(6) = Iterative refinement is done when this is nonzero.
Default: 0.

RPARAM — Real vector of length 5.

RPARAM(1) = The upper limit on the growth factor. The computation stops when the growth factor exceeds the limit.
Default: 10^{16}.

RPARAM(2) = The stability factor. The absolute value of the pivotal element must be bigger than the largest element in absolute value in its row divided by RPARAM(2).
Default: 10.0.

RPARAM(3) = Drop-tolerance. Any element in A will be removed if its absolute value becomes smaller than the drop-tolerance at any stage of the Gaussian elimination.
Default: 0.0.

RPARAM(4) = The growth factor. It is calculated as the largest element in absolute value in A at any stage of the Gaussian elimination divided by the largest element in absolute value in the original A matrix. (Output)
Large value of the growth factor indicates that an appreciable error in the computed solution is possible.

RPARAM(5) = The value of the smallest pivotal element in absolute value. (Output)

If double precision is required, then DL4LZG is called and RPARAM is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where *A* is a *n* × *n* complex sparse matrix. The sparse coordinate format for the matrix *A* requires one complex and two integer vectors. The complex array a contains all the nonzeros in *A*. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column numbers for these entries in *A*. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in *A* zero.

The subroutine LSLZG solves a system of linear algebraic equations having a complex sparse coefficient matrix. It first uses the routine LFTZG (page 212) to perform an *LU* factorization of the coefficient matrix. The solution of the linear system is then found using LFSZG (page 217). The routine LFTZG by default uses a *symmetric Markowitz strategy* (Crowe et al. 1990) to choose pivots that most likely would reduce fill-ins while maintaining numerical stability. Different strategies are also provided as options for row oriented or column oriented problems. The algorithm can be expressed as

$$P\,AQ = LU$$

where *P* and *Q* are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and *L* and *U* are lower and upper triangular matrices, respectively. Finally, the solution *x* is obtained by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

Example

As an example, consider the 6 × 6 linear system:

$$A = \begin{bmatrix} 10+7i & 0 & 0 & 0 & 0 & 0 \\ 0 & 3+2i & -3+0i & -1+2i & 0 & 0 \\ 0 & 0 & 4+2i & 0 & 0 & 0 \\ -2-4i & 0 & 0 & 1+6i & -1+3i & 0 \\ -5+4i & 0 & 0 & -5+0i & 12+2i & -7+7i \\ -1+12i & -2+8i & 0 & 0 & 0 & 3+7i \end{bmatrix}$$

Let

$$x^T = (1 + i, 2 + 2i, 3 + 3i, 4 + 4i, 5 + 5i, 6 + 6i)$$

so that

$$Ax = (3 + 17i, -19 + 5i, 6 + 18i, -38 + 32i, -63 + 49i, -57 + 83i)^T$$

The number of nonzeros in A is `nz` = 15. The sparse coordinate form for A is given by:

irow	6	2	2	4	3	1	5	4	6	5	5	6	4	2	5
jcol	6	2	3	5	3	1	1	4	1	4	5	2	1	4	6

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
C
      INTEGER    IPARAM(6), IPATH, IROW(NZ), JCOL(NZ)
      REAL       RPARAM(5)
      COMPLEX    A(NZ), B(N), X(N)
      EXTERNAL   LSLZG, WRCRN
C
      DATA A/(3.0,7.0), (3.0,2.0), (-3.0,0.0), (-1.0,3.0), (4.0,2.0),
     &     (10.0,7.0), (-5.0,4.0), (1.0,6.0), (-1.0,12.0), (-5.0,0.0),
     &     (12.0,2.0), (-2.0,8.0), (-2.0,-4.0), (-1.0,2.0), (-7.0,7.0)/
      DATA B/(3.0,17.0), (-19.0,5.0), (6.0,18.0), (-38.0,32.0),
     &     (-63.0,49.0), (-57.0,83.0)/
      DATA IROW/6, 2, 2, 4, 3, 1, 5, 4, 6, 5, 5, 6, 4, 2, 5/
      DATA JCOL/6, 2, 3, 5, 3, 1, 1, 4, 1, 4, 5, 2, 1, 4, 6/
C
      IPATH     = 1
C                                Use default options
      IPARAM(1) = 0
      CALL LSLZG (N, NZ, A, IROW, JCOL, B, IPATH, IPARAM, RPARAM, X)
C
      CALL WRCRN ('X', N, 1, X, N, 0)
      END
```

Output

```
           X
1  ( 1.000, 1.000)
2  ( 2.000, 2.000)
3  ( 3.000, 3.000)
4  ( 4.000, 4.000)
```

```
5  ( 5.000, 5.000)
6  ( 6.000, 6.000)
```

LFTZG/DLFTZG (Single/Double precision)

Compute the *LU* factorization of a complex general sparse matrix.

Usage

```
CALL LFTZG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL,
            FAC, IRFAC, JCFAC, IPVT, JPVT)
```

Arguments

N — Number of equations. (Input)

NZ — The number of nonzero coefficients in the linear system. (Input)

A — Complex vector of length `NZ` containing the nonzero coefficients of the linear system. (Input)

IROW — Vector of length `NZ` containing the row numbers of the corresponding elements in `A`. (Input)

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in `A`. (Input)

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 3.

RPARAM — Parameter vector of length 5. (Input/Output)
See Comment 3.

NFAC — On input, the dimension of vector `FAC`. (Input/Output)
On output, the number of nonzero coefficients in the triangular matrix *L* and *U*.

NL — The number of nonzero coefficients in the triangular matrix *L* excluding the diagonal elements. (Output)

FAC — Complex vector of length `NFAC` containing the nonzero elements of *L* (excluding the diagonals) in the first `NL` locations and the nonzero elements of *U* in `NL` + 1 to `NFAC` locations. (Output)

IRFAC — Vector of length `NFAC` containing the row numbers of the corresponding elements in `FAC`. (Output)

JCFAC — Vector of length `NFAC` containing the column numbers of the corresponding elements in `FAC`. (Output)

IPVT — Vector of length `N` containing the row pivoting information for the *LU* factorization. (Output)

JPVT — Vector of length N containing the column pivoting information for the *LU* factorization. (Output)

Comments

1. Automatic workspace usage is

   LFTZG 15N + 6 * MAXNZ units, or
   DLFTZG 15N + 8 * MAXNZ units.

   MAXNZ is the maximal number of nonzero elements at any stage of the Gaussian elimination. In the absence of other information, setting MAXNZ equal to 3 ∗ NZ is recommended. Higher or lower values may be used depending on fill-in, see IPARAM(5) in Comment 3.

   Workspace may be explicitly provided, if desired, by use of L2TZG/DL2TZG. The reference is

   ```
   CALL L2TZG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM,
               NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT,
               WK, LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   WK — Complex work vector of length LWK.

   LWK — The length of WK, LWK should be at least MAXNZ.

   IWK — Integer work vector of length LIWK.

   LIWK — The length of IWK, LIWK should be at least 15N + 4 * MAXNZ.

   The workspace limit is determined by MAXNZ, where

   ```
   MAXNZ = MIN0(LWK, INT(0.25(LIWK-15N)))
   ```

2. Informational errors

Type	Code	
3	1	The coefficient matrix is numerically singular.
3	2	The growth factor is too large to continue.

3. If the default parameters are desired for LFTZG, then set IPARAM(1) to zero and call the routine LFTZG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM. then the following steps should be taken before calling LFTZG:

   CALL L4LZG (IPARAM, RPARAM)
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to L4LZG will set IPARAM and RPARAM to their default values so only nondefault values need to be set above. The arguments are as follows:

   IPARAM — Integer vector of length 6.

   IPARAM(1) = Initialization flag.

IPARAM(2) = The pivoting strategy.

IPARAM(2)	**Action**
1	Markowitz row search
2	Markowitz column search
3	Symmetric Markowitz search

Default: 3.

IPARAM(3) = The number of rows which have least numbers of nonzero elements that will be searched for a pivotal element.
Default: 3.

IPARAM(4) = The maximal number of nonzero elements in A at any stage of the Gaussian elimination. (Output)

IPARAM(5) = The workspace limit.

IPARAM(5)	**Action**
0	Default limit, see Comment 1.
integer	This integer value replaces the default workspace limit. When L2TZG is called, the values of LWK and LIWK are used instead of IPARAM(5).

Default: 0.

IPARAM(6) = Not used in LFTZG.

RPARAM — Real vector of length 5.

RPARAM(1) = The upper limit on the growth factor. The computation stops when the growth factor exceeds the limit.
Default: 10^{16}.

RPARAM(2) = The stability factor. The absolute value of the pivotal element must be bigger than the largest element in absolute value in its row divided by RPARAM(2).
Default: 10.0.

RPARAM(3) = Drop-tolerance. Any element in the lower triangular factor *L* will be removed if its absolute value becomes smaller than the drop-tolerance at any stage of the Gaussian elimination.
Default: 0.0.

RPARAM(4) = The growth factor. It is calculated as the largest element in absolute value in A at any stage of the Gaussian elimination divided by the largest element in absolute value in the original A matrix. (Output)
Large value of the growth factor indicates that an appreciable error in the computed solution is possible.

RPARAM(5) = The value of the smallest pivotal element in absolute value. (Output)

If double precision is required, then `DL4LZG` is called and `RPARAM` is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is a complex $n \times n$ sparse matrix. The sparse coordinate format for the matrix A requires one complex and two integer vectors. The complex array `a` contains all the nonzeros in A. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column indices for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in A zero.

The routine `LFTZG` performs an LU factorization of the coefficient matrix A. It uses by default a *symmetric Markowitz strategy* (Crowe et al. 1990) to choose pivots that most likely would reduce fill-ins while maintaining numerical stability. Different strategies are also provided as options for row oriented or column oriented problems. The algorithm can be expressed as

$$P\,AQ = LU$$

where P and Q are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and L and U are lower and upper triangular matrices, respectively.

Finally, the solution x is obtained using `LFSZG` (page 217) by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

Example

As an example, the following 6×6 matrix is factorized, and the outcome is printed:

$$A = \begin{bmatrix} 10+7i & 0 & 0 & 0 & 0 & 0 \\ 0 & 3+2i & -3+0i & -1+2i & 0 & 0 \\ 0 & 0 & 4+2i & 0 & 0 & 0 \\ -2-4i & 0 & 0 & 1+6i & -1+3i & 0 \\ -5+4i & 0 & 0 & -5+0i & 12+2i & -7+7i \\ -1+12i & -2+8i & 0 & 0 & 0 & 3+7i \end{bmatrix}$$

The sparse coordinate form for *A* is given by:

irow	6	2	2	4	3	1	5	4	6	5	5	6	4	2	5
jcol	6	2	3	5	3	1	1	4	1	4	5	2	1	4	6

```
      INTEGER    N, NFAC, NZ
      PARAMETER  (N=6, NZ=15, NFAC=3*NZ)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IPARAM(6), IPVT(N), IRFAC(NFAC), IROW(NZ), JCFAC(NFAC),
     &           JCOL(NZ), JPVT(N), NL
      REAL       RPARAM(5)
      COMPLEX    A(NZ), FAC(NFAC)
C
      DATA A/(3.0,7.0), (3.0,2.0), (-3.0,0.0), (-1.0,3.0), (4.0,2.0),
     &     (10.0,7.0), (-5.0,4.0), (1.0,6.0), (-1.0,12.0), (-5.0,0.0),
     &     (12.0,2.0), (-2.0,8.0), (-2.0,-4.0), (-1.0,2.0), (-7.0,7.0)/
      DATA IROW/6, 2, 2, 4, 3, 1, 5, 4, 6, 5, 5, 6, 4, 2, 5/
      DATA JCOL/6, 2, 3, 5, 3, 1, 1, 4, 1, 4, 5, 2, 1, 4, 6/
C
      IPARAM(1) = 0
C                                 Use default options
      CALL LFTZG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL, FAC,
     &            IRFAC, JCFAC, IPVT, JPVT)
C
      CALL WRCRN ('fac',NFAC,1,FAC,NFAC,0)
      CALL WRIRN (' irfac ', 1, NFAC, IRFAC, 1, 0)
      CALL WRIRN (' jcfac ', 1, NFAC, JCFAC, 1, 0)
      CALL WRIRN (' p ', 1, N, IPVT, 1, 0)
      CALL WRIRN (' q ', 1, N, JPVT, 1, 0)
C
      END
```

Output

```
           fac
 1  (  0.50,  0.85)
 2  (  0.15, -0.41)
 3  ( -0.60,  0.30)
 4  (  2.23, -1.97)
 5  ( -0.15,  0.50)
 6  ( -0.04,  0.26)
 7  ( -0.32, -0.17)
 8  ( -0.92,  7.46)
 9  ( -6.71, -6.42)
10  ( 12.00,  2.00)
11  ( -1.00,  2.00)
12  ( -3.32,  0.21)
13  (  3.00,  7.00)
14  ( -2.00,  8.00)
15  ( 10.00,  7.00)
16  (  4.00,  2.00)

                                      irfac
1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
3   4   4   5   5   6   6   6   5   5   4   4   3   3   2   1
```

```
                                   jcfac
1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
2    3    1    4    2    5    2    6    6    5    6    4    4    3    2    1

               p
1    2    3    4    5    6
3    1    6    2    5    4

               q
1    2    3    4    5    6
3    1    2    6    5    4
```

LFSZG/DLFSZG (Single/Double precision)

Solve a complex sparse system of linear equations given the *LU* factorization of the coefficient matrix.

Usage

```
CALL LFSZG (N, NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT, B,
            IPATH, X)
```

Arguments

N — Number of equations. (Input)

NFAC — The number of nonzero coefficients in `FAC` as output from subroutine `LFTZG`/`DLFTZG`. (Input)

NL — The number of nonzero coefficients in the triangular matrix *L* excluding the diagonal elements as output from subroutine `LFTZG`/`DLFTZG`. (Input)

FAC — Complex vector of length `NFAC` containing the nonzero elements of *L* (excluding the diagonals) in the first `NL` locations and the nonzero elements of *U* in `NL`+ 1 to `NFAC` locations as output from subroutine `LFTZG`/`DLFTZG`. (Input)

IRFAC — Vector of length `NFAC` containing the row numbers of the corresponding elements in `FAC` as output from subroutine `LFTZG`/`DLFTZG`. (Input)

JCFAC — Vector of length `NFAC` containing the column numbers of the corresponding elements in `FAC` as output from subroutine `LFTZG`/`DLFTZG`. (Input)

IPVT — Vector of length `N` containing the row pivoting information for the *LU* factorization as output from subroutine `LFTZG`/`DLFTZG`. (Input)

JPVT — Vector of length `N` containing the column pivoting information for the *LU* factorization as output from subroutine `LFTZG`/`DLFTZG`. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Path indicator. (Input)
`IPATH` = 1 means the system $Ax = b$ is solved.
`IPATH` = 2 means the system $A^H x = b$ is solved .

X — Complex vector of length `N` containing the solution to the linear system. (Output)

Algorithm

Consider the linear equation

$$Ax = b$$

where A is a complex $n \times n$ sparse matrix. The sparse coordinate format for the matrix A requires one complex and two integer vectors. The complex array `a` contains all the nonzeros in A. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column numbers for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in A zero.

The routine `LFSZG` computes the solution of the linear equation given its LU factorization. The factorization is performed by calling `LFTZG` (page 212). The solution of the linear system is then found by the forward and backward substitution. The algorithm can be expressed as

$$P\,AQ = LU$$

where P and Q are the row and column permutation matrices determined by the Markowitz strategy (Duff et al. 1986), and L and U are lower and upper triangular matrices, respectively.

Finally, the solution x is obtained by the following calculations:

$$1)\ Lz = Pb$$

$$2)\ Uy = z$$

$$3)\ x = Qy$$

For more details, see Crowe et al. (1990).

Example

As an example, consider the 6 × 6 linear system:

$$A=\begin{bmatrix} 10+7i & 0 & 0 & 0 & 0 & 0 \\ 0 & 3+2i & -3+0i & -1+2i & 0 & 0 \\ 0 & 0 & 4+2i & 0 & 0 & 0 \\ -2-4i & 0 & 0 & 1+6i & -1+3i & 0 \\ -5+4i & 0 & 0 & -5+0i & 12+2i & -7+7i \\ -1+12i & -2+8i & 0 & 0 & 0 & 3+7i \end{bmatrix}$$

Let

$$x_1{}^T=(1+i,2+2i,3+3i,4+4i,5+5i,6+6i)$$

so that

$$Ax_1 = (3 + 17i, -19 + 5i, 6 + 18i, -38 + 32i, -63 + 49i, -57 + 83i)^T$$

and

$$x_2{}^T=(6+6i,5+5i,4+4i,3+3i,2+2i,1+i)$$

so that

$$Ax_2 = (18 + 102i, -16 + 16i, 8 + 24i, -11 -11i, -63 + 7i, -132 + 106i)^T$$

The sparse coordinate form for *A* is given by:

irow	6	2	2	4	3	1	5	4	6	5	5	6	4	2	5
jcol	6	2	3	5	3	1	1	4	1	4	5	2	1	4	6

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
C
      INTEGER    IPARAM(6), IPATH, IPVT(N), IRFAC(3*NZ), IROW(NZ),
     &           JCFAC(3*NZ), JCOL(NZ), JPVT(N), NFAC, NL
      REAL       RPARAM(5)
      COMPLEX    A(NZ), B(N,2), FAC(3*NZ), X(N)
      CHARACTER  TITLE(2)*2
C
      DATA A/(3.0,7.0), (3.0,2.0), (-3.0,0.0), (-1.0,3.0), (4.0,2.0),
     &     (10.0,7.0), (-5.0,4.0), (1.0,6.0), (-1.0,12.0), (-5.0,0.0),
     &     (12.0,2.0), (-2.0,8.0), (-2.0,-4.0), (-1.0,2.0), (-7.0,7.0)/
      DATA B/(3.0,17.0), (-19.0,5.0), (6.0,18.0), (-38.0,32.0),
     &     (-63.0,49.0), (-57.0,83.0), (18.0,102.0), (-16.0,16.0),
     &     (8.0,24.0), (-11.0,-11.0), (-63.0,7.0), (-132.0,106.0)/
      DATA IROW/6, 2, 2, 4, 3, 1, 5, 4, 6, 5, 5, 6, 4, 2, 5/
      DATA JCOL/6, 2, 3, 5, 3, 1, 1, 4, 1, 4, 5, 2, 1, 4, 6/
      DATA TITLE/'x1','x2'/
C
      NFAC = 3*NZ
C                                 Use default options
      IPARAM(1) = 0
C                                 Perform LU factorization
      CALL LFTZG (N, NZ, A, IROW, JCOL, IPARAM, RPARAM, NFAC, NL, FAC,
     &            IRFAC, JCFAC, IPVT, JPVT)
C
      IPATH = 1
      DO 10 I = 1,2
C                                Solve A * X(i) = B(i)
         CALL LFSZG (N, NFAC, NL, FAC, IRFAC, JCFAC, IPVT, JPVT,
     &               B(1,I), IPATH, X)
         CALL WRCRN (TITLE(I), N, 1, X, N, 0)
   10 CONTINUE
C
      END
```

Output

```
          x1
1  ( 1.000, 1.000)
2  ( 2.000, 2.000)
3  ( 3.000, 3.000)
4  ( 4.000, 4.000)
5  ( 5.000, 5.000)
6  ( 6.000, 6.000)

         x2
1  ( 6.000, 6.000)
2  ( 5.000, 5.000)
3  ( 4.000, 4.000)
4  ( 3.000, 3.000)
5  ( 2.000, 2.000)
6  ( 1.000, 1.000)
```

LSLXD/DLSLXD (Single/Double precision)

Solve a sparse system of symmetric positive definite linear algebraic equations by Gaussian elimination.

Usage

`CALL LSLXD (N, NZ, A, IROW, JCOL, B, ITWKSP, X)`

Arguments

N — Number of equations.   (Input)

NZ — The number of nonzero coefficients in the lower triangle of the linear system.   (Input)

A — Vector of length `NZ` containing the nonzero coefficients in the lower triangle of the linear system.   (Input)
The sparse matrix has nonzeroes only in entries (`IROW` (*i*), `JCOL`(*i*)) for *i* = 1 to `NZ`, and at this location the sparse matrix has value `A`(*i*).

IROW — Vector of length `NZ` containing the row numbers of the corresponding elements in the lower triangle of `A`.   (Input)
Note `IROW`(*i*) ≥ `JCOL`(*i*), since we are only indexing the lower triangle.

JCOL — Vector of length NZ containing the column numbers of the corresponding elements in the lower triangle of A. (Input)

B — Vector of length N containing the right-hand side of the linear system. (Input)

ITWKSP — The total workspace needed. (Input)
If the default is desired, set ITWKSP to zero. See Comment 1 for the default.

X — Vector of length N containing the solution to the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSLXD 18N + 21NZ + 9 units, or

   DLSLXD 20N + 27NZ + 9 units. This is the default if ITWKSP is zero. If the value is positive, ITWKSP units are automatically allocated.

   Workspace may be explicitly provided, if desired, by use of L2LXD/DL2LXD. The reference is

   ```
   CALL L2LXD (N, NZ, A, IROW, JCOL, B, X, IPER,
               IPARAM, RPARAM, WK, LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   IPER — Vector of length N containing the ordering.

   IPARAM — Integer vector of length 4. See Comment 3.

   RPARAM — Real vector of length 2. See Comment 3.

   WK — Real work vector of length LWK.

   LWK — The length of WK, LWK should be at least 2N + 6NZ.

   IWK — Integer work vector of length LIWK.

   LIWK — The length of IWK, LIWK should be at least 15N + 15NZ + 9.

   Note that the parameter ITWKSP is not an argument to this routine.

2. Informational errors

Type	Code	
4	1	The coefficient matrix is not positive definite.
4	2	A column without nonzero elements has been found in the coefficient matrix.

3. If the default parameters are desired for L2LXD, then set IPARAM(1) to zero and call the routine L2LXD. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling L2LXD.

   ```
   CALL L4LXD (IPARAM, RPARAM)
   ```
   Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to `L4LXD` will set `IPARAM` and `RPARAM` to their default values, so only nondefault values need to be set above. The arguments are as follows:

IPARAM — Integer vector of length 4.

`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = The numerical factorization method.

`IPARAM(2)`	**Action**
0	Multifrontal
1	Sparse column

Default: 0.

`IPARAM`(3) = The ordering option.

`IPARAM(3)`	**Action**
0	Minimum degree ordering
1	User's ordering specified in `IPER`

Default: 0.

`IPARAM`(4) = The total number of nonzeros in the factorization matrix.

RPARAM — Real vector of length 2.

`RPARAM`(1) = The value of the largest diagonal element in the Cholesky factorization.

`RPARAM`(2) = The value of the smallest diagonal element in the Cholesky factorization.

If double precision is required, then `DL4LXD` is called and `RPARAM` is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where *A* is sparse, positive definite and symmetric. The sparse coordinate format for the matrix *A* requires one real and two integer vectors. The real array `a` contains all the nonzeros in the *lower triangle* of *A* including the diagonal. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column indices for these entries in *A*. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \dots, \mathtt{nz}$$

$$\mathtt{irow}(i) \ge \mathtt{jcol}(i) \qquad i = 1, \dots, \mathtt{nz}$$

with all other entries in the lower triangle of *A* zero.

The subroutine `LSLXD` solves a system of linear algebraic equations having a real, sparse and positive definite coefficient matrix. It first uses the routine `LSCXD` (page 224) to compute a symbolic factorization of a permutation of the coefficient matrix. It then calls `LNFXD` (page 228) to perform the numerical factorization. The solution of the linear system is then found using `LFSXD` (page 232).

The routine `LSCXD` computes a minimum degree ordering or uses a user-supplied ordering to set up the sparse data structure for the Cholesky factor, *L*. Then the routine `LNFXD` produces the numerical entries in *L* so that we have

$$P\,AP^T = LL^T$$

Here *P* is the permutation matrix determined by the ordering.

The numerical computations can be carried out in one of two ways. The first method performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Finally, the solution *x* is obtained by the following calculations:

$$1)\ Ly_1 = Pb$$

$$2)\ L^T y_2 = y_1$$

$$3)\ x = P^T y_2$$

The routine `LFSXD` accepts *b* and the permutation vector which determines *P*. It then returns *x*.

Example

As an example consider the 5 × 5 linear system:

$$A = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

Let $x^T = (1, 2, 3, 4, 5)$ so that $Ax = (23, 55, 107, 197, 278)^T$. The number of nonzeros in the lower triangle of *A* is `nz` = 10. The sparse coordinate form for the lower triangle of *A* is given by:

irow	1	2	3	3	4	4	5	5	5	5
jcol	1	2	1	3	3	4	1	2	4	5
a	10	20	1	30	4	40	2	3	5	50

or equivalently by

irow	4	5	5	5	1	2	3	3	4	5
jcol	4	1	2	4	1	2	1	3	3	5
a	40	2	3	5	10	20	1	30	4	50

```
      INTEGER    N, NZ
      PARAMETER  (N=5, NZ=10)
C
      INTEGER    IROW(NZ), JCOL(NZ), ITWKSP
      REAL       A(NZ), B(N), X(N)
      EXTERNAL   LSLXD, WRRRN
C
      DATA A/10., 20., 1., 30., 4., 40., 2., 3., 5., 50./
      DATA B/23., 55., 107., 197., 278./
      DATA IROW/1, 2, 3, 3, 4, 4, 5, 5, 5, 5/
      DATA JCOL/1, 2, 1, 3, 3, 4, 1, 2, 4, 5/
C                                 Use default workspace
      ITWKSP = 0
C                                 Solve A * X = B
      CALL LSLXD (N, NZ, A, IROW, JCOL, B, ITWKSP, X)
C                                 Print results
      CALL WRRRN (’ x ’, 1, N, X, 1, 0)
      END
```

Output

```
                     x
    1       2       3       4       5
1.000   2.000   3.000   4.000   5.000
```

LSCXD/DLSCXD (Single/Double precision)

Perform the symbolic Cholesky factorization for a sparse symmetric matrix using a minimum degree ordering or a user-specified ordering, and set up the data structure for the numerical Cholesky factorization.

Usage

```
CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
```

Arguments

N — Number of equations.   (Input)

NZ — Total number of the nonzeros in the lower triangular part of the symmetric matrix, including the nonzeros on the diagonal.   (Input)

IROW — Vector of length NZ containing the row subscripts of the nonzeros in the lower triangular part of the matrix including the nonzeros on the diagonal. (Input)

JCOL — Vector of length NZ containing the column subscripts of the nonzeros in the lower triangular part of the matrix including the nonzeros on the diagonal. (Input)
(IROW (K), JCOL(K)) gives the row and column indices of the *k*-th nonzero element of the matrix stored in coordinate form. Note, IROW(K) ≥ JCOL(K).

IJOB — Integer parameter selecting an ordering to permute the matrix symmetrically. (Input)
IJOB = 0 selects the user ordering specified in IPER and reorders it so that the multifrontal method can be used in the numerical factorization.
IJOB = 1 selects the user ordering specified in IPER.
IJOB = 2 selects a minimum degree ordering.
IJOB = 3 selects a minimum degree ordering suitable for the multifrontal method in the numerical factorization.

ITWKSP — The total workspace needed. (Input)
If the default is desired, set ITWKSP to zero. See Comment 1 for the default.

MAXSUB — Number of subscripts contained in array NZSUB. (Input/Output)
On input, MAXSUB gives the size of the array NZSUB.
Note that when default workspace (ITWKSP = 0) is used, set MAXSUB = 3 * NZ.
Otherwise (ITWKSP > 0), set MAXSUB = (ITWKSP – 10 * N – 7) / 4. On output, MAXSUB gives the number of subscripts used by the compressed subscript format.

NZSUB — Vector of length MAXSUB containing the row subscripts for the off-diagonal nonzeros in the Cholesky factor in compressed format. (Output)

INZSUB — Vector of length N + 1 containing pointers for NZSUB. The row subscripts for the off-diagonal nonzeros in column J are stored in NZSUB from location INZSUB (J) to INZSUB(J + 1) – 1. (Output)

MAXNZ — Total number of off-diagonal nonzeros in the Cholesky factor. (Output)

ILNZ — Vector of length N + 1 containing pointers to the Cholesky factor. The off-diagonal nonzeros in column J of the factor are stored from location ILNZ (J) to ILNZ(J + 1) – 1. (Output)
(ILNZ, NZSUB, INZSUB) sets up the data structure for the off-diagonal nonzeros of the Cholesky factor in column ordered form using compressed subscript format.

IPER — Vector of length N containing the ordering specified by IJOB. (Input/Output)
IPER (I) = K indicates that the original row K is the new row I.

INVPER — Vector of length N containing the inverse permutation. (Output)
INVPER (K) = I indicates that the original row K is the new row I.

ISPACE — The storage space needed for stack of frontal matrices.   (Output)

Comments

1. Automatic workspace usage is

   `LSCXD` 10`N` + 12`NZ` + 7 units. This is the default if `ITWKSP` is zero. If the value is positive, `ITWKSP` units are automatically allocated.

   Workspace may be explicitly provided, if desired, by use of `L2CXD`. The reference is

   ```
   CALL L2CXD (N, NZ, IROW, JCOL, IJOB, MAXSUB, NZSUB,
               INZSUB, MAXNZ, ILNZ, IPER, INVPER,
               ISPACE, LIWK, IWK)
   ```

   The additional arguments are as follows:

   LIWK — The length of `IWK`, `LIWK` should be at least 10`N` + 12`NZ` + 7. Note that the argument `MAXSUB` should be set to (`LIWK` − 10`N` − 7)/4.

   IWK — Integer work vector of length `LIWK`.

   Note that the parameter `ITWKSP` is not an argument to this routine.

2. Informational errors

Type	Code	
4	1	The matrix is structurally singular.

Algorithm

Consider the linear equation

$$Ax = b$$

where *A* is sparse, positive definite and symmetric. The sparse coordinate format for the matrix *A* requires one real and two integer vectors. The real array `a` contains all the nonzeros in the *lower triangle* of *A* including the diagonal. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column indices for these entries in *A*. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

$$\mathtt{irow}(i) \geq \mathtt{jcol}(i) \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in the lower triangle of *A* zero.

The routine `LSCXD` computes a minimum degree ordering or uses a user-supplied ordering to set up the sparse data structure for the Cholesky factor, *L*. Then the routine `LNFXD` (page 228) produces the numerical entries in *L* so that we have

$$P\,AP^T = LL^T$$

Here, *P* is the permutation matrix determined by the ordering.

The numerical computations can be carried out in one of two ways. The first method performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Example

As an example, the following matrix is symbolically factorized, and the result is printed:

$$A = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

The number of nonzeros in the lower triangle of *A* is `nz`= 10. The sparse coordinate form for the lower triangle of *A* is given by:

irow	1	2	3	3	4	4	5	5	5	5
jcol	1	2	1	3	3	4	1	2	4	5

or equivalently by

irow	4	5	5	5	1	2	3	3	4	5
jcol	4	1	2	4	1	2	1	3	3	5

```
      INTEGER    N, NZ
      PARAMETER  (N=5, NZ=10)
C
      INTEGER    IJOB, ILNZ(N+1), INVPER(N), INZSUB(N+1), IPER(N),
     &           IROW(NZ), ISPACE, ITWKSP, JCOL(NZ), MAXNZ, MAXSUB,
     &           NZSUB(3*NZ)
      EXTERNAL   LSCXD, WRIRN
C
      DATA IROW/1, 2, 3, 3, 4, 4, 5, 5, 5, 5/
      DATA JCOL/1, 2, 1, 3, 3, 4, 1, 2, 4, 5/
C                                  Select minimum degree ordering
C                                  for multifrontal method
      IJOB = 3
C                                  Use default workspace
      ITWKSP = 0
      MAXSUB = 3*NZ
      CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
     &            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
```

```
C                                  Print results
      CALL WRIRN (’ iper ’, 1, N, IPER, 1, 0)
      CALL WRIRN (’ invper ’, 1, N, INVPER, 1, 0)
      CALL WRIRN (’ nzsub ’, 1, MAXSUB, NZSUB, 1, 0)
      CALL WRIRN (’ inzsub ’, 1, N+1, INZSUB, 1, 0)
      CALL WRIRN (’ ilnz ’, 1, N+1, ILNZ, 1, 0)
      END
```

Output

```
          iper
1   2   3   4   5
2   1   5   4   3

        invper
1   2   3   4   5
2   1   5   4   3

      nzsub
1   2   3   4
3   5   4   5

          inzsub
1   2   3   4   5   6
1   1   3   4   4   4

           ilnz
1   2   3   4   5   6
1   2   4   6   7   7
```

LNFXD/DLNFXD (Single/Double precision)

Compute the numerical Cholesky factorization of a sparse symmetrical matrix *A*.

Usage

```
CALL LNFXD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB,
            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE,
            ITWKSP, DIAG, RLNZ, RPARAM)
```

Arguments

N — Number of equations. (Input)

NZ — The number of nonzero coefficients in the linear system. (Input)

A — Vector of length `NZ` containing the nonzero coefficients of the lower triangle of the linear system. (Input)

IROW — Vector of length `NZ` containing the row numbers of the corresponding elements in the lower triangle of `A`. (Input)

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in the lower triangle of `A`. (Input)

IJOB — Integer parameter selecting factorization method. (Input)
IJOB = 1 yields factorization in sparse column format.
IJOB = 2 yields factorization using multifrontal method.

MAXSUB — Number of subscripts contained in array NZSUB as output from subroutine LSCXD/DLSCXD. (Input)

NZSUB — Vector of length MAXSUB containing the row subscripts for the nonzeros in the Cholesky factor in compressed format as output from subroutine LSCXD/DLSCXD. (Input)

INZSUB — Vector of length N + 1 containing pointers for NZSUB as output from subroutine LSCXD/DLSCXD. (Input)
The row subscripts for the nonzeros in column J are stored from location INZSUB (J) to INZSUB(J + 1) − 1.

MAXNZ — Length of RLNZ as output from subroutine LSCXD/DLSCXD. (Input)

ILNZ — Vector of length N + 1 containing pointers to the Cholesky factor as output from subroutine LSCXD/DLSCXD. (Input)
The row subscripts for the nonzeros in column J of the factor are stored from location ILNZ(J) to ILNZ(J + 1) − 1. (ILNZ, NZSUB, INZSUB) sets up the compressed data structure in column ordered form for the Cholesky factor.

IPER — Vector of length N containing the permutation as output from subroutine LSCXD/DLSCXD. (Input)

INVPER — Vector of length N containing the inverse permutation as output from subroutine LSCXD/DLSCXD. (Input)

ISPACE — The storage space needed for the stack of frontal matrices as output from subroutine LSCXD/DLSCXD. (Input)

ITWKSP — The total workspace needed. (Input)
If the default is desired, set ITWKSP to zero. See Comment 1 for the default.

DIAG — Vector of length N containing the diagonal of the factor. (Output)

RLNZ — Vector of length MAXNZ containing the strictly lower triangle nonzeros of the Cholesky factor. (Output)

RPARAM — Parameter vector containing factorization information. (Output)
RPARAM(1) = smallest diagonal element.
RPARAM(2) = largest diagonal element.

Comments

1. Automatic workspace usage is

   LNFXD 3N + 3NZ units, or

   DLNFXD 4N + 6NZ units. This is the default if ITWKSP is zero. If the value is positive, ITWKSP units are automatically allocated.

Workspace may be explicitly provided by use of L2FXD/DL2FXD . The reference is

```
CALL L2FXD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB,
            NZSUB,INZSUB, MAXNZ, ILNZ, IPER, INVPER,
            ISPACE,DIAG, RLNZ, RPARAM, WK, LWK, IWK,
            LIWK)
```

The additional arguments are as follows:

WK — Real work vector of length LWK.

LWK — The length of WK, LWK should be at least N + 3NZ.

IWK — Integer work vector of length LIWK.

LIWK — The length of IWK, LIWK should be at least 2N.

Note that the parameter ITWKSP is not an argument to this routine.

2. Informational errors

Type	Code	
4	1	The coefficient matrix is not positive definite.
4	2	A column without nonzero elements has been found in the coefficient matrix.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is sparse, positive definite and symmetric. The sparse coordinate format for the matrix A requires one real and two integer vectors. The real array a contains all the nonzeros in the *lower triangle* of A including the diagonal. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column indices for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \dots, \mathtt{nz}$$

$$\mathtt{irow}(i) \ge \mathtt{jcol}(i) \qquad i = 1, \dots, \mathtt{nz}$$

with all other entries in the lower triangle of A zero. The routine LNFXD produces the Cholesky factorization of $P\,AP^T$given the symbolic factorization of A which is computed by LSCXD (page 224). That is, this routine computes L which satisfies

$$P\,AP^T = LL^T$$

The diagonal of L is stored in DIAG and the strictly lower triangular part of L is stored in compressed subscript form in R = RLNZ as follows. The nonzeros in the j-th column of L are stored in locations $R(i), \dots, R(i + k)$ where i = ILNZ(j) and k = ILNZ(j + 1) − ILNZ(j) − 1. The row subscripts are stored in the vector NZSUB from locations INZSUB(j) to INZSUB(j + 1) − 1.

The numerical computations can be carried out in one of two ways. The first method (when `IJOB` = 2) performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method (when `IJOB` = 1) is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Example

As an example, consider the 5 × 5 linear system:

$$A = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

The number of nonzeros in the lower triangle of *A* is `nz` = 10. The sparse coordinate form for the lower triangle of *A* is given by:

irow	1	2	3	3	4	4	5	5	5	5
jcol	1	2	1	3	3	4	1	2	4	5
a	10	20	1	30	4	40	2	3	5	50

or equivalently by

irow	4	5	5	5	1	2	3	3	4	5
jcol	4	1	2	4	1	2	1	3	3	5
a	40	2	3	5	10	20	1	30	4	50

We first call `LSCXD`, page 224, to produce the symbolic information needed to pass on to `LNFXD`. Then call `LNFXD` to factor this matrix. The results are displayed below.

```
      INTEGER    N, NZ, NRLNZ
      PARAMETER  (N=5, NZ=10, NRLNZ=10)
C
      INTEGER    IJOB, ILNZ(N+1), INVPER(N), INZSUB(N+1), IPER(N),
     &           IROW(NZ), ISPACE, ITWKSP, JCOL(NZ), MAXNZ, MAXSUB,
     &           NZSUB(3*NZ)
      REAL       A(NZ), DIAG(N), RLNZ(NRLNZ), RPARAM(2)
      EXTERNAL   LNFXD, LSCXD, WRRRN
C
      DATA A/10., 20., 1., 30., 4., 40., 2., 3., 5., 50./
      DATA IROW/1, 2, 3, 3, 4, 4, 5, 5, 5, 5/
      DATA JCOL/1, 2, 1, 3, 3, 4, 1, 2, 4, 5/
```

```
C                                  Select minimum degree ordering
C                                  for multifrontal method
      IJOB = 3
C                                  Use default workspace
      ITWKSP = 0
      MAXSUB = 3*NZ
      CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
     &            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
C                                  Check if NRLNZ is large enough
      IF (NRLNZ .GE. MAXNZ) THEN
C                                  Choose multifrontal method
         IJOB = 2
         CALL LNFXD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB, INZSUB,
     &               MAXNZ, ILNZ, IPER, INVPER, ISPACE, ITWKSP, DIAG,
     &               RLNZ, RPARAM)
C                                  Print results
         CALL WRRRN (’ diag ’, 1, N, DIAG, 1, 0)
         CALL WRRRN (’ rlnz ’, 1, MAXNZ, RLNZ, 1, 0)
      END IF
C
      END
```

Output

```
             diag
    1       2       3       4       5
4.472   3.162   7.011   6.284   5.430

                          rlnz
     1        2        3        4        5        6
0.6708   0.6325   0.3162   0.7132  -0.0285   0.6398
```

LFSXD/DLFSXD (Single/Double precision)

Solve a real sparse symmetric positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.

Usage

```
CALL LFSXD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ,
            DIAG, IPER, B, X)
```

Arguments

N — Number of equations.   (Input)

MAXSUB — Number of subscripts contained in array `NZSUB` as output from subroutine `LSCXD`/`DLSCXD`.   (Input)

NZSUB — Vector of length `MAXSUB` containing the row subscripts for the off-diagonal nonzeros in the factor as output from subroutine `LSCXD`/`DLSCXD`. (Input)

INZSUB — Vector of length `N` + 1 containing pointers for `NZSUB` as output from subroutine `LSCXD`/`DLSCXD`.   (Input)

The row subscripts of column J are stored from location INZSUB(J) to INZSUB(J + 1) − 1.

MAXNZ — Total number of off-diagonal nonzeros in the Cholesky factor as output from subroutine LSCXD/DLSCXD. (Input)

RLNZ — Vector of length MAXNZ containing the off-diagonal nonzeros in the factor in column ordered format as output from subroutine LNFXD/DLNFXD. (Input)

ILNZ — Vector of length N + 1 containing pointers to RLNZ as output from subroutine LSCXD/DLSCXD. The nonzeros in column J of the factor are stored from location ILNZ(J) to ILNZ(J + 1) − 1. (Input)
The values (RLNZ, ILNZ, NZSUB, INZSUB) give the off-diagonal nonzeros of the factor in a compressed subscript data format.

DIAG — Vector of length N containing the diagonals of the Cholesky factor as output from subroutine LNFXD/DLNFXD. (Input)

IPER — Vector of length N containing the ordering as output from subroutine LSCXD/DLSCXD. (Input)
IPER(I) = K indicates that the original row K is the new row I.

B — Vector of length N containing the right-hand side. (Input)

X — Vector of length N containing the solution. (Output)

Comments

Informational error

Type	Code	
4	1	The input matrix is numerically singular.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is sparse, positive definite and symmetric. The sparse coordinate format for the matrix A requires one real and two integer vectors. The real array a contains all the nonzeros in the *lower triangle* of A including the diagonal. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column indices for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

$$\mathtt{irow}(i) \ge \mathtt{jcol}(i) \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in the lower triangle of A zero.

The routine LFSXD computes the solution of the linear system given its Cholesky factorization. The factorization is performed by calling LSCXD (page 224) followed by LNFXD (page 228). The routine LSCXD computes a minimum degree ordering or uses a user-supplied ordering to set up the sparse data structure for the Cholesky factor, *L*. Then the routine `LNFXD` produces the numerical entries in *L* so that we have

$$P\,AP^{T} = LL^{T}$$

Here *P* is the permutation matrix determined by the ordering.

The numerical computations can be carried out in one of two ways. The first method performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Finally, the solution *x* is obtained by the following calculations:

$$1)\ Ly_1 = Pb$$

$$2)\ L^{T}y_2 = y_1$$

$$3)\ x = P^{T}y_2$$

Example

As an example, consider the 5 × 5 linear system:

$$A = \begin{bmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{bmatrix}$$

Let

$$x_1^T = (1, 2, 3, 4, 5)$$

so that Ax_1 = (23, 55, 107, 197, 278) ^{T}, and

$$x_2^T = (5, 4, 3, 2, 1)$$

so that Ax_2 = (55, 83, 103, 97, 82) ^{T}. The number of nonzeros in the lower triangle of *A* is `nz` = 10. The sparse coordinate form for the lower triangle of *A* is given by:

irow	1	2	3	3	4	4	5	5	5	5
jcol	1	2	1	3	3	4	1	2	4	5
a	10	20	1	30	4	40	2	3	5	50

or equivalently by

irow	4	5	5	5	1	2	3	3	4	5
jcol	4	1	2	4	1	2	1	3	3	5
a	40	2	3	5	10	20	1	30	4	50

```
      INTEGER    N, NZ, NRLNZ
      PARAMETER  (N=5, NZ=10, NRLNZ=10)
C
      INTEGER    IJOB, ILNZ(N+1), INVPER(N), INZSUB(N+1), IPER(N),
     &           IROW(NZ), ISPACE, ITWKSP, JCOL(NZ), MAXNZ, MAXSUB,
     &           NZSUB(3*NZ)
      REAL       A(NZ), B1(N), B2(N), DIAG(N), RLNZ(NRLNZ), RPARAM(2),
     &           X(N)
      EXTERNAL   LFSXD, LNFXD, LSCXD, WRRRN
C
      DATA A/10., 20., 1., 30., 4., 40., 2., 3., 5., 50./
      DATA B1/23., 55., 107., 197., 278./
      DATA B2/55., 83., 103., 97., 82./
      DATA IROW/1, 2, 3, 3, 4, 4, 5, 5, 5, 5/
      DATA JCOL/1, 2, 1, 3, 3, 4, 1, 2, 4, 5/
C                                 Select minimum degree ordering
C                                 for multifrontal method
      IJOB = 3
C                                 Use default workspace
      ITWKSP = 0
      MAXSUB = 3*NZ
      CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
     &            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
C                                 Check if NRLNZ is large enough
      IF (NRLNZ .GE. MAXNZ) THEN
C                                 Choose multifrontal method
         IJOB = 2
         CALL LNFXD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB, INZSUB,
     &               MAXNZ, ILNZ, IPER, INVPER, ISPACE, ITWKSP, DIAG,
     &               RLNZ, RPARAM)
C                                 Solve A * X1 = B1
         CALL LFSXD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ, DIAG,
     &               IPER, B1, X)
C                                 Print X1
         CALL WRRRN (’ x1 ’, 1, N, X, 1, 0)
C                                 Solve A * X2 = B2
         CALL LFSXD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ, DIAG,
     &               IPER, B2, X)
C                                 Print X2
         CALL WRRRN (’ x2 ’, 1, N, X, 1, 0)
      END IF
C
      END
```

Output

```
                        x1
    1        2        3        4        5
1.000    2.000    3.000    4.000    5.000

                        x2
    1        2        3        4        5
5.000    4.000    3.000    2.000    1.000
```

LSLZD/DLSLZD (Single/Double precision)

Solve a complex sparse Hermitian positive definite system of linear equations by Gaussian elimination.

Usage

`CALL LSLZD (N, NZ, A, IROW, JCOL, B, ITWKSP, X)`

Arguments

N — Number of equations.   (Input)

NZ — The number of nonzero coefficients in the lower triangle of the linear system.   (Input)

A — Complex vector of length `NZ` containing the nonzero coefficients in the lower triangle of the linear system.   (Input)
The sparse matrix has nonzeroes only in entries (`IROW` (*i*), `JCOL`(*i*)) for *i* = 1 to `NZ`, and at this location the sparse matrix has value `A`(*i*).

IROW — Vector of length `NZ` containing the row numbers of the corresponding elements in the lower triangle of `A`.   (Input)
Note `IROW`(*i*) ≥ `JCOL`(*i*), since we are only indexing the lower triangle.

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in the lower triangle of `A`.   (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system.   (Input)

ITWKSP — The total workspace needed.   (Input)
If the default is desired, set `ITWKSP` to zero. See Comment 1 for the default.

X — Complex vector of length `N` containing the solution to the linear system.   (Output)

Comments

1. Automatic workspace usage is

   `LSLZD` 20`N` + 27`NZ` + 9 units, or

   `DLSLZD` 24`N` + 39`NZ` + 9 units. This is the default if `ITWKSP` is zero. If this value is positive, `ITWKSP` units are automatically allocated.

   Workspace may be explicitly provided, if desired, by use of `L2LZD`/`DL2LZD`. The reference is

   ```
   CALL L2LZD (N, NZ, A, IROW, JCOL, B, X, IPER,
               IPARAM, RPARAM, WK, LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   IPER — Vector of length `N` containing the ordering.

   IPARAM — Integer vector of length 4. See Comment 3.

   RPARAM — Real vector of length 2. See Comment 3.

   WK — Complex work vector of length `LWK`.

   LWK — The length of `WK`, `LWK` should be at least 2`N` + 6`NZ`.

   IWK — Integer work vector of length `LIWK`.

   LIWK — The length of `IWK`, `LIWK` should be at least 15`N` + 15`NZ` + 9.

   Note that the parameter `ITWKSP` is not an argument for this routine.

2. Informational errors

Type	Code	
4	1	The coefficient matrix is not positive definite.
4	2	A column without nonzero elements has been found in the coefficient matrix.

3. If the default parameters are desired for `L2LZD`, then set `IPARAM`(1) to zero and call the routine `L2LZD`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `L2LZD`.

   `CALL L4LZD (IPARAM, RPARAM)`
   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

   Note that the call to `L4LZD` will set `IPARAM` and `RPARAM` to their default values, so only nondefault values need to be set above. The arguments are as follows:

   IPARAM — Integer vector of length 4.
   `IPARAM`(1) = Initialization flag.

IPARAM(2) = The numerical factorization method.

IPARAM(2)	**Action**
0	Multifrontal
1	Sparse column

Default: 0.

IPARAM(3) = The ordering option.

IPARAM(3)	**Action**
0	Minimum degree ordering
1	User's ordering specified in IPER

Default: 0.

IPARAM(4) = The total number of nonzeros in the factorization matrix.

RPARAM — Real vector of length 2.
RPARAM(1) = The absolute value of the largest diagonal element in the Cholesky factorization.
RPARAM(2) = The absolute value of the smallest diagonal element in the Cholesky factorization.

If double precision is required, then DL4LZD is called and RPARAM is declared double precision.

Algorithm

Consider the linear equation

$$Ax = b$$

where *A* is sparse, positive definite and Hermitian. The sparse coordinate format for the matrix *A* requires one complex and two integer vectors. The complex array a contains all the nonzeros in the lower triangle of *A* including the diagonal. Let the number of nonzeros be nz. The two integer arrays irow and jcol, each of length nz, contain the row and column indices for these entries in *A*. That is

$$A_{\mathtt{irow}(i),\mathtt{jcol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

$$\mathtt{irow}(i) \geq \mathtt{jcol}(i) \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in the lower triangle of *A* zero.

The routine LSLZD solves a system of linear algebraic equations having a complex, sparse, Hermitian and positive definite coefficient matrix. It first uses the routine LSCXD (page 224) to compute a symbolic factorization of a permutation of the coefficient matrix. It then calls LNFZD (page 240) to perform the numerical factorization. The solution of the linear system is then found using LFSZD (page 244).

The routine LSCXD computes a minimum degree ordering or uses a user-supplied ordering to set up the sparse data structure for the Cholesky factor, *L*. Then the routine LNFZD produces the numerical entries in *L* so that we have

$$P\,AP^T = LL^H$$

Here P is the permutation matrix determined by the ordering.

The numerical computations can be carried out in one of two ways. The first method performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Finally, the solution *x* is obtained by the following calculations:

$$1)\quad Ly_1 = Pb$$

$$2)\quad L^H y_2 = y_1$$

$$3)\quad x = P^T y_2$$

The routine `LFSZD` accepts *b* and the permutation vector which determines *P* . It then returns *x*.

Example

As an example, consider the 3 × 3 linear system:

$$A = \begin{bmatrix} 2+0i & -1+i & 0 \\ -1-i & 4+0i & 1+2i \\ 0 & 1-2i & 10+0i \end{bmatrix}$$

Let x^T = (1 + *i*, 2 + 2*i*, 3 + 3*i*) so that *Ax* = (–2 + 2*i*, 5 + 15*i*, 36 + 28*i*)$^{T.}$ The number of nonzeros in the lower triangle of *A* is `nz` = 5. The sparse coordinate form for the lower triangle of *A* is given by:

irow	1	2	3	2	3
jcol	1	2	3	1	2
a	$2+0i$	$4+0i$	$10+0i$	$-1-i$	$1-2i$

or equivalently by

irow	3	2	3	1	2
jcol	3	1	2	1	2
a	$10+0i$	$-1-i$	$1-2i$	$2+0i$	$4+0i$

```
      INTEGER    N, NZ
      PARAMETER  (N=3, NZ=5)
C
```

```
      INTEGER    IROW(NZ), JCOL(NZ), ITWKSP
      COMPLEX    A(NZ), B(N), X(N)
      EXTERNAL   LSLZD, WRCRN
C
      DATA A/(2.0,0.0), (4.0,0.0), (10.0,0.0), (-1.0,-1.0), (1.0,-2.0)/
      DATA B/(-2.0,2.0), (5.0,15.0), (36.0,28.0)/
      DATA IROW/1, 2, 3, 2, 3/
      DATA JCOL/1, 2, 3, 1, 2/
C                                Use default workspace
      ITWKSP = 0
C                                Solve A * X = B
      CALL LSLZD (N, NZ, A, IROW, JCOL, B, ITWKSP, X)
C                                Print results
      CALL WRCRN (’ x ’, 1, N, X, 1, 0)
      END
```

Output

```
                          x
              1                 2                 3
( 1.000, 1.000)  ( 2.000, 2.000)  ( 3.000, 3.000)
```

LNFZD/DLNFZD (Single/Double precision)

Compute the numerical Cholesky factorization of a sparse Hermitian matrix *A*.

Usage

```
CALL LNFZD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB,
            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE,
            ITWKSP, DIAG, RLNZ, RPARAM)
```

Arguments

N — Number of equations. (Input)

NZ — The number of nonzero coefficients in the linear system. (Input)

A — Complex vector of length NZ containing the nonzero coefficients of the lower triangle of the linear system. (Input)

IROW — Vector of length NZ containing the row numbers of the corresponding elements in the lower triangle of A. (Input)

JCOL — Vector of length NZ containing the column numbers of the corresponding elements in the lower triangle of A. (Input)

IJOB — Integer parameter selecting factorization method. (Input)
IJOB = 1 yields factorization in sparse column format.
IJOB = 2 yields factorization using multifrontal method.

MAXSUB — Number of subscripts contained in array NZSUB as output from subroutine LSCXD/DLSCXD. (Input)

NZSUB — Vector of length MAXSUB containing the row subscripts for the nonzeros in the Cholesky factor in compressed format as output from subroutine LSCXD/DLSCXD. (Input)

INZSUB — Vector of length N + 1 containing pointers for NZSUB as output from subroutine LSCXD/DLSCXD. (Input)
The row subscripts for the nonzeros in column J are stored from location INZSUB(J) to INZSUB(J + 1) − 1.

MAXNZ — Length of RLNZ as output from subroutine LSCXD/DLSCXD. (Input)

ILNZ — Vector of length N + 1 containing pointers to the Cholesky factor as output from subroutine LSCXD/DLSCXD. (Input)
The row subscripts for the nonzeros in column J of the factor are stored from location ILNZ(J) to ILNZ(J + 1) − 1.
(ILNZ , NZSUB, INZSUB) sets up the compressed data structure in column ordered form for the Cholesky factor.

IPER — Vector of length N containing the permutation as output from subroutine LSCXD/DLSCXD. (Input)

INVPER — Vector of length N containing the inverse permutation as output from subroutine LSCXD/DLSCXD. (Input)

ISPACE — The storage space needed for the stack of frontal matrices as output from subroutine LSCXD/DLSCXD. (Input)

ITWKSP — The total workspace needed. (Input)
If the default is desired, set ITWKSP to zero. See Comment 1 for the default.

DIAG — Complex vector of length N containing the diagonal of the factor. (Output)

RLNZ — Complex vector of length MAXNZ containing the strictly lower triangle nonzeros of the Cholesky factor. (Output)

RPARAM — Parameter vector containing factorization information. (Output)
RPARAM (1) = smallest diagonal element in absolute value.
RPARAM (2) = largest diagonal element in absolute value.

Comments

1. Automatic workspace usage is

   LNFZD 4N + 6NZ units, or

   DLNFZD 6N + 12NZ units. This is the default if ITWKSP is zero. If the value is positive, ITWKSP units are automatically allocated.

   Workspace may be explicitly provided by use of L2FZD/DL2FZD. The reference is

```
CALL L2FZD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB,
            NZSUB, INZSUB, MAXNZ, ILNZ, IPER,
            INVPER, ISPACE, DIAG, RLNZ, RPARAM, WK,
            LWK, IWK, LIWK)
```

The additional arguments are as follows:

WK — Complex work vector of length `LWK`.

LWK — The length of `WK`, `LWK` should be at least `N` + 3`NZ`.

IWK — Integer work vector of length `LIWK`.

LIWK — The length of `IWK`, `LIWK` should be at least 2`N`.

Note that the parameter `ITWKSP` is not an argument to this routine.

2. Informational errors

Type	Code	
4	1	The coefficient matrix is not positive definite.
4	2	A column without nonzero elements has been found in the coefficient matrix.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is sparse, positive definite and Hermitian. The sparse coordinate format for the matrix A requires one complex and two integer vectors. The complex array `a` contains all the nonzeros in the *lower triangle* of A including the diagonal. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column indices for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

$$\mathtt{irow}(i) \ge \mathtt{jcol}(i) \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in the lower triangle of A zero.

The routine `LNFZD` produces the Cholesky factorization of $P\,AP^T$ given the symbolic factorization of A which is computed by `LSCXD` (page 224). That is, this routine computes L which satisfies

$$P\,AP^T = LL^H$$

The diagonal of L is stored in `DIAG` and the strictly lower triangular part of L is stored in compressed subscript form in R = `RLNZ` as follows. The nonzeros in the jth column of L are stored in locations $R(i), \ldots, R(i + k)$ where i = `ILNZ`(j) and k = `ILNZ`(j + 1) − `ILNZ`(j) − 1. The row subscripts are stored in the vector `NZSUB` from locations `INZSUB`(j) to `INZSUB`(j + 1) − 1.

The numerical computations can be carried out in one of two ways. The first method (when `IJOB` = 2) performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method (when `IJOB` = 1) is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme.

Example

As an example, consider the 3 × 3 linear system:

$$A = \begin{bmatrix} 2+0i & -1+i & 0 \\ -1-i & 4+0i & 1+2i \\ 0 & 1-2i & 10+0i \end{bmatrix}$$

The number of nonzeros in the lower triangle of *A* is `nz` = 5. The sparse coordinate form for the lower triangle of *A* is given by:

$$\begin{array}{rrrrrr} \text{irow} & 1 & 2 & 3 & 2 & 3 \\ \text{jcol} & 1 & 2 & 3 & 1 & 2 \\ \text{a} & 2+0i & 4+0i & 10+0i & -1-i & 1-2i \end{array}$$

or equivalently by

$$\begin{array}{rrrrrr} \text{irow} & 3 & 2 & 3 & 1 & 2 \\ \text{jcol} & 3 & 1 & 2 & 1 & 2 \\ \text{a} & 10+0i & -1-i & 1-2i & 2+0i & 4+0i \end{array}$$

We first call `LSCXD` to produce the symbolic information needed to pass on to `LNFZD`. Then call `LNFZD` to factor this matrix. The results are displayed below.

```
      INTEGER    N, NZ, NRLNZ
      PARAMETER  (N=3, NZ=5, NRLNZ=5)
C
      INTEGER    IJOB, ILNZ(N+1), INVPER(N), INZSUB(N+1), IPER(N),
     &           IROW(NZ), ISPACE, ITWKSP, JCOL(NZ), MAXNZ, MAXSUB,
     &           NZSUB(3*NZ)
      REAL       RPARAM(2)
      COMPLEX    A(NZ), DIAG(N), RLNZ(NRLNZ)
      EXTERNAL   LNFZD, LSCXD, WRCRN
C
      DATA A/(2.0,0.0), (4.0,0.0), (10.0,0.0), (-1.0,-1.0), (1.0,-2.0)/
      DATA IROW/1, 2, 3, 2, 3/
      DATA JCOL/1, 2, 3, 1, 2/
C                                 Select minimum degree ordering
C                                 for multifrontal method
      IJOB = 3
C                                 Use default workspace
      ITWKSP = 0
      MAXSUB = 3*NZ
      CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
     &            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
C                                  Check if NRLNZ is large enough
      IF (NRLNZ .GE. MAXNZ) THEN
C                                  Choose multifrontal method
         IJOB = 2
         CALL LNFZD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB, INZSUB,
     &               MAXNZ, ILNZ, IPER, INVPER, ISPACE, ITWKSP, DIAG,
     &               RLNZ, RPARAM)
C                                  Print results
         CALL WRCRN (’ diag ’, 1, N, DIAG, 1, 0)
         CALL WRCRN (’ rlnz ’, 1, MAXNZ, RLNZ, 1, 0)
      END IF
C
      END
```

Output

```
                         diag
              1                 2                 3
( 1.414, 0.000)  ( 1.732, 0.000)  ( 2.887, 0.000)

                rlnz
              1                 2
(-0.707,-0.707)  ( 0.577,-1.155)
```

LFSZD/DLFSZD (Single/Double precision)

Solve a complex sparse Hermitian positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.

Usage

```
CALL LFSZD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ,
            DIAG,IPER, B, X)
```

Arguments

N — Number of equations.   (Input)

MAXSUB — Number of subscripts contained in array `NZSUB` as output from subroutine `LSCXD`/`DLSCXD`.   (Input)

NZSUB — Vector of length `MAXSUB` containing the row subscripts for the off-diagonal nonzeros in the factor as output from subroutine `LSCXD`/`DLSCXD`. (Input)

INZSUB — Vector of length `N` + 1 containing pointers for `NZSUB` as output from subroutine `LSCXD`/`DLSCXD`.   (Input)
The row subscripts of column `J` are stored from location `INZSUB`(`J`) to `INZSUB` (`J` + 1) – 1.

MAXNZ — Total number of off-diagonal nonzeros in the Cholesky factor as output from subroutine `LSCXD`/`DLSCXD`. (Input)

RLNZ — Complex vector of length `MAXNZ` containing the off-diagonal nonzeros in the factor in column ordered format as output from subroutine `LNFZD`/`DLNFZD`. (Input)

ILNZ — Vector of length `N` +1 containing pointers to `RLNZ` as output from subroutine `LSCXD`/`DLSCXD`. The nonzeros in column `J` of the factor are stored from location `ILNZ`(`J`) to `ILNZ`(`J` + 1) – 1. (Input)
The values (`RLNZ`, `ILNZ`, `NZSUB`, `INZSUB`) give the off-diagonal nonzeros of the factor in a compressed subscript data format.

DIAG — Complex vector of length `N` containing the diagonals of the Cholesky factor as output from subroutine `LNFZD`/`DLNFZD`. (Input)

IPER — Vector of length `N` containing the ordering as output from subroutine `LSCXD`/`DLSCXD`. (Input)
`IPER`(`I`) = `K` indicates that the original row `K` is the new row `I`.

B — Complex vector of length `N` containing the right-hand side. (Input)

X — Complex vector of length `N` containing the solution. (Output)

Comments

Informational error

Type	Code	
4	1	The input matrix is numerically singular.

Algorithm

Consider the linear equation

$$Ax = b$$

where A is sparse, positive definite and Hermitian. The sparse coordinate format for the matrix A requires one complex and two integer vectors. The complex array `a` contains all the nonzeros in the *lower triangle* of A including the diagonal. Let the number of nonzeros be `nz`. The two integer arrays `irow` and `jcol`, each of length `nz`, contain the row and column indices for these entries in A. That is

$$A_{\mathtt{irow}(i),\mathtt{icol}(i)} = \mathtt{a}(i), \qquad i = 1, \ldots, \mathtt{nz}$$

$$\mathtt{irow}(i) \ge \mathtt{jcol}(i) \qquad i = 1, \ldots, \mathtt{nz}$$

with all other entries in the lower triangle of A zero.

The routine `LFSZD` computes the solution of the linear system given its Cholesky factorization. The factorization is performed by calling `LSCXD` (page 224) followed by `LNFZD` (page 240). The routine `LSCXD` computes a minimum degree ordering or uses a user-supplied ordering to set up the sparse data structure for the Cholesky factor, L. Then the routine `LNFZD` produces the numerical entries in L so that we have

$$P\,AP^T = LL^H$$

Here P is the permutation matrix determined by the ordering.

The numerical computations can be carried out in one of two ways. The first method performs the factorization using a multifrontal technique. This option requires more storage but in certain cases will be faster. The multifrontal method is based on the routines in Liu (1987). For detailed description of this method, see Liu (1990), also Duff and Reid (1983, 1984), Ashcraft (1987), Ashcraft et al. (1987), and Liu (1986, 1989). The second method is fully described in George and Liu (1981). This is just the standard factorization method based on the sparse compressed storage scheme. Finally, the solution x is obtained by the following calculations:

$$1)\ Ly_1 = Pb$$

$$2)\ L^H y_2 = y_1$$

$$3)\ x = P^T y_2$$

Example

As an example, consider the 3 × 3 linear system:

$$A = \begin{bmatrix} 2+0i & -1+i & 0 \\ -1-i & 4+0i & 1+2i \\ 0 & 1-2i & 10+0i \end{bmatrix}$$

Let

$$x_1^T = (1+i, 2+2i, 3+3i)$$

so that $Ax_1 = (-2 + 2i, 5 + 15i, 36 + 28i)^T$, and

$$x_2^T = (3+3i, 2+2i, 1+1i)$$

so that $Ax_2 = (2 + 6i, 7 - 5i, 16 + 8i)^T$. The number of nonzeros in the lower triangle of A is `nz` = 5. The sparse coordinate form for the lower triangle of A is given by:

irow	1	2	3	2	3
jcol	1	2	3	1	2
a	$2+0i$	$4+0i$	$10+0i$	$-1-i$	$1-2i$

or equivalently by

irow	3	2	3	1	2
jcol	3	1	2	1	2
a	$10+0i$	$-1-i$	$1-2i$	$2+0i$	$4+0i$

```
      INTEGER    N, NZ, NRLNZ
      PARAMETER  (N=3, NZ=5, NRLNZ=5)
C
      INTEGER    IJOB, ILNZ(N+1), INVPER(N), INZSUB(N+1), IPER(N),
     &           IROW(NZ), ISPACE, ITWKSP, JCOL(NZ), MAXNZ, MAXSUB,
     &           NZSUB(3*NZ)
      COMPLEX    A(NZ), B1(N), B2(N), DIAG(N), RLNZ(NRLNZ), X(N)
      REAL       RPARAM(2)
      EXTERNAL   LFSZD, LNFZD, LSCXD, WRCRN
C
      DATA A/(2.0,0.0), (4.0,0.0), (10.0,0.0), (-1.0,-1.0), (1.0,-2.0)/
      DATA B1/(-2.0,2.0), (5.0,15.0), (36.0,28.0)/
      DATA B2/(2.0,6.0), (7.0,5.0), (16.0,8.0)/
      DATA IROW/1, 2, 3, 2, 3/
      DATA JCOL/1, 2, 3, 1, 2/
C                                  Select minimum degree ordering
C                                  for multifrontal method
      IJOB = 3
C                                  Use default workspace
      ITWKSP = 0
      MAXSUB = 3*NZ
      CALL LSCXD (N, NZ, IROW, JCOL, IJOB, ITWKSP, MAXSUB, NZSUB,
     &            INZSUB, MAXNZ, ILNZ, IPER, INVPER, ISPACE)
C                                  Check if NRLNZ is large enough
      IF (NRLNZ .GE. MAXNZ) THEN
C                                  Choose multifrontal method
         IJOB = 2
         CALL LNFZD (N, NZ, A, IROW, JCOL, IJOB, MAXSUB, NZSUB, INZSUB,
     &               MAXNZ, ILNZ, IPER, INVPER, ISPACE, ITWKSP, DIAG,
     &               RLNZ, RPARAM)
C                                  Solve A * X1 = B1
         CALL LFSZD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ, DIAG,
     &               IPER, B1, X)
C                                  Print X1
         CALL WRCRN (’ x1 ’, 1, N, X, 1, 0)
C                                  Solve A * X2 = B2
         CALL LFSZD (N, MAXSUB, NZSUB, INZSUB, MAXNZ, RLNZ, ILNZ, DIAG,
     &            IPER, B2, X)
C                                  Print X2
         CALL WRCRN (’ x2 ’, 1, N, X, 1, 0)
      END IF
C
      END
```

Output

```
                          x1
              1                2                3
( 1.000, 1.000)  ( 2.000, 2.000)  ( 3.000, 3.000)

                         x2
              1                2                3
( 3.000, 3.000)  ( 2.000, 2.000)  ( 1.000, 1.000)
```

LSLTO/DLSLTO (Single/Double precision)

Solve a real Toeplitz linear system.

Usage

```
CALL LSLTO (N, A, B, IPATH, X)
```

Arguments

N — Order of the matrix represented by `A`. (Input)

A — Real vector of length 2`N` – 1 containing the first row of the coefficient matrix followed by its first column beginning with the second element. (Input) See Comment 2.

B — Real vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Integer flag. (Input)
`IPATH` = 1 means the system $Ax = B$ is solved.
`IPATH` = 2 means the system $A^T x = B$ is solved.

X — Real vector of length `N` containing the solution of the linear system. (Output)
If `B` is not needed then `B` and `X` may share the same storage locations.

Comments

1. Automatic workspace usage is

   `LSLTO` 2`N` – 2 units, or
   `DLSLTO` 4`N` – 4 units.

   Workspace may be explicitly provided, if desired, by use of `L2LTO`/`DL2LTO`. The reference is

   ```
   CALL L2LTO (N, A, B, IPATH, X, WK)
   ```

   The additional argument is

   WK — Work vector of length 2`N` – 2.

2. Because of the special structure of Toeplitz matrices, the first row and the first column of a Toeplitz matrix completely characterize the matrix. Hence, only the elements `A`(1, 1), …, `A`(1, `N`), `A`(2, 1), …, `A`(`N`, 1) need to be stored.

Algorithm

Toeplitz matrices have entries that are constant along each diagonal, for example,

$$A = \begin{bmatrix} p_0 & p_1 & p_2 & p_4 \\ p_{-1} & p_0 & p_1 & p_2 \\ p_{-2} & p_{-1} & p_0 & p_1 \\ p_{-3} & p_{-2} & p_{-1} & p_0 \end{bmatrix}$$

The routine `LSLTO` is based on the routine `TSLS` in the TOEPLITZ package, see Arushanian et al. (1983). It is based on an algorithm of Trench (1964). This algorithm is also described by Golub and van Loan (1983), pages 125−133.

Example

A system of four linear equations is solved. Note that only the first row and column of the matrix *A* are entered.

```
C                                 Declare variables
      INTEGER    IPATH, N
      PARAMETER  (N=4)
      REAL       A(2*N-1), B(N), X(N)
C                                 Set values for  A, and B
C
C                                 A = (  2   -3   -1    6  )
C                                     (  1    2   -3   -1  )
C                                     (  4    1    2   -3  )
C                                     (  3    4    1    2  )
C
C                                 B = ( 16  -29   -7    5  )
C
      DATA A/2.0, -3.0, -1.0, 6.0, 1.0, 4.0, 3.0/
      DATA B/16.0, -29.0, -7.0, 5.0/
C                                 Solve AX = B
      IPATH = 1
      CALL LSLTO (N, A, B, IPATH, X)
C                                 Print results
      CALL WRRRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
                 X
     1        2        3        4
-2.000   -1.000    7.000    4.000
```

LSLTC/DLSLTC (Single/Double precision)

Solve a complex Toeplitz linear system.

Usage

`CALL LSLTC (N, A, B, IPATH, X)`

Arguments

N — Order of the matrix represented by `A`.   (Input)

A — Complex vector of length 2N − 1 containing the first row of the coefficient matrix followed by its first column beginning with the second element. (Input) See Comment 2.

B — Complex vector of length N containing the right-hand side of the linear system. (Input)

IPATH — Integer flag. (Input)
IPATH = 1 means the system $Ax = B$ is solved.
IPATH = 2 means the system $A^T x = B$ is solved.

X — Complex vector of length N containing the solution of the linear system. (Output)

Comments

1. Automatic workspace usage is

   LSLTC 4N − 4 units, or
   DLSLTC 8N − 8 units.

   Workspace may be explicitly provided, if desired, by use of L2LTC/DL2LTC. The reference is

   ```
   CALL L2LTC (N, A, B, IPATH, X, WK)
   ```

   The additional argument is

   WK — Complex work vector of length 2N − 2.

2. Because of the special structure of Toeplitz matrices, the first row and the first column of a Toeplitz matrix completely characterize the matrix. Hence, only the elements A(1, 1), …, A(1, N), A(2, 1), …, A(N, 1) need to be stored.

Algorithm

Toeplitz matrices have entries which are constant along each diagonal, for example,

$$A = \begin{bmatrix} p_0 & p_1 & p_2 & p_4 \\ p_{-1} & p_0 & p_1 & p_2 \\ p_{-2} & p_{-1} & p_0 & p_1 \\ p_{-3} & p_{-2} & p_{-1} & p_0 \end{bmatrix}$$

The routine LSLTC is based on the routine TSLC in the TOEPLITZ package, see Arushanian et al. (1983). It is based on an algorithm of Trench (1964). This algorithm is also described by Golub and van Loan (1983), pages 125−133.

Example

A system of four complex linear equations is solved. Note that only the first row and column of the matrix *A* are entered.

```
C                                 Declare variables
      PARAMETER  (N=4)
      COMPLEX    A(2*N-1), B(N), X(N)
C                                 Set values for  A and B
C
C                                 A = ( 2+2i    -3     1+4i   6-2i )
C                                     (  i      2+2i   -3     1+4i )
C                                     ( 4+2i     i     2+2i   -3   )
C                                     ( 3-4i    4+2i    i     2+2i )
C
C                                 B = ( 6+65i  -29-16i  7+i  -10+i )
C
      DATA A/(2.0,2.0), (-3.0,0.0), (1.0,4.0), (6.0,-2.0), (0.0,1.0),
     &       (4.0,2.0), (3.0,-4.0)/
      DATA B/(6.0,65.0), (-29.0,-16.0), (7.0,1.0), (-10.0,1.0)/
C                                 Solve AX = B
      IPATH = 1
      CALL LSLTC (N, A, B, IPATH, X)
C                                 Print results
      CALL WRCRN ('X', 1, N, X, 1, 0)
      END
```

Output

```
                                   X
              1                2                3                4
(-2.000, 0.000)  (-1.000,-5.000)  ( 7.000, 2.000)  ( 0.000, 4.000)
```

LSLCC/DLSLCC (Single/Double precision)

Solve a complex circulant linear system.

Usage

`CALL LSLCC (N, A, B, IPATH, X)`

Arguments

N — Order of the matrix represented by `A`. (Input)

A — Complex vector of length `N` containing the first row of the coefficient matrix. (Input)

B — Complex vector of length `N` containing the right-hand side of the linear system. (Input)

IPATH — Integer flag. (Input)
`IPATH` = 1 means the system $Ax = B$ is solved.

`IPATH` = 2 means the system $A^T x = B$ is solved.

X — Complex vector of length `N` containing the solution of the linear system. (Output)

Comments

1. Automatic workspace usage is

   `LSLCC` 8`N` + 15 units, or
   `DLSLCC` 16`N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `L2LCC`/`DL2LCC`. The reference is

   ```
   CALL L2LCC (N, A, B, IPATH, X, ACOPY, WK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work vector of length `N`. If `A` is not needed, then `A` and `ACOPY` may be the same.

   WK — Work vector of length 6`N` + 15.

2. Informational error

Type	Code	
4	2	The input matrix is singular.

3. Because of the special structure of circulant matrices, the first row of a circulant matrix completely characterizes the matrix. Hence, only the elements `A`(1, 1), …, `A`(1, `N`) need to be stored.

Algorithm

Circulant matrices have the property that each row is obtained by shifting the row above it one place to the right. Entries that are shifted off at the right re-enter at the left. For example,

$$A = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_4 & p_1 & p_2 & p_3 \\ p_3 & p_4 & p_1 & p_2 \\ p_2 & p_3 & p_4 & p_1 \end{bmatrix}$$

If $q_k = p_{-k}$ and the subscripts on p and q are interpreted modulo N, then

$$(Ax)_j = \sum_{i=1}^{N} p_{i-j+1} x_i = \sum_{i=1}^{N} q_{j-i+1} x_i = (q * x)_i$$

where $q * x$ is the convolution of q and x. By the convolution theorem, if $q * x = b$, then

$$\hat{q} \otimes \hat{x} = \hat{b}, \text{where } \hat{q}$$

is the discrete Fourier transform of q as computed by the IMSL routine `FFTCF` and $\otimes$ denotes elementwise multiplication. By division,

$$\hat{x} = \hat{b} \varnothing \hat{q}$$

where $\varnothing$ denotes elementwise division. The vector x is recovered from

$$\hat{x}$$

through the use of IMSL routine `FFTCB`.

To solve $A^T x = b$, use the vector p instead of q in the above algorithm.

Example

A system of four linear equations is solved. Note that only the first row of the matrix A is entered.

```
C                                 Declare variables
      INTEGER    IPATH, N
      PARAMETER  (N=4)
      COMPLEX    A(N), B(N), X(N)
C                                 Set values for  A, and B
C
C                                 A = ( 2+2i -3+0i  1+4i  6-2i)
C
C                                 B = (6+65i  -41-10i  -8-30i  63-3i)
C
      DATA A/(2.0,2.0), (-3.0,0.0), (1.0,4.0), (6.0,-2.0)/
      DATA B/(6.0,65.0), (-41.0,-10.0), (-8.0,-30.0), (63.0,-3.0)/
C                                 Solve AX = B     (IPATH = 1)
      IPATH = 1
      CALL LSLCC (N, A, B, IPATH, X)
C                                 Print results
      CALL WRCRN (’X’, 1, N, X, 1, 0)
      END
```

Output

```
              1                2                3                4
(-2.000, 0.000)  (-1.000,-5.000)  ( 7.000, 2.000)  ( 0.000, 4.000)
```

PCGRC/DPCGRC (Single/Double precision)

Solve a real symmetric definite linear system using a preconditioned conjugate gradient method with reverse communication.

Usage

`CALL PCGRC (IDO, N, X, P, R, Z, RELERR, ITMAX)`

Arguments

IDO — Flag indicating task to be done.   (Input/Output)
On the initial call `IDO` must be 0. If the routine returns with `IDO` = 1, then set

Z = AP, where A is the matrix, and call PCGRC again. If the routine returns with IDO = 2, then set Z to the solution of the system MZ = R, where M is the preconditioning matrix, and call PCGRC again. If the routine returns with IDO = 3, then the iteration has converged and X contains the solution.

N — Order of the linear system. (Input)

X — Array of length N containing the solution. (Input/Output)
On input, X contains the initial guess of the solution. On output, X contains the solution to the system.

P — Array of length N. (Output)
Its use is described under IDO.

R — Array of length N. (Input/Output)
On initial input, it contains the right-hand side of the linear system. On output, it contains the residual.

Z — Array of length N. (Input)
When IDO = 1, it contains AP, where A is the linear system. When IDO = 2, it contains the solution of MZ = R, where M is the preconditioning matrix. When IDO = 0, it is ignored. Its use is described under IDO.

RELERR — Relative error desired. (Input)

ITMAX — Maximum number of iterations allowed. (Input)

Comments

1. Automatic workspace usage is

   PCGRC 8 * ITMAX units, or
   DPCGRC 15 * ITMAX units.

   Workspace may be explicitly provided, if desired, by use of P2GRC/DP2GRC. The reference is

```
CALL P2GRC (IDO, N, X, P, R, Z, RELERR, ITMAX, TRI,
            WK, IWK)
```

   The additional arguments are as follows:

   TRI — Workspace of length 2 * ITMAX containing a tridiagonal matrix (in band symmetric form) whose largest eigenvalue is approximately the same as the largest eigenvalue of the iteration matrix. The workspace arrays TRI, WK and IWK should not be changed between the initial call with IDO = 0 and PCGRC/DPCGRC returning with IDO = 3.

   WK — Workspace of length 5 * ITMAX.

   IWK — Workspace of length ITMAX.

2. Informational errors

Type	Code	
4	1	The preconditioning matrix is singular.
4	2	The preconditioning matrix is not definite.
4	3	The linear system is not definite.
4	4	The linear system is singular.
4	5	No convergence after `ITMAX` iterations.

Algorithm

Routine `PCGRC` solves the symmetric definite linear system $Ax = b$ using the preconditioned conjugate gradient method. This method is described in detail by Golub and Van Loan (1983, Chapter 10), and in Hageman and Young (1981, Chapter 7).

The *preconditioning matrix*, M, is a matrix that approximates A, and for which the linear system $Mz = r$ is easy to solve. These two properties are in conflict; balancing them is a topic of much current research.

The number of iterations needed depends on the matrix and the error tolerance `RELERR`. As a rough guide, `ITMAX` = $\mathtt{N}^{1/2}$ is often sufficient when `N` >> 1. See the references for further information.

Let M be the preconditioning matrix, let b, p, r, x and z be vectors and let τ be the desired relative error. Then the algorithm used is as follows.

```
$\lambda = -1$

$p_0 = x_0$

$r_1 = b - Ap$

For $k = 1, \ldots,$ itmax

    $z_k = M^{-1} r_k$

    If $k = 1$ then
        $\beta_k = 1$
        $p_k = z_k$

    Else
        $\beta_k = z_k^T r_k / z_{k-1}^T r_{k-1}$
        $p_k = z_k + \beta_k p_k$

    End if
    $z_k = Ap$

    $\alpha_k = z_{k-1}^T r_{k-1} / z_k^T p_k$

    $x_k = x_k + \alpha_k p_k$

    $r_k = r_k - \alpha_k z_k$

    If $(\|z_k\|_2 \le \tau(1 - \lambda)\|x_k\|_2)$ Then
        Recompute $\lambda$
```

```
            If  $(\|z_k\|_2 \le \tau(1 - \lambda)\|x_k\|_2)$  Exit
      End if
end loop
```

Here λ is an estimate of $\lambda_{\max}(G)$, the largest eigenvalue of the iteration matrix $G = I - M^{-1} A$. The stopping criterion is based on the result (Hageman and Young, 1981, pages 148−151)

$$\frac{\left\|x_k - x\right\|_M}{\left\|x\right\|_M} \le \frac{1}{1-\lambda_{\max}(G)} \frac{\left\|z_k\right\|_M}{\left\|x_k\right\|_M}$$

Where

$$\left\|x\right\|_M^2 = x^T Mx$$

It is known that

$$\lambda_{\max}(T_1) \le \lambda_{\max}(T_2) \le \cdots \le \lambda_{\max}(G) < 1$$

where the T_n are the symmetric, tridiagonal matrices

$$T_n = \begin{bmatrix} \mu_1 & \omega_2 & & & \\ \omega_2 & \mu_2 & \omega_3 & & \\ & \omega_3 & \mu_3 & \omega_4 & \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

with

$$\mu_k = 1 - \beta_k / \alpha_{k-1} - 1/\alpha_k, \mu_1 = 1 - 1/\alpha_1$$

and

$$\omega_k = \sqrt{\beta_k} / \alpha_{k-1}$$

The largest eigenvalue of T_k is found using the routine `EVASB`. Usually this eigenvalue computation is needed for only a few of the iterations.

Example 1

In this example, the solution to a linear system is found. The coefficient matrix A is stored as a full matrix. The preconditioning matrix is the diagonal of A. This is called the *Jacobi preconditioner*. It is also used by the IMSL routine `JCGRC` on page 259.

```
      INTEGER    LDA, N
      PARAMETER  (N=3, LDA=N)
C
      INTEGER    IDO, ITMAX, J
      REAL       A(LDA,N), B(N), P(N), R(N), RELERR, X(N), Z(N)
      EXTERNAL   MURRV, PCGRC, SCOPY, WRRRN
C                                 (   1,  -3,   2   )
C                           A =   (  -3,  10,  -5   )
```

```
C                                  (    2,  -5,   6   )
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C                             B =   (   27.0, -78.0, 64.0  )
      DATA B/27.0, -78.0, 64.0/
C                                  Set R to right side
      CALL SCOPY (N, B, 1, R, 1)
C                                  Initial guess for X is B
      CALL SCOPY (N, B, 1, X, 1)
C
      ITMAX  = 100
      RELERR = 1.0E-5
      IDO    = 0
   10 CALL PCGRC (IDO, N, X, P, R, Z, RELERR, ITMAX)
      IF (IDO .EQ. 1) THEN
C                                  Set z = Ap
         CALL MURRV (N, N, A, LDA, N, P, 1, N, Z)
         GO TO 10
      ELSE IF (IDO .EQ. 2) THEN
C                                  Use diagonal of A as the
C                                  preconditioning matrix M
C                                  and set z = inv(M)*r
         DO 20  J=1, N
            Z(J) = R(J)/A(J,J)
   20    CONTINUE
         GO TO 10
      END IF
C                                  Print the solution
      CALL WRRRN ('Solution', N, 1, X, N, 0)
C
      END
```

Output

```
Solution
1   1.001
2  -4.000
3   7.000
```

Example 2

In this example, a more complicated preconditioner is used to find the solution of a linear system which occurs in a finite-difference solution of Laplace’s equation on a 4 × 4 grid. The matrix is

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & & & & & \\ -1 & 4 & -1 & 0 & -1 & & & & \\ 0 & -1 & 4 & -1 & 0 & -1 & & & \\ -1 & 0 & -1 & 4 & -1 & 0 & -1 & & \\ & -1 & 0 & -1 & 4 & -1 & 0 & -1 & \\ & & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\ & & & -1 & 0 & -1 & 4 & -1 & 0 \\ & & & & -1 & 0 & -1 & 4 & -1 \\ & & & & & -1 & 0 & -1 & 4 \end{bmatrix}$$

The preconditioning matrix *M* is the symmetric tridiagonal part of *A*,

$$M = \begin{bmatrix} 4 & -1 & & & & & & & \\ -1 & 4 & -1 & & & & & & \\ & -1 & 4 & -1 & & & & & \\ & & -1 & 4 & -1 & & & & \\ & & & -1 & 4 & -1 & & & \\ & & & & -1 & 4 & -1 & & \\ & & & & & -1 & 4 & -1 & \\ & & & & & & -1 & 4 & -1 \\ & & & & & & & -1 & 4 \end{bmatrix}$$

Note that *M*, called `PRECND` in the program, is factored once.

```
      INTEGER    LDA, LDPRE, N, NCODA, NCOPRE
      PARAMETER  (N=9, NCODA=3, NCOPRE=1, LDA=2*NCODA+1,
     &           LDPRE=NCOPRE+1)
C
      INTEGER    IDO, ITMAX
      REAL       A(LDA,N), P(N), PRECND(LDPRE,N), PREFAC(LDPRE,N),
     &           R(N), RCOND, RELERR, X(N), Z(N)
      EXTERNAL   LFCQS, LSLQS, MURBV, PCGRC, SSET, WRRRN
C                                 Set A in band form
      DATA A/3*0.0, 4.0, -1.0, 0.0, -1.0, 2*0.0, -1.0, 4.0, -1.0, 0.0,
     &     -1.0, 2*0.0, -1.0, 4.0, -1.0, 0.0, -1.0, -1.0, 0.0, -1.0,
     &     4.0, -1.0, 0.0, -1.0, -1.0, 0.0, -1.0, 4.0, -1.0, 0.0,
     &     -1.0, -1.0, 0.0, -1.0, 4.0, -1.0, 0.0, -1.0, -1.0, 0.0,
     &     -1.0, 4.0, -1.0, 2*0.0, -1.0, 0.0, -1.0, 4.0, -1.0, 2*0.0,
     &     -1.0, 0.0, -1.0, 4.0, 3*0.0/
C                                  Set PRECND in band symmetric form
      DATA PRECND/0.0, 4.0, -1.0, 4.0, -1.0, 4.0, 0.0, 4.0, -1.0, 4.0,
     &     0.0, 4.0, 0.0, 4.0, -1.0, 4.0, 0.0, 4.0/
C                                  Right side is (1, ..., 1)
      CALL SSET (N, 1.0, R, 1)
C                                  Initial guess for X is 0
      CALL SSET (N, 0.0, X, 1)
C                                  Factor the preconditioning matrix
      CALL LFCQS (N, PRECND, LDPRE, NCOPRE, PREFAC, LDPRE, RCOND)
C
      ITMAX  = 100
      RELERR = 1.0E-4
      IDO    = 0
   10 CALL PCGRC (IDO, N, X, P, R, Z, RELERR, ITMAX)
      IF (IDO .EQ. 1) THEN
C                                  Set z = Ap
         CALL MURBV (N, A, LDA, NCODA, NCODA, N, P, 1, N, Z)
         GO TO 10
      ELSE IF (IDO .EQ. 2) THEN
C                                  Solve PRECND*z = r for r
         CALL LSLQS (N, PREFAC, LDPRE, NCOPRE, R, Z)
         GO TO 10
      END IF
C                                  Print the solution
      CALL WRRRN ('Solution', N, 1, X, N, 0)
C
      END
```

Output

```
Solution
1   0.955
2   1.241
3   1.349
4   1.578
5   1.660
6   1.578
7   1.349
8   1.241
9   0.955
```

JCGRC/DJCGRC (Single/Double precision)

Solve a real symmetric definite linear system using the Jacobi-preconditioned conjugate gradient method with reverse communication.

Usage

```
CALL JCGRC (IDO, N, DIAG, X, P, R, Z, RELERR, ITMAX)
```

Arguments

IDO — Flag indicating task to be done. (Input/Output)
On the initial call IDO must be 0. If the routine returns with IDO = 1, then set Z = A ∗ P, where A is the matrix, and call JCGRC again. If the routine returns with IDO = 2, then the iteration has converged and X contains the solution.

N — Order of the linear system. (Input)

DIAG — Vector of length N containing the diagonal of the matrix. (Input)
Its elements must be all strictly positive or all strictly negative.

X — Array of length N containing the solution. (Input/Output)
On input, X contains the initial guess of the solution. On output, X contains the solution to the system.

P — Array of length N. (Output)
Its use is described under IDO.

R — Array of length N. (Input/Output)
On initial input, it contains the right-hand side of the linear system. On output, it contains the residual.

Z — Array of length N. (Input)
When IDO = 1, it contains AP, where A is the linear system. When IDO = 0, it is ignored. Its use is described under IDO.

RELERR — Relative error desired. (Input)

ITMAX — Maximum number of iterations allowed. (Input)

Comments

1. Automatic workspace usage is

   JCGRC 8 * ITMAX units, or
   DJCGRC 15 * ITMAX units.

   Workspace may be explicitly provided, if desired, by use of J2GRC/DJ2GRC. The reference is

   ```
   CALL J2GRC (IDO, N, DIAG, X, P, R, Z, RELERR, ITMAX,
               TRI, WK, IWK)
   ```

   The additional arguments are as follows:

   TRI — Workspace of length 2 * ITMAX containing a tridiagonal matrix (in band symmetric form) whose largest eigenvalue is approximately the same as the largest eigenvalue of the iteration matrix. The workspace arrays TRI, WK and IWK should not be changed between the initial call with IDO = 0 and JCGRC/DJCGRC returning with IDO = 2.

   WK — Workspace of length 5 * ITMAX.

   IWK — Workspace of length ITMAX.

2. Informational errors

Type	Code	
4	1	The diagonal contains a zero.
4	2	The diagonal elements have different signs.
4	3	No convergence after `ITMAX` iterations.
4	4	The linear system is not definite.
4	5	The linear system is singular.

Algorithm

Routine `JCGRC` solves the symmetric definite linear system $Ax = b$ using the Jacobi conjugate gradient method. This method is described in detail by Golub and Van Loan (1983, Chapter 10), and in Hageman and Young (1981, Chapter 7).

This routine is a special case of the routine `PCGRC`, with the diagonal of the matrix `A` used as the preconditioning matrix. For details of the algorithm see `PCGRC`, page 253.

The number of iterations needed depends on the matrix and the error tolerance `RELERR`. As a rough guide, `ITMAX` = $N^{1/2}$ is often sufficient when N » 1. See the references for further information.

Example

In this example, the solution to a linear system is found. The coefficient matrix A is stored as a full matrix.

```
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
C
      INTEGER    IDO, ITMAX
      REAL       A(LDA,N), B(N), DIAG(N), P(N), R(N), RELERR, X(N),
     &           Z(N)
      EXTERNAL   JCGRC, MURRV, SCOPY, WRRRN
C                                  (   1,  -3,   2   )
C                            A =   (  -3,  10,  -5   )
C                                  (   2,  -5,   6   )
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C                            B =   (   27.0, -78.0, 64.0  )
      DATA B/27.0, -78.0, 64.0/
C                                  Set R to right side
      CALL SCOPY (N, B, 1, R, 1)
C                                  Initial guess for X is B
      CALL SCOPY (N, B, 1, X, 1)
C                                  Copy diagonal of A to DIAG
      CALL SCOPY (N, A, LDA+1, DIAG, 1)
C                                  Set parameters
      ITMAX  = 100
      RELERR = 1.0E-5
      IDO    = 0
   10 CALL JCGRC (IDO, N, DIAG, X, P, R, Z, RELERR, ITMAX)
      IF (IDO .EQ. 1) THEN
C                                  Set z = Ap
```

```
         CALL MURRV (N, N, A, LDA, N, P, 1, N, Z)
         GO TO 10
      END IF
C                                  Print the solution
      CALL WRRRN ('Solution', N, 1, X, N, 0)
C
      END
```

Output

```
Solution
1   1.001
2  -4.000
3   7.000
```

GMRES/DGMRES (Single/Double precision)

Use GMRES with reverse communication to generate an approximate solution of $Ax = b$.

Usage

```
CALL GMRES (IDO, N, X, P, R, Z, TOL)
```

Arguments

IDO — Flag indicating task to be done. (Input/Output)
On the initial call IDO must be 0. If the routine returns with IDO = 1, then set $Z = AP$, where A is the matrix, and call GMRES again. If the routine returns with IDO = 2, then set Z to the solution of the system $MZ = P$, where M is the preconditioning matrix, and call GMRES again. If the routine returns with IDO = 3, set Z = $AM^{-1}P$, and call GMRES again. If the routine returns with IDO = 4, the iteration has converged, and X contains the approximate solution to the linear system.

N — Order of the linear system. (Input)

X — Array of length N containing an approximate solution. (Input/Output)
On input, X contains an initial guess of the solution. On output, X contains the approximate solution.

P — Array of length N. (Output)
Its use is described under IDO.

R — Array of length N. (Input/Output)
On initial input, it contains the right-hand side of the linear system. On output, it contains the residual, $b - Ax$.

Z — Array of length N. (Input)
When IDO = 1, it contains AP, where A is the coefficient matrix. When IDO = 2, it contains $M^{-1}P$. When IDO = 3, it contains $AM^{-1}P$. When IDO = 0, it is ignored.

TOL — Stopping tolerance. (Input/Output)
The algorithm attempts to generate a solution x such that $|b - Ax| \leq$ `TOL*`$|b|$. On output, `TOL` contains the final residual norm.

Comments

1. Automatic workspace usage is

   `GMRES` `N`(`KDMAX` + 2) + `KDMAX`^{2} + 3`KDMAX` + 2 units, or

   `DGMRES` 2`N`(`KDMAX` + 2) + 2`KDMAX`^{2} + 6`KDMAX` + 4 where
   `KDMAX` = `MIN`(`N`, 20) units.

   Workspace may be explicitly provided, if desired, by use of `G2RES`/`DG2RES`. The reference is

   ```
   CALL G2RES (IDO, N, X, P, R, Z, TOL, INFO, USRNPR,
               USRNRM, WORK)
   ```

   The additional arguments are as follows:

   INFO — Integer vector of length 10 used to change parameters of `GMRES`. (Input/Output).

   For any components `INFO(1)` ... `INFO(7)` with value zero on input, the default value is used.
   `INFO(1)` = `IMP`, the flag indicating the desired implementation.

IMP	**Action**
1	first Gram-Schmidt implementation
2	second Gram-Schmidt implementation
3	first Householder implementation
4	second Householder implementation

   Default: `IMP` = 1

   `INFO(2)` = `KDMAX`, the maximum Krylor subspace dimension, i.e., the maximum allowable number of `GMRES` iterations before restarting. It must satisfy 1 ≤ `KDMAX` ≤ *N*.
   Default: `KDMAX` = min(`N`, 20)

   `INFO(3)` = `ITMAX`, the maximum number of `GMRES` iterations allowed.
   Default: `ITMAX` = 1000

   `INFO(4)` = `IRP`, the flag indicating whether right preconditioning is used.
   If `IRP` = 0, no right preconditioning is performed. If `IRP` = 1, right preconditioning is performed. If `IRP` = 0, then `IDO` = 2 or 3 will not occur.
   Default: `IRP` = 0

   `INFO(5)` = `IRESUP`, the flag that indicates the desired residual vector updating prior to restarting or on termination.

IRESUP	**Action**
1	update by linear combination, restarting only
2	update by linear combination, restarting and termination
3	update by direct evaluation, restarting only
4	update by direct evaluation, restarting and termination

Updating by direct evaluation requires an otherwise unnecessary matrix-vector product. The alternative is to update by forming a linear combination of various available vectors. This may or may not be cheaper and may be less reliable if the residual vector has been greatly reduced. If IRESUP = 2 or 4, then the residual vector is returned in WORK(1), ..., WORK(N). This is useful in some applications but costs another unnecessary residual update. It is recommended that IRESUP = 1 or 2 be used, unless matrix-vector products are inexpensive or great residual reduction is required. In this case use IRESUP = 3 or 4. The meaning of "inexpensive" varies with IMP as follows:

IMP	≤
1	(KDMAX + 1) *N flops
2	N flops
3	(2*KDMAX + 1) *N flops
4	(2*KDMAX + 1) *N flops

"Great residual reduction" means that TOL is only a few orders of magnitude larger than machine epsilon.
Default: IRESUP = 1

INFO(6) = flag for indicating the inner product and norm used in the Gram-Schmidt implementations. If INFO(6) = 0, sdot and snrm2, from BLAS, are used. If INFO(6) = 1, the user must provide the routines, as specified under arguments USRNPR and USRNRM.
Default: INFO(6) = 0

INFO(7) = IPRINT, the print flag. If IPRINT = 0, no printing is performed. If IPRINT = 1, print the iteration numbers and residuals.
Default: IPRINT = 0

INFO(8) = the total number of GMRES iterations on output.

INFO(9) = the total number of matrix-vector products in GMRES on output.

INFO(10) = the total number of right preconditioner solves in GMRES on output if IRP = 1.

USRNPR — User-supplied FUNCTION to use as the inner product in the Gram-Schmidt implementation, if INFO(6) = 1. If INFO(6) = 0, the dummy function G8RES/DG8RES may be used. The usage is

```
REAL FUNCTION USRNPR (N, SX, INCX, SY, INCY)
```

N — Length of vectors X and Y.   (Input)

SX — Real vector of length MAX(N*IABS(INCX),1). (Input)

INCX — Displacement between elements of SX. (Input)
X(I) is defined to be SX(1+(I-1)*INCX) if INCX is greater than 0, or SX(1+(I-N)*INCX) if INCX is less than 0.

SY — Real vector of length MAX(N*IABS(INXY),1). (Input)

INCY — Displacement between elements of SY. (Input)
Y(I) is defined to be SY(1+(I-1)*INCY) if INCY is greater than 0, or SY(1+(I-N)*INCY) if INCY is less than zero.
USRNPR must be declared EXTERNAL in the calling program.

USRNRM — User-supplied FUNCTION to use as the norm ||*X*|| in the Gram-Schmidt implementation, if INFO(6) = 1. If INFO(6) = 0, the dummy function G9RES/DG9RES may be used.The usage is

```
REAL FUNCTION USRNRM (N, SX, INCX)
```

N — Length of vectors X and Y. (Input)

SX — Real vector of length MAX(N*IABS(INCX),1). (Input)

INCX — Displacement between elements of SX. (Input)
X(I) is defined to be SX(1+(I-1)*INCX) if INCX is greater than 0, or SX(1+(I-N)*INCX) if INCX is less than 0.
USRNRM must be declared EXTERNAL in the calling program.

WORK — Work array whose length is dependent on the chosen implementation.

IMP	**length of *WORK***
1	N*(KDMAX + 2) + KDMAX**2 + 3 *KDMAX + 2
2	N*(KDMAX + 2) + KDMAX**2 + 2 *KDMAX + 1
3	N*(KDMAX + 2) + 3 *KDMAX + 2
4	N*(KDMAX + 2) + KDMAX**2 + 2 *KDMAX + 2

Algorithm

The routine GMRES implements restarted GMRES with reverse communication to generate an approximate solution to $Ax = b$. It is based on GMRESD by Homer Walker.

There are four distinct GMRES implementations, selectable through the parameter vector INFO. The first Gram-Schmidt implementation, INFO(1) = 1, is essentially the original algorithm by Saad and Schultz (1986). The second Gram-Schmidt implementation, developed by Homer Walker and Lou Zhou, is simpler than the first implementation. The least squares problem is constructed in upper-triangular form and the residual vector updating at the end of a GMRES cycle is cheaper. The first Householder implementation is algorithm 2.2 of Walker (1988), but with more efficient correction accumulation at the end of each GMRES cycle. The second Householder implementation is algorithm 3.1 of Walker (1988). The products of Householder transformations are expanded as sums, allowing most work to be formulated as large scale matrix-vector operations. Although BLAS are used wherever possible, extensive use of Level 2 BLAS in the second Householder implementation may yield a performance advantage on certain computing environments.

The Gram-Schmidt implementations are less expensive than the Householder, the latter requiring about twice as much arithmetic beyond the coefficient matrix/vector products. However, the Householder implementations may be more reliable near the limits of residual reduction. See Walker (1988) for details. Issues such as the cost of coefficient matrix/vector products, availability of effective preconditioners, and features of particular computing environments may serve to mitigate the extra expense of the Householder implementations.

Example 1

This is a simple example of GMRES usage. A solution to a small linear system is found. The coefficient matrix *A* is stored as a full matrix, and no preconditioning is used. Typically, preconditioning is required to achieve convergence in a reasonable number of iterations.

```
C                   Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=3, LDA=N)
C                                 Specifications for local variables
      INTEGER    IDO, NOUT
      REAL       P(N), TOL, X(N), Z(N)
      REAL       A(LDA,N), R(N)
      SAVE       A, R
C                                 Specifications for intrinsics
      INTRINSIC  SQRT
      REAL       SQRT
C                                 Specifications for subroutines
      EXTERNAL   GMRES, MURRV, SSET, UMACH, WRRRN
C                                 Specifications for functions
      EXTERNAL   AMACH
      REAL       AMACH
C                                 ( 33.0  16.0  72.0)
C                             A = (-24.0 -10.0 -57.0)
C                                 ( 18.0 -11.0   7.0)
C
C                             B = (129.0 -96.0   8.5)
C
      DATA A/33.0, -24.0, 18.0, 16.0, -10.0, -11.0, 72.0, -57.0, 7.0/
      DATA R/129.0, -96.0, 8.5/
C
      CALL UMACH (2, NOUT)
C
C                                 Initial guess = (0 ... 0)
C
      CALL SSET (N, 0.0, X, 1)
C                                 Set stopping tolerance to
C                                 square root of machine epsilon
      TOL = SQRT(AMACH(4))
      IDO = 0
   10 CONTINUE
```

```
     CALL GMRES (IDO, N, X, P, R, Z, TOL)
     IF (IDO .EQ. 1) THEN
c                                  Set z = A*p
         CALL MURRV (N, N, A, LDA, N, P, 1, N, Z)
         GO TO 10
      END IF
c
      CALL WRRRN ('Solution', 1, N, X, 1, 0)

      WRITE (NOUT,'(A11, E15.5)') 'Residual = ', TOL

      END
```

Output

```
      Solution
    1       2       3
1.000   1.500   1.000
Residual =     0.29746E-05
```

Example 2

This example solves a linear system with a coefficient matrix stored in coordinate form, the same problem as in the document example for `LSLXG`, page 195. Jacobi preconditioning is used, i.e. the preconditioning matrix M is the diagonal matrix with $M_{ii} = A_{ii}$, for $i = 1, \ldots, n$.

```
      INTEGER    N, NZ
      PARAMETER  (N=6, NZ=15)
c                                  Specifications for local variables
      INTEGER    IDO, INFO(10), NOUT
      REAL       P(N), TOL, WORK(1000), X(N), Z(N)
      REAL       DIAGIN(N), R(N)
c                                  Specifications for intrinsics
      INTRINSIC  SQRT
      REAL       SQRT
c                                  Specifications for subroutines
      EXTERNAL   AMULTP, G2RES, ISET, SCOPY, SHPROD, SSET, UMACH, WRRRN
c                                  Specifications for functions
      EXTERNAL   AMACH, G8RES, G9RES
      REAL       AMACH
c
      DATA DIAGIN/0.1, 0.1, 0.0666667, 0.1, 1.0, 0.16666667/
      DATA R/10.0, 7.0, 45.0, 33.0, -34.0, 31.0/
c
      CALL UMACH (2, NOUT)
c                                  Initial guess = (1 ... 1)
      CALL SSET (N, 1.0, X, 1)
c                                  Set up the options vector INFO
c                                  to use preconditioning
      CALL ISET (10, 0, INFO, 1)
      INFO(4) = 1
c                                  Set stopping tolerance to
c                                  square root of machine epsilon
      TOL = SQRT(AMACH(4))
      IDO = 0
   10 CONTINUE
      CALL G2RES (IDO, N, X, P, R, Z, TOL, INFO, G8RES, G9RES, WORK)
      IF (IDO .EQ. 1) THEN
```

```
C                                  Set z = A*p
         CALL AMULTP (P, Z)
         GO TO 10
      ELSE IF (IDO .EQ. 2) THEN
C
C                                  Set z = inv(M)*p
C                                  The diagonal of inv(M) is stored
C                                  in DIAGIN
C
         CALL SHPROD (N, DIAGIN, 1, P, 1, Z, 1)
         GO TO 10
      ELSE IF (IDO .EQ. 3) THEN
C
C                                  Set z = A*inv(M)*p
C
         CALL SHPROD (N, DIAGIN, 1, P, 1, Z, 1)
         CALL SCOPY (N, Z, 1, P, 1)
         CALL AMULTP (P, Z)
         GO TO 10
      END IF
C
      CALL WRRRN ('Solution', N, 1, X, N, 0)
      WRITE (NOUT,'(A11, E15.5)') 'Residual = ', TOL
      END
C
      SUBROUTINE AMULTP (P, Z)
      INTEGER    NZ
      PARAMETER  (NZ=15)
C                                  SPECIFICATIONS FOR ARGUMENTS
      REAL       P(*), Z(*)
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    N
      PARAMETER  (N=6)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I
      INTEGER    IROW(NZ), JCOL(NZ)
      REAL       A(NZ)
      SAVE       A, IROW, JCOL
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSET
C                                  Define the matrix A
C
      DATA A/6.0, 10.0, 15.0, -3.0, 10.0, -1.0, -1.0, -3.0, -5.0, 1.0,
     &     10.0, -1.0, -2.0, -1.0, -2.0/
      DATA IROW/6, 2, 3, 2, 4, 4, 5, 5, 5, 5, 1, 6, 6, 2, 4/
      DATA JCOL/6, 2, 3, 3, 4, 5, 1, 6, 4, 5, 1, 1, 2, 4, 1/
C
      CALL SSET (N, 0.0, Z, 1)
C                                  Accumulate the product A*p in z
      DO 10  I=1, NZ
         Z(IROW(I)) = Z(IROW(I)) + A(I)*P(JCOL(I))
   10 CONTINUE
      RETURN
      END
```

Output

```
 Solution
1   1.000
2   2.000
3   3.000
4   4.000
5   5.000
6   6.000
Residual =     0.25882E-05
```

Example 3

The coefficient matrix in this example corresponds to the five-point discretization of the 2-d Poisson equation with the Dirichlet boundary condition. Assuming the natural ordering of the unknowns, and moving all boundary terms to the right hand side, we obtain the block tridiagonal matrix

$$A = \begin{bmatrix} T & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & T \end{bmatrix}$$

where

$$T = \begin{bmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix}$$

and I is the identity matrix. Discretizing on a $k \times k$ grid implies that T and I are both $k \times k$, and thus the coefficient matrix A is $k^2 \times k^2$.

The problem is solved twice, with discretization on a 50×50 grid. During both solutions, use the second Householder implementation to take advantage of the large scale matrix/vector operations done in Level 2 BLAS. Also choose to update the residual vector by direct evaluation since the small tolerance will require large residual reduction.

The first solution uses no preconditioning. For the second solution, we construct a block diagonal preconditioning matrix

$$M = \begin{bmatrix} T & & \\ & \ddots & \\ & & T \end{bmatrix}$$

M is factored once, and these factors are used in the forward solves and back substitutions necessary when `GMRES` returns with `IDO` = 2 or 3.

Timings are obtained for both solutions, and the ratio of the time for the solution with no preconditioning to the time for the solution with preconditioning is printed. Though the exact results are machine dependent, we see that the savings realized by faster convergence from using a preconditioner exceed the cost of factoring M and performing repeated forward and back solves.

```
      INTEGER    K, N
      PARAMETER  (K=50, N=K*K)
C                                 Specifications for local variables
      INTEGER    IDO, INFO(10), IR(20), IS(20), NOUT
      REAL       A(2*N), B(2*N), C(2*N), G8RES, G9RES, P(2*N), R(N),
     &           TNOPRE, TOL, TPRE, U(2*N), WORK(100000), X(N),
     &           Y(2*N), Z(2*N)
C                                 Specifications for subroutines
      EXTERNAL   AMULTP, G2RES, ISET, LSLCR, SCOPY, SSET, UMACH
C                                 Specifications for functions
      EXTERNAL   AMACH, CPSEC
      REAL       AMACH, CPSEC
C
      CALL UMACH (2, NOUT)
C                                 Right hand side and initial guess
C                                 to (1 ... 1)
      CALL SSET (N, 1.0, R, 1)
      CALL SSET (N, 1.0, X, 1)
C                                 Use the 2nd Householder
C                                 implementation and update the
C                                 residual by direct evaluation
      CALL ISET (10, 0, INFO, 1)
      INFO(1) = 4
      INFO(5) = 3
      TOL     = 100.0*AMACH(4)
      IDO     = 0
C                                 Time the solution with no
C                                 preconditioning
      TNOPRE  = CPSEC()
   10 CONTINUE
      CALL G2RES (IDO, N, X, P, R, Z, TOL, INFO, G8RES, G9RES, WORK)
      IF (IDO .EQ. 1) THEN
C
C                                 Set z = A*p
C
         CALL AMULTP (K, P, Z)
         GO TO 10
      END IF
      TNOPRE = CPSEC() - TNOPRE
C
      WRITE (NOUT,'(A32, I4)') 'Iterations, no preconditioner = ',
     &                         INFO(8)
C
C                                 Solve again using the diagonal blocks
C                                 of A as the preconditioning matrix M
      CALL SSET (N, 1.0, R, 1)
      CALL SSET (N, 1.0, X, 1)
C                                 Define M
      CALL SSET (N-1, -1.0, B, 1)
      CALL SSET (N-1, -1.0, C, 1)
      CALL SSET (N, 4.0, A, 1)
      INFO(4) = 1
      TOL     = 100.0*AMACH(4)
      IDO     = 0
      TPRE    = CPSEC()
C                                  Compute the LDU factorization of M
C
      CALL LSLCR (N, C, A, B, 6, Y, U, IR, IS)
   20 CONTINUE
      CALL G2RES (IDO, N, X, P, R, Z, TOL, INFO)
         IF (IDO .EQ. 1) THEN
C
C                                  Set z = A*p
C
         CALL AMULTP (K, P, Z)
         GO TO 20
      ELSE IF (IDO .EQ. 2) THEN
C
C                                  Set z = inv(M)*p
C
         CALL SCOPY (N, P, 1, Z, 1)
         CALL LSLCR (N, C, A, B, 5, Z, U, IR, IS)
         GO TO 20
      ELSE IF (IDO .EQ. 3) THEN
C
C                                  Set z = A*inv(M)*p
C
         CALL LSLCR (N, C, A, B, 5, P, U, IR, IS)
         CALL AMULTP (K, P, Z)
         GO TO 20
      END IF
      TPRE = CPSEC() - TPRE
      WRITE (NOUT,’(A35, I4)’) ’Iterations, with preconditioning = ’,
     &                        INFO(8)
      WRITE (NOUT,’(A45, F10.5)’) ’(Precondition time)/(No ’//
     &                           ’precondition time) = ’, TPRE/TNOPRE
C
      END
C
      SUBROUTINE AMULTP (K, P, Z)
C                                  Specifications for arguments
      INTEGER    K
      REAL       P(*), Z(*)
C                                  Specifications for local variables
      INTEGER    I, N
C                                  Specifications for subroutines
      EXTERNAL   SAXPY, SVCAL
C
      N = K*K
C                                  Multiply by diagonal blocks
C
      CALL SVCAL (N, 4.0, P, 1, Z, 1)
      CALL SAXPY (N-1, -1.0, P(2), 1, Z, 1)
      CALL SAXPY (N-1, -1.0, P, 1, Z(2), 1)
C
C                                  Correct for terms not properly in
C                                  block diagonal
      DO 10  I=K, N - K, K
         Z(I)   = Z(I) + P(I+1)
         Z(I+1) = Z(I+1) + P(I)
```

```
10 CONTINUE
C                                  Do the super and subdiagonal blocks,
C                                  the -I’s
C
      CALL SAXPY (N-K, -1.0, P(K+1), 1, Z, 1)
      CALL SAXPY (N-K, -1.0, P, 1, Z(K+1), 1)
C
      RETURN
      END
```

Output

```
Iterations, no preconditioner =  329
Iterations, with preconditioning =  192
(Precondition time)/(No precondition time) =    0.66278
```

LSQRR/DLSQRR (Single/Double precision)

Solve a linear least-squares problem without iterative refinement.

Usage

CALL LSQRR (NRA, NCA, A, LDA, B, TOL, X, RES, KBASIS)

Arguments

NRA — Number of rows of A.   (Input)

NCA — Number of columns of A.   (Input)

A — NRA by NCA matrix containing the coefficient matrix of the least-squares system to be solved.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

B — Vector of length NRA containing the right-hand side of the least-squares system.   (Input)

TOL — Scalar containing the nonnegative tolerance used to determine the subset of columns of *A* to be included in the solution.   (Input)
If TOL is zero, a full complement of min(NRA, NCA) columns is used. See Comments.

X — Vector of length NCA containing the solution vector with components corresponding to the columns not used set to zero.   (Output)

RES — Vector of length NRA containing the residual vector B – A * X.   (Output)

KBASIS — Scalar containing the number of columns used in the solution. (Output)

Comments

1. Automatic workspace usage is

   LSQRR (NRA + 4) * NCA – 1 units, or
   DLSQRR (2 * NRA + 7) * NCA – 2 units.

   Workspace may be explicitly provided, if desired, by use of L2QRR/DL2QRR. The reference is

   ```
   CALL L2QRR (NRA, NCA, A, LDA, B, TOL, X, RES,
               KBASIS, QR, QRAUX, IPVT, WORK)
   ```

   The additional arguments are as follows:

   QR — Work vector of length NRA * NCA representing an NRA by NCA matrix that contains information from the *QR* factorization of A. If A is not needed, *QR* can share the same storage locations as A.

   QRAUX — Work vector of length NCA containing information about the orthogonal factor of the *QR* factorization of A.

   IPVT — Integer work vector of length NCA containing the pivoting information for the *QR* factorization of A.

   WORK — Work vector of length 2 * NCA – 1.

2. Routine LSQRR calculates the *QR* decomposition with pivoting of a matrix A and tests the diagonal elements against a user-supplied tolerance TOL. The first integer KBASIS = *k* is determined for which

   $$\left|r_{k+1,k+1}\right| \le \text{TOL} \; * \left|r_{11}\right|$$

   In effect, this condition implies that a set of columns with a condition number approximately bounded by 1.0/TOL is used. Then, LQRSL performs a truncated fit of the first KBASIS columns of the permuted A to an input vector B. The coefficient of this fit is unscrambled to correspond to the original columns of A, and the coefficients corresponding to unused columns are set to zero. It may be helpful to scale the rows and columns of A so that the error estimates in the elements of the scaled matrix are roughly equal to TOL.

3. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2QRR the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSQRR. Additional memory allocation for FAC and option value restoration are done automatically in LSQRR. Users directly calling L2QRR can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSQRR or L2QRR. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSQRR temporarily replaces IVAL(2) by IVAL(1). The routine L2CRG computes the condition number if IVAL(2) = 2. Otherwise L2CRG skips this computation. LSQRR restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSQRR solves the linear least-squares problem. The routine LQRRR, page 286, is first used to compute the *QR* decomposition of *A*. Pivoting, with all rows free, is used. Column *k* is in the basis if

$$\left|R_{kk}\right| \le \tau\left|R_{11}\right|$$

with τ = TOL. The truncated least-squares problem is then solved using IMSL routine LQRSL, page 292. Finally, the components in the solution, with the same index as columns that are not in the basis, are set to zero; and then, the permutation determined by the pivoting in IMSL routine LQRRR is applied.

Example

Consider the problem of finding the coefficients c_i in

$$f(x) = c_0 + c_1 x + c_2 x^2$$

given data at x = 1, 2, 3 and 4, using the method of least squares. The row of the matrix *A* contains the value of 1, x and x^2 at the data points. The vector *b* contains the data, chosen such that $c_0 \approx 1$, $c_1 \approx 2$ and $c_2 \approx 0$. The routine LSQRR solves this least-squares problem.

```
C                                  Declare variables
      PARAMETER  (NRA=4, NCA=3, LDA=NRA)
      REAL       A(LDA,NCA), B(NRA), X(NCA), RES(NRA), TOL
C
C                                  Set values for A
C
C                                  A = (  1    2     4   )
C                                      (  1    4    16   )
C                                      (  1    6    36   )
C                                      (  1    8    64   )
C
      DATA A/4*1.0, 2.0, 4.0, 6.0, 8.0, 4.0, 16.0, 36.0, 64.0/
C
C                                  Set values for B
C
      DATA B/ 4.999,  9.001,  12.999,  17.001 /
C
C                                  Solve the least squares problem
```

```
      TOL = 1.0E-4
      CALL LSQRR (NRA, NCA, A, LDA, B, TOL, X, RES, KBASIS)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’KBASIS = ’, KBASIS
      CALL WRRRN (’X’, 1, NCA, X, 1, 0)
      CALL WRRRN (’RES’, 1, NRA, RES, 1, 0)
C
      END
```

Output

```
KBASIS =   3

            X
    1       2       3
0.999   2.000   0.000

                    RES
        1          2          3          4
-0.000400   0.001200  -0.001200   0.000400
```

LQRRV/DLQRRV (Single/Double precision)

Compute the least-squares solution using Householder transformations applied in blocked form.

Usage

CALL LQRRV (NRA, NCA, NUMEXC, A, LDA, X, LDX)

Arguments

NRA — Number of rows in the matrix.   (Input)

NCA — Number of columns in the matrix.   (Input)

NUMEXC — Number of right-hand sides.   (Input)

A — Real LDA by (NCA + NUMEXC) array containing the matrix and right-hand sides.   (Input)
The right-hand sides are input in A(1 : NRA, NCA + j), j = 1, …, NUMEXC. The array A is preserved upon output. The Householder factorization of the matrix is computed and used to solve the systems.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

X — Real LDX by NUMEXC array containing the solution.   (Output)

LDX — Leading dimension of the solution array X exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   LQRRV LDA * (NCA + NUMEXC) + (NCA + NUMEXC + 1) * (NB + 1) units, or

   DLQRRV 2(LDA * (NCA + NUMEXC) + (NCA + NUMEXC + 1) * (NB + 1)) units.

   where NB is the block size. The default value is NB = 1. This value can be reset. See Comment 3 below.

   Workspace may be explicitly provided, if desired, by use of L2RRV/DL2RRV. The reference is

   ```
   CALL L2RRV (NRA, NCA, NUMEXC, A, LDA, X, LDX, FAC,
             LDFAC, WK)
   ```

   The additional arguments are as follows:

   FAC — Work vector of length LDFAC * (NCA + NUMEXC) containing the Householder factorization of the matrix on output. If the input data is not needed, A and FAC can share the same storage locations.

   LDFAC — Leading dimension of the array FAC exactly as specified in the dimension statement of the calling program. (Input)
   If A and FAC are sharing the same storage, then LDA = LDFAC is required.

   WK — Work vector of length (NCA + NUMEXC + 1) * (NB + 1) . The default value is NB = 1. This value can be reset. See item 3 below.

2. Informational errors

Type	Code	
4	1	The input matrix is singular.

3. Integer Options with Chapter 10 Options Manager

   5 This option allows the user to reset the blocking factor used in computing the factorization. On some computers, changing IVAL(*) to a value larger than 1 will result in greater efficiency. The value IVAL(*) is the maximum value to use. (The software is specialized so that IVAL(*) is reset to an “optimal” used value within routine L2RRV.) The user can control the blocking by resetting IVAL(*) to a smaller value than the default. Default values are IVAL(*) = 1, IMACH(5).

   6 This option is the vector dimension where a shift is made from in-line level-2 loops to the use of level-2 BLAS in forming the partial product of Householder transformations. Default value is IVAL(*) = IMACH(5).

   10 This option allows the user to control the factorization step. If the value is 1 the Householder factorization will be computed.

If the value is 2, the factorization will not be computed. In this latter case the decomposition has already been computed. Default value is `IVAL(*)` = 1.

11 This option allows the user to control the solving steps. The rules for `IVAL(*)` are:

1. Compute $b \leftarrow Q^T b$, and $x \leftarrow R^+ b$.
2. Compute $b \leftarrow Q^T b$.
3. Compute $b \leftarrow Qb$.
4. Compute $x \leftarrow R^+ b$.

Default value is `IVAL (*)` = 1. Note that `IVAL (*)` = 2 or 3 may only be set when calling `L2RRV/DL2RRV`.

Algorithm

The routine `LQRRV` computes the *QR* decomposition of a matrix *A* using blocked Householder transformations. It is based on the storage-efficient *WY* representation for products of Householder transformations. See Schreiber and Van Loan (1989).

The routine `LQRRV` determines an orthogonal matrix *Q* and an upper triangular matrix *R* such that *A* = *QR*. The *QR* factorization of a matrix *A* having `NRA` rows and `NCA` columns is as follows:

```
Initialize $A_1 \leftarrow A$
For $k$ = 1, min(NRA - 1, NCA)
    Determine a Householder transformation for column $k$ of $A_k$ having the form
        $H_k = I - \tau_k \mu_k \mu_k^T$
    where $u_k$ has zeros in the first $k - 1$ positions and $\tau_k$ is a scalar.
    Update
        $A_k \leftarrow H_k A_{k-1} = A_{k-1} - \tau_k \mu_k \left( A_{k-1}^T \mu_k \right)^T$
End $k$
```

Thus,

$$A_p = H_p H_{p-1} \cdots H_1 A = Q^T A = R$$

where *p* = min(`NRA` – 1, `NCA`). The matrix *Q* is not produced directly by `LQRRV`. The information needed to construct the Householder transformations is saved instead. If the matrix *Q* is needed explicitly, Q^T can be determined while the matrix is factored. No pivoting among the columns is done. The primary purpose of `LQRRV` is to give the user a high-performance *QR* least-squares solver. It is intended for least-squares problems that are well-posed. For background, see Golub and Van Loan (1989, page 225). During the *QR* factorization, the most time–consuming step is computing the matrix–vector update $A_k \leftarrow H_k A_{k-1}$. The routine LQRRV constructs “block” of NB Householder transformations in which the update is “rich” in matrix multiplication. The product of NB Householder transformations are written in the form

$$H_k H_{k+1} \cdots H_{k+nb-1} = I + YTY^T$$

where $Y_{NRA \times NB}$ is a lower trapezoidal matrix and $T_{NB \times NB}$ is upper triangular. The optimal choice of the block size parameter NB varies among computer systems. Users may want to change it from its default value of 1.

Example

Given a real $m \times k$ matrix B it is often necessary to compute the k least-squares solutions of the linear system $AX = B$, where A is an $m \times n$ real matrix. When $m > n$ the system is considered *overdetermined*. A solution with a zero residual normally does not exist. Instead the minimization problem

$$\min_{x_j \in \mathbf{R}^n} \left\| Ax_j - b_j \right\|_2$$

is solved k times where x_j, b_j are the j-th columns of the matrices X, B respectively. When A is of full column rank there exits a unique solution X_{LS} that solves the above minimization problem. By using the routine LQRRV, X_{LS} is computed.

```
C                                 Declare variables
      INTEGER    LDA, LDX, NCA, NRA, NUMEXC
      PARAMETER  (NCA=3, NRA=5, NUMEXC=2, LDA=NRA, LDX=NCA)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       X(LDX,NUMEXC)
C                                 SPECIFICATIONS FOR SAVE VARIABLES
      REAL       A(LDA,NCA+NUMEXC)
      SAVE       A
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   LQRRV, SGEMM, WRRRN
C
C                                 Set values for A and the
C                                 righthand sides.
C
C                                 A = (  1    2     4 |   7  10)
C                                     (  1    4    16 |  21  10)
C                                     (  1    6    36 |  43  9 )
C                                     (  1    8    64 |  73  10)
C                                     (  1   10   100 | 111  10)
C
      DATA A/5*1.0, 2.0, 4.0, 6.0, 8.0, 10.0, 4.0, 16.0, 36.0, 64.0,
     &     100.0, 7.0, 21.0, 43.0, 73.0, 111.0, 2*10., 9., 2*10./
C
C
C                                 QR factorization and solution
      CALL LQRRV (NRA, NCA, NUMEXC, A, LDA, X, LDX)
      CALL WRRRN (’SOLUTIONS 1-2’, NCA, NUMEXC, X, LDX, 0)
C                                 Compute residuals and print
      CALL SGEMM (’N’, ’N’, NRA, NUMEXC, NCA, 1.E0, A, LDA, X, LDX,
     &            -1.E0, A(1,NCA+1), LDA)
```

```
      CALL WRRRN ('RESIDUALS 1-2', NRA, NUMEXC, A(1,NCA+1), LDA, 0)
C
      END
```

Output

```
   SOLUTIONS 1-2
        1       2
1    1.00   10.80
2    1.00   -0.43
3    1.00    0.04

   RESIDUALS 1-2
        1        2
1   0.0000   0.0857
2   0.0000  -0.3429
3   0.0000   0.5143
4   0.0000  -0.3429
5   0.0000   0.0857
```

LSBRR/DLSBRR (Single/Double precision)

Solve a linear least-squares problem with iterative refinement.

Usage

CALL LSBRR (NRA, NCA, A, LDA, B, TOL, X, RES, KBASIS)

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA matrix containing the coefficient matrix of the least-squares system to be solved. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Real vector of length NRA containing the right-hand side of the least-squares system. (Input)

TOL — Real scalar containing the nonnegative tolerance used to determine the subset of columns of A to be included in the solution. (Input)
If TOL is zero, a full complement of min(NRA, NCA) columns is used. See Comments.

X — Real vector of length NCA containing the solution vector with components corresponding to the columns not used set to zero. (Output)

RES — Real vector of length NRA containing the residual vector $B - AX$. (Output)

KBASIS — Integer scalar containing the number of columns used in the solution. (Output)

Comments

1. Automatic workspace usage is

   LSBRR NRA * NCA + 4 * NCA + NRA – 1 units, or
   DLSBRR 2 * NRA * NCA + 7 * NCA + 2 * NRA – 2 units.

   Workspace may be explicitly provided, if desired, by use of L2BRR/DL2BRR. The reference is

   ```
   CALL L2BRR (NRA, NCA, A, LDA, B, TOL, X, RES,
                KBASIS,QR, BRRUX, IPVT, WK)
   ```

   The additional arguments are as follows:

   QR — Work vector of length NRA * NCA representing an NRA by NCA matrix that contains information from the *QR* factorization of A. See LQRRR for details.

   BRRUX — Work vector of length NCA containing information about the orthogonal factor of the *QR* factorization of A. See LQRRR for details.

   IPVT — Integer work vector of length NCA containing the pivoting information for the *QR* factorization of A. See LQRRR for details.

   WK — Work vector of length NRA + 2 * NCA – 1.

2. Informational error

Type	Code	
4	1	The data matrix is too ill-conditioned for iterative refinement to be effective.

3. Routine LSBRR calculates the *QR* decomposition with pivoting of a matrix *A* and tests the diagonal elements against a user-supplied tolerance TOL. The first integer KBASIS = *k* is determined for which

   $$\left|r_{k+1,k+1}\right| \le \mathrm{TOL} * \left|r_{11}\right|$$

   In effect, this condition implies that a set of columns with a condition number approximately bounded by 1.0/TOL is used. Then, LQRSL performs a truncated fit of the first KBASIS columns of the permuted A to an input vector B. The coefficient of this fit is unscrambled to correspond to the original columns of A, and the coefficients corresponding to unused columns are set to zero. It may be helpful to scale the rows and columns of A so that the error estimates in the elements of the scaled matrix are roughly equal to TOL. The iterative refinement method of Björck is then applied to this factorization.

4. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2BRR the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSBRR. Additional memory allocation for FAC and option value restoration are done automatically in LSBRR. Users directly calling L2BRR can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSBRR or L2BRR. Default values for the option are IVAL(*) = 1, 16, 0, 1.

   17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSBRR temporarily replaces IVAL(2) by IVAL(1). The routine L2CRG computes the condition number if IVAL(2) = 2. Otherwise L2CRG skips this computation. LSBRR restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

Routine LSBRR solves the linear least-squares problem using iterative refinement. The iterative refinement algorithm is due to Björck (1967, 1968). It is also described by Golub and Van Loan (1983, pages 182−183).

Example

This example solves the linear least-squares problem with A, an 8 × 4 matrix. Note that the second and fourth columns of A are identical. Routine LSBRR determines that there are three columns in the basis.

```
C                                 Declare variables
      PARAMETER  (NRA=8, NCA=4, LDA=NRA)
      REAL       A(LDA,NCA), B(NRA), X(NCA), RES(NRA), TOL
C
C                                 Set values for A
C
C                                 A = (   1    5   15     5  )
C                                     (   1    4   17     4  )
C                                     (   1    7   14     7  )
C                                     (   1    3   18     3  )
C                                     (   1    1   15     1  )
C                                     (   1    8   11     8  )
C                                     (   1    3    9     3  )
C                                     (   1    4   10     4  )
C
      DATA A/8*1, 5., 4., 7., 3., 1., 8., 3., 4., 15., 17., 14.,
     &   18., 15., 11., 9., 10., 5., 4., 7., 3., 1., 8., 3., 4. /
C
C                                 Set values for B
C
```

```
      DATA B/ 30., 31., 35., 29., 18., 35., 20., 22. /
C
C                                 Solve the least squares problem
      TOL = 1.0E-4
      CALL LSBRR (NRA, NCA, A, LDA, B, TOL, X, RES, KBASIS)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’KBASIS = ’, KBASIS
      CALL WRRRN (’X’, 1, NCA, X, 1, 0)
      CALL WRRRN (’RES’, 1, NRA, RES, 1, 0)
C
      END
```

Output

```
KBASIS =   3
                X
      1       2       3       4
  0.636   2.845   1.058   0.000

                              RES
      1       2       3       4       5       6       7       8
 -0.733   0.996  -0.365   0.783  -1.353  -0.036   1.306  -0.597
```

LCLSQ/DLCLSQ (Single/Double precision)

Solve a linear least-squares problem with linear constraints.

Usage

```
CALL LCLSQ (NRA, NCA, NCON, A, LDA, B, C, LDC, BL, BU,
            IRTYPE, XLB, XUB, X, RES)
```

Arguments

NRA — Number of least-squares equations. (Input)

NCA — Number of variables. (Input)

NCON — Number of constraints. (Input)

A — Matrix of dimension `NRA` by `NCA` containing the coefficients of the `NRA` least squares equations. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)
`LDA` must be at least `NRA`.

B — Vector of length `NRA` containing the right-hand sides of the least squares equations. (Input)

C — Matrix of dimension `NCON` by `NCA` containing the coefficients of the `NCON` constraints. (Input)
If `NCON` = 0, `C` is not referenced.

LDC — Leading dimension of C exactly as specified in the dimension statement of the calling program. (Input)
LDC must be at least NCON.

BL — Vector of length NCON containing the lower limit of the general constraints. (Input)
If there is no lower limit on the I-th constraint, then BL(I) will not be referenced.

BU — Vector of length NCON containing the upper limit of the general constraints. (Input)
If there is no upper limit on the I-th constraint, then BU(I) will not be referenced. If there is no range constraint, BL and BU can share the same storage locations.

IRTYPE — Vector of length NCON indicating the type of constraints exclusive of simple bounds, where IRTYPE(I) = 0, 1, 2, 3 indicates .EQ., .LE., .GE., and range constraints respectively. (Input)

XLB — Vector of length NCA containing the lower bound on the variables. (Input)
If there is no lower bound on the I-th variable, then XLB(I) should be set to 1.0E30.

XUB — Vector of length NCA containing the upper bound on the variables. (Input)
If there is no upper bound on the I-th variable, then XUB(I) should be set to –1.0E30.

X — Vector of length NCA containing the approximate solution. (Output)

RES — Vector of length NRA containing the residuals $B - AX$ of the least-squares equations at the approximate solution. (Output)

Comments

1. Automatic workspace usage is

   LCLSQ (NCON + MAXDIM) * (NCA + NCON + 1) + 13 * NCA + 12 * NCON + 3 units, or

   DLCLSQ 2 * (NCON + MAXDIM) * (NCA + NCON + 1) + 23 * NCA + 21 * NCON + 6 units, where MAXDIM = max(NRA, NCA)

   Workspace may be explicitly provided, if desired, by use of L2LSQ/DL2LSQ. The reference is

   ```
   CALL L2LSQ (NRA, NCA, NCON, A, LDA, B, C, LDC, BL,
               BU, IRTYPE, XLB, XUB, X, RES, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length (NCON + MAXDIM) * (NCA + NCON + 1) + 10 * NCA + 9 * NCON + 3.

   IWK — Integer work vector of length 3 * (NCON + NCA).

2. Informational errors

Type	Code	
3	1	The rank determination tolerance is less than machine precision.
4	2	The bounds on the variables are inconsistent.
4	3	The constraint bounds are inconsistent.
4	4	Maximum number of iterations exceeded.

3. Integer Options with Chapter 10 Options Manager

13 Debug output flag. If more detailed output is desired, set this option to the value 1. Otherwise, set it to 0. Default value is 0.

14 Maximum number of add/drop iterations. If the value of this option is zero, up to 5 * max(nra, nca) iterations will be allowed. Otherwise set this option to the desired iteration limit. Default value is 0.

4. Floating Point Options with Chapter 10 Options Manager

2 The value of this option is the relative rank determination tolerance to be used. Default value is sqrt(`AMACH` (4)).

5 The value of this option is the absolute rank determination tolerance to be used. Default value is sqrt(`AMACH` (4)).

Algorithm

The routine `LCLSQ` solves linear least-squares problems with linear constraints. These are systems of least-squares equations of the form $Ax \cong$ b

subject to

$$b_l \le C_x \le b_u$$

$$x_l \le x \le x_u$$

Here, A is the coefficient matrix of the least-squares equations, b is the right-hand side, and C is the coefficient matrix of the constraints. The vectors b_l, b_u, x_l and x_u are the lower and upper bounds on the constraints and the variables, respectively. The system is solved by defining dependent variables $y \equiv Cx$ and then solving the least squares system with the lower and upper bounds on x and y. The equation Cx – y = 0 is a set of equality constraints. These constraints are realized by heavy weighting, i.e. a penalty method, Hanson, (1986, pages 826–834).

Example

A linear least-squares problem with linear constraints is solved.

```
C
C     Solve the following in the least squares sense:
C           3x1 + 2x2 +  x3 = 3.3
```

```
C           4x1 + 2x2 +  x3 = 2.3
C           2x1 + 2x2 +  x3 = 1.3
C            x1 +  x2 +  x3 = 1.0
C
C     Subject to:  x1 + x2 + x3 <= 1
C                  0 <= x1 <= .5
C                  0 <= x2 <= .5
C                  0 <= x3 <= .5
C
C ----------------------------------------------------------------------
C                               Declaration of variables
C
      INTEGER     NRA, NCA, MCON, LDA, LDC
      PARAMETER   (NRA=4, NCA=3, MCON=1, LDC=MCON, LDA=NRA)
C
      INTEGER     IRTYPE(MCON), NOUT
      REAL        A(LDA,NCA), B(NRA), BC(MCON), C(LDC,NCA), RES(NRA),
     &            RESNRM, XSOL(NCA), XLB(NCA), XUB(NCA)
C
      EXTERNAL    SNRM2, UMACH
      REAL        SNRM2
C                               Data initialization
C
      DATA A/3.0E0, 4.0E0, 2.0E0, 1.0E0, 2.0E0,
     &       2.0E0, 2.0E0, 1.0E0, 1.0E0, 1.0E0, 1.0E0, 1.0E0/,
     &     B/3.3E0, 2.3E0, 1.3E0, 1.0E0/,
     &     C/3*1.0E0/,
     &     BC/1.0E0/, IRTYPE/1/, XLB/3*0.0E0/, XUB/3*.5E0/
C
C                               Solve the bounded, constrained
C                               least squares problem.
C
      CALL LCLSQ (NRA, NCA, MCON, A, LDA, B, C, LDC, BC, BC, IRTYPE,
     &            XLB, XUB, XSOL, RES)
C                               Compute the 2-norm of the residuals.
      RESNRM = SNRM2 (NRA, RES, 1)
C                               Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT, 999) XSOL, RES, RESNRM
C
 999  FORMAT ('  The solution is ', 3F9.4, //, '  The residuals ',
     &       'evaluated at the solution are ', /, 18X, 4F9.4, //,
     &       '  The norm of the residual vector is ', F8.4)
C
      END
C
```

Output

```
 The solution is    0.5000   0.3000   0.2000
 The residuals evaluated at the solution are
                  -1.0000   0.5000   0.5000   0.0000

     The norm of the residual vector is   1.2247
```

LQRRR/DLQRRR (Single/Double precision)

Compute the *QR* decomposition, *AP* = *QR*, using Householder transformations.

Usage

```
CALL LQRRR (NRA, NCA, A, LDA, PIVOT, IPVT, QR, LDQR, QRAUX,
            CONORM)
```

Arguments

NRA — Number of rows of A.   (Input)

NCA — Number of columns of A.   (Input)

A — Real NRA by NCA matrix containing the matrix whose *QR* factorization is to be computed.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

PIVOT — Logical variable.   (Input)
PIVOT = .TRUE. means column pivoting is enforced.
PIVOT = .FALSE. means column pivoting is not done.

IPVT — Integer vector of length NCA containing information that controls the final order of the columns of the factored matrix A.   (Input/Output)
On input, if IPVT(K) > 0, then the K-th column of A is an initial column. If IPVT(K) = 0, then the K-th column of A is a free column. If IPVT(K) < 0, then the K-th column of A is a final column. See Comments.
On output, IPVT(K) contains the index of the column of A that has been interchanged into the K-th column. This defines the permutation matrix *P*. The array IPVT is referenced only if PIVOT is equal to .TRUE.

QR — Real NRA by NCA matrix containing information required for the *QR* factorization.   (Output)
The upper trapezoidal part of *QR* contains the upper trapezoidal part of *R* with its diagonal elements ordered in decreasing magnitude. The strict lower trapezoidal part of *QR* contains information to recover the orthogonal matrix *Q* of the factorization. Arguments A and QR can occupy the same storage locations. In this case, A will not be preserved on output.

LDQR — Leading dimension of QR exactly as specified in the dimension statement of the calling program.   (Input)

QRAUX — Real vector of length NCA containing information about the orthogonal part of the decomposition in the first min(NRA, NCA) position. (Output)

CONORM — Real vector of length NCA containing the norms of the columns of the input matrix.   (Output)

If this information is not needed, CONORM and QRAUX can share the same storage locations.

Comments

1. Automatic workspace usage is

   LQRRR 2NCA – 1 units, or
   DLQRRR 4NCA – 2 units.

   Workspace may be explicitly provided, if desired, by use of L2RRR/DL2RRR. The reference is

   ```
   CALL L2RRR (NRA, NCA, A, LDA, PIVOT, IPVT, QR, LDQR,
               QRAUX, CONORM, WORK)
   ```

   The additional argument is

   WORK — Work vector of length 2NCA – 1. Only NCA – 1 locations of WORK are referenced if PIVOT = .FALSE. .

2. LQRRR determines an orthogonal matrix *Q*, permutation matrix *P*, and an upper trapezoidal matrix *R* with diagonal elements of nonincreasing magnitude, such that *AP* = *QR*. The Householder transformation for column *k*, *k* = 1, …, min(NRA, NCA) is of the form

   $$I - u_k^{-1} u u^T$$

   where *u* has zeros in the first *k* – 1 positions. If the explicit matrix *Q* is needed, the user can call routine LQERR (page 289) after calling LQRRR. This routine accumulates *Q* from its factored form.

3. Before the decomposition is computed, initial columns are moved to the beginning and the final columns to the end of the array A. Both initial and final columns are not moved during the computation. Only free columns are moved. Pivoting, if requested, is done on the free columns of largest reduced norm.

4. When pivoting has been selected by having entries of IPVT initialized to zero, an estimate of the condition number of A can be obtained from the output by computing the magnitude of the number QR(1, 1)/QR(K, K), where K = MIN(NRA, NCA). This estimate can be used to select the number of columns, KBASIS, used in the solution step computed with routine LQRSL (page 292).

Algorithm

The routine LQRRR computes the *QR* decomposition of a matrix using Householder transformations. It is based on the LINPACK routine SQRDC; see Dongarra et al. (1979).

LQRRR determines an orthogonal matrix *Q*, a permutation matrix *P*, and an upper trapezoidal matrix *R* with diagonal elements of nonincreasing magnitude, such that *AP* = *QR*. The Householder transformation for column *k* is of the form

$$I - \frac{u_k u_k^T}{p_k}$$

for $k = 1, 2, \ldots, \min(\mathrm{NRA}, \mathrm{NCA})$, where *u* has zeros in the first $k - 1$ positions. The matrix *Q* is not produced directly by LQRRR . Instead the information needed to reconstruct the Householder transformations is saved. If the matrix *Q* is needed explicitly, the subroutine LQERR, described on page 289, can be called after LQRRR. This routine accumulates *Q* from its factored form.

Before the decomposition is computed, initial columns are moved to the beginning of the array *A* and the final columns to the end. Both initial and final columns are frozen in place during the computation. Only free columns are pivoted. Pivoting, when requested, is done on the free columns of largest reduced norm.

Example

In various statistical algorithms it is necessary to compute $q = x^T(A^T A)^{-1} x$, where *A* is a rectangular matrix of full column rank. By using the *QR* decomposition, *q* can be computed without forming $A^T A$. Note that

$$A^T A = (QRP^{-1})^T (QRP^{-1}) = P^{-\mathrm{T}} R^T (Q^T Q) RP^{-1} = P\, R^T RP^T$$

since *Q* is orthogonal ($Q^T Q = I$) and *P* is a permutation matrix. Let

$$Q^T AP = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

where R_1 is an upper triangular nonsingular matrix. Then

$$x^T \left(A^T A\right)^{-1} x = x^T P R_1^{-1} R_1^{-T} P^{-1} x = \left\| R_1^{-T} P^{-1} x \right\|_2^2$$

In the following program, first the vector $t = P^{-1}\ x$ is computed. Then

$$t := R_1^{-T} t$$

Finally,

$$q = \|t\|^2$$

```
C                                 Declare variables
      INTEGER    LDA, LDQR, NCA, NRA
      PARAMETER  (NCA=3, NRA=4, LDA=NRA, LDQR=NRA)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDQ
      PARAMETER  (LDQ=NRA)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
```

```
      INTEGER    IPVT(NCA), NOUT
      REAL       CONORM(NCA), Q, QR(LDQR,NCA), QRAUX(NCA), T(NCA)
      LOGICAL    PIVOT
      REAL       A(LDA,NCA), X(NCA)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   ISET, LQRRR, LSLRT, PERMU, UMACH
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   SDOT
      REAL       SDOT
C
C                                  Set values for A
C
C                                  A = (  1    2     4   )
C                                      (  1    4    16   )
C                                      (  1    6    36   )
C                                      (  1    8    64   )
C
      DATA A/4*1.0, 2.0, 4.0, 6.0, 8.0, 4.0, 16.0, 36.0, 64.0/
C
C                                  Set values for X
C
C                                  X = (  1    2     3  )
      DATA X/1.0, 2.0, 3.0/
C
C                                  QR factorization
      PIVOT = .TRUE.
      CALL ISET (NCA, 0, IPVT, 1)
      CALL LQRRR (NRA, NCA, A, LDA, PIVOT, IPVT, QR, LDQR, QRAUX,
     &            CONORM)
C                                  Set t = inv(P)*x
      CALL PERMU (NCA, X, IPVT, 1, T)
C                                  Compute t = inv(trans(R))*t
      CALL LSLRT (NCA, QR, LDQR, T, 4, T)
C                                  Compute 2-norm of t, squared.
      Q = SDOT(NCA,T,1,T,1)
C                                  Print result
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’Q = ’, Q
C
      END
```

Output

```
Q =    0.840624
```

LQERR/DLQERR (Single/Double precision)

Accumulate the orthogonal matrix *Q* from its factored form given the *QR* factorization of a rectangular matrix *A*.

Usage

```
CALL LQERR (NRQR, NCQR, QR, LDQR, QRAUX, Q, LDQ)
```

Arguments

NRQR — Number of rows in QR.   (Input)

NCQR — Number of columns in QR.   (Input)

QR — Real NRQR by NCQR matrix containing the factored form of the matrix Q in the first min(NRQR, NCQR) columns of the strict lower trapezoidal part of QR as output from subroutine LQRRR/DLQRRR.   (Input)

LDQR — Leading dimension of QR exactly as specified in the dimension statement of the calling program.   (Input)

QRAUX — Real vector of length NCQR containing information about the orthogonal part of the decomposition in the first min(NRQR, NCQR) position as output from routine LQRRR/DLQRRR.   (Input)

Q — Real NRQR by NRQR matrix containing the accumulated orthogonal matrix Q; Q and QR can share the same storage locations if QR is not needed.   (Output)

LDQ — Leading dimension of Q exactly as specified in the dimension statement of the calling program.   (Input)

Comments

Automatic workspace usage is

LQERR   2 * NRQR units, or
DLQERR 4 * NRQR units.

Workspace may be explicitly provided, if desired, by use of L2ERR/DL2ERR. The reference is

```
CALL L2ERR (NRQR, NCQR, QR, LDQR, QRAUX, Q, LDQ, WK)
```

The additional argument is

WK — Work vector of length 2 * NRQR.

Algorithm

The routine LQERR accumulates the Householder transformations computed by IMSL routine LQRRR, page 286, to produce the orthogonal matrix *Q*.

Example

In this example, the orthogonal matrix *Q* in the *QR* decomposition of a matrix *A* is computed. The product *X* = *QR* is also computed. Note that *X* can be obtained from *A* by reordering the columns of *A* according to IPVT.

```
C                                 Declare variables
      INTEGER    LDA, LDQ, LDQR, NCA, NRA
      PARAMETER  (NCA=3, NRA=4, LDA=NRA, LDQ=NRA, LDQR=NRA)
C
      INTEGER    IPVT(NCA), J
      REAL       A(LDA,NCA), CONORM(NCA), Q(LDQ,NRA), QR(LDQR,NCA),
     &           QRAUX(NCA), R(NRA,NCA), X(NRA,NCA)
      LOGICAL    PIVOT
      EXTERNAL   ISET, LQERR, LQRRR, MRRRR, SCOPY, SSET, WRIRN, WRRRN
C
C                                 Set values for A
C
C                                 A = (  1    2     4   )
C                                     (  1    4    16   )
C                                     (  1    6    36   )
C                                     (  1    8    64   )
C
      DATA A/4*1.0, 2.0, 4.0, 6.0, 8.0, 4.0, 16.0, 36.0, 64.0/
C
C                                 QR factorization
C                                 Set IPVT = 0 (all columns free)
      CALL ISET (NCA, 0, IPVT, 1)
      PIVOT = .TRUE.
      CALL LQRRR (NRA, NCA, A, LDA, PIVOT, IPVT, QR, LDQR, QRAUX,
     &            CONORM)
C                                 Accumulate Q
      CALL LQERR (NRA, NCA, QR, LDQR, QRAUX, Q, LDQ)
C                                 R is the upper trapezoidal part of QR
      CALL SSET (NRA*NCA, 0.0, R, 1)
      DO 10  J=1, NRA
         CALL SCOPY (J, QR(1,J), 1, R(1,J), 1)
   10 CONTINUE
C                                 Compute X = Q*R
      CALL MRRRR (NRA, NRA, Q, LDQ, NRA, NCA, R, NRA, NRA, NCA, X, LDA)
C                                 Print results
      CALL WRIRN (’IPVT’, 1, NCA, IPVT, 1, 0)
      CALL WRRRN (’Q’, NRA, NRA, Q, LDQ, 0)
      CALL WRRRN (’R’, NRA, NCA, R, NRA, 0)
      CALL WRRRN (’X = Q*R’, NRA, NCA, X, LDA, 0)
C
      END
```

Output

```
   IPVT
 1   2   3
 3   2   1
                   Q
         1        2        3        4
1  -0.0531  -0.5422   0.8082  -0.2236
2  -0.2126  -0.6574  -0.2694   0.6708
3  -0.4783  -0.3458  -0.4490  -0.6708
4  -0.8504   0.3928   0.2694   0.2236

             R
        1       2       3
1  -75.26  -10.63   -1.59
2    0.00   -2.65   -1.15
3    0.00    0.00    0.36
4    0.00    0.00    0.00

          X = Q*R
        1       2       3
1    4.00    2.00    1.00
2   16.00    4.00    1.00
```

```
3   36.00    6.00    1.00
4   64.00    8.00    1.00
```

LQRSL/DLQRSL (Single/Double precision)

Compute the coordinate transformation, projection, and complete the solution of the least-squares problem $Ax = b$.

Usage

```
CALL LQRSL (NRA, KBASIS, QR, LDQR, QRAUX, B, IPATH, QB,
            QTB, X, RES, AX)
```

Arguments

NRA — Number of rows of matrix A. (Input)

KBASIS — Number of columns of the submatrix A_k of A. (Input)
The value `KBASIS` must not exceed min(`NRA`, `NCA`), where `NCA` is the number of columns in matrix A. The value `NCA` is an argument to routine `LQRRR` (page 286). The value of `KBASIS` is normally `NCA` unless the matrix is rank-deficient. The user must analyze the problem data and determine the value of `KBASIS`. See Comments.

QR — `NRA` by `NCA` array containing information about the QR factorization of A as output from routine `LQRRR`/`DLQRRR`. (Input)

LDQR — Leading dimension of QR exactly as specified in the dimension statement of the calling program. (Input)

QRAUX — Vector of length `NCA` containing information about the QR factorization of A as output from routine `LQRRR`/`DLQRRR`. (Input)

B — Vector b of length `NRA` to be manipulated. (Input)

IPATH — Option parameter specifying what is to be computed. (Input)
The value `IPATH` has the decimal expansion `IJKLM`, such that:
`I` ≠ 0 means compute Qb;

`J` ≠ 0 means compute Q^Tb;

`K` ≠ 0 means compute Q^Tb and x;

`L` ≠ 0 means compute Q^Tb and $b - Ax$;

`M` ≠ 0 means compute Q^Tb and Ax.

For example, if the decimal number `IPATH` = 01101, then `I` = 0, `J` = 1, `K` = 1, `L`= 0, and `M`= 1.

QB — Vector of length `NRA` containing Qb if requested in the option `IPATH`. (Output)

QTB — Vector of length NRA containing $Q^T b$ if requested in the option IPATH. (Output)

X — Vector of length KBASIS containing the solution of the least-squares problem $A_k x = b$, if this is requested in the option IPATH. (Output)
If pivoting was requested in routine LQRRR/DLQRRR, then the J-th entry of X will be associated with column IPVT(J) of the original matrix A. See Comments.

RES — Vector of length NRA containing the residuals $(b - Ax)$ of the least-squares problem if requested in the option IPATH. (Output)
This vector is the orthogonal projection of b onto the orthogonal complement of the column space of A.

AX — Vector of length NRA containing the least-squares approximation Ax if requested in the option IPATH. (Output)
This vector is the orthogonal projection of b onto the column space of A.

Comments

1. Informational error

Type	Code	
4	1	Computation of the least-squares solution of AK * X = B is requested, but the upper triangular matrix R from the QR factorization is singular.

2. This routine is designed to be used together with LQRRR. It assumes that LQRRR/DLQRR has been called to get QR, QRAUX and IPVT. The submatrix A_k mentioned above is actually equal to A_k = (A(IPVT(1)), A(IPVT(2)), …, A(IPVT (KBASIS))), where A(IPVT(I)) is the IPVT(I)-th column of the original matrix.

Algorithm

Routine LQRSL is based on the LINPACK routine SQRSL, see Dongarra et al. (1979).

The most important use of LQRSL is for solving the least-squares problem $Ax = b$, with coefficient matrix A and data vector b. This problem can be formulated, using the *normal equations* method, as $A^T Ax = A^T b$. Using LQRRR (page 286) the QR decomposition of A, $AP = QR$, is computed. Here P is a permutation matrix ($P^{-1} = P^{\mathrm{T}}$), Q is an orthogonal matrix ($Q^{-1} = Q^T$) and R is an upper trapezoidal matrix. The normal equations can then be written as

$$(PR^T)(Q^T Q)R(P^T x) = (PR^T)Q^T b$$

If $A^T A$ is nonsingular, then R is also nonsingular and the normal equations can be written as $R(P^T x) = Q^T b$. LQRSL can be used to compute $Q^T b$ and then solve for $P^T x$. Note that the *permuted* solution is returned.

The routine LQRSL can also be used to compute the least-squares residual, $b - Ax$. This is the projection of b onto the orthogonal complement of the column space of A. It can also compute Qb, $Q^T b$ and Ax, the orthogonal projection of x onto the column space of A.

Example

Consider the problem of finding the coefficients c_i in

$$f(x) = c_0 + c_1 x + c_2 x^2$$

given data at $x_i = 2_i$, ι = 1, 2, 3, 4, using the method of least squares. The row of the matrix A contains the value of 1, x_i and

$$x_i^2$$

at the data points. The vector b contains the data. The routine LQRRR is used to compute the QR decomposition of A. Then LQRSL is then used to solve the least-squares problem and compute the residual vector.

```
C                                 Declare variables
      PARAMETER  (NRA=4, NCA=3, KBASIS=3, LDA=NRA, LDQR=NRA)
      INTEGER    IPVT(NCA)
      REAL       A(LDA,NCA), QR(LDQR,NCA), QRAUX(NCA), CONORM(NCA),
     &           X(KBASIS), QB(1), QTB(NRA), RES(NRA),
     &           AX(1), B(NRA)
      LOGICAL    PIVOT
C
C                                 Set values for A
C
C                                 A = (  1    2     4   )
C                                     (  1    4    16   )
C                                     (  1    6    36   )
C                                     (  1    8    64   )
C
      DATA A/4*1.0, 2.0, 4.0, 6.0, 8.0, 4.0, 16.0, 36.0, 64.0/
C
C                                 Set values for B
C
C                                 B = ( 16.99  57.01  120.99  209.01 )
      DATA B/ 16.99,  57.01,  120.99,  209.01 /
C
C                                 QR factorization
      PIVOT = .TRUE.
      CALL ISET (NCA, 0, IPVT, 1)
      CALL LQRRR (NRA, NCA, A, LDA, PIVOT, IPVT, QR, LDQR, QRAUX,
     &  CONORM)
C                                 Solve the least squares problem
      IPATH = 00110
      CALL LQRSL (NRA, KBASIS, QR, LDQR, QRAUX, B, IPATH, QB, QTB, X,
     &   RES, AX)
C                                 Print results
      CALL WRIRN ('IPVT', 1, NCA, IPVT, 1, 0)
      CALL WRRRN ('X', 1, KBASIS, X, 1, 0)
      CALL WRRRN ('RES', 1, NRA, RES, 1, 0)
C
```

```
      END
```

Output

```
   IPVT
 1   2   3
 3   2   1

           X
    1       2       3
3.000   2.002   0.990

                   RES
       1         2         3         4
-0.00400   0.01200  -0.01200   0.00400
```

Note that since `IPVT` is (3, 2, 1) the array `X` contains the solution coefficients c_i in reverse order.

LUPQR/DLUPQR (Single/Double precision)

Compute an updated *QR* factorization after the rank-one matrix αxy^T is added.

Usage

```
CALL LUPQR (NROW, NCOL, ALPHA, W, Y, Q, LDQ, R, LDR, IPATH,
            QNEW, LDQNEW, RNEW, LDRNEW)
```

Arguments

NROW — Number of rows in the matrix $A = Q * R$. (Input)

NCOL — Number of columns in the matrix $A = Q * R$. (Input)

ALPHA — Scalar determining the rank-one update to be added. (Input)

W — Vector of length `NROW` determining the rank-one matrix to be added. (Input)
The updated matrix is $A + \alpha xy^T$. If `I` = 0 then *W* contains the vector *x*. If `I` = 1 then *W* contains the vector $Q^T x$.

Y — Vector of length `NCOL` determining the rank-one matrix to be added. (Input)

Q — Matrix of order `NROW` containing the *Q* matrix from the *QR* factorization. (Input)
Ignored if `IPATH` = 0.

LDQ — Leading dimension of `Q` exactly as specified in the dimension statement of the calling program. (Input)
Ignored if `IPATH` = 0.

R — Matrix of order NROW by NCOL containing the *R* matrix from the *QR* factorization.   (Input)
Only the upper trapezoidal part of R is referenced.

LDR — Leading dimension of R exactly as specified in the dimension statement of the calling program.   (Input)

IPATH — Flag used to control the computation of the *QR* update.   (Input)
IPATH has the decimal expansion IJ such that: I = 0 means *W* contains the vector *x*. I = 1 means W contains the vector $Q^T x$. J = 0 means do not update the matrix Q. J = 1 means update the matrix *Q*. For example, if IPATH = 10 then, I = 1 and J = 0.

QNEW — Matrix of order NROW containing the updated Q matrix in the *QR* factorization.   (Output)
Ignored if J = 0, see IPATH for definition of J.

LDQNEW — Leading dimension of QNEW exactly as specified in the dimension statement of the calling program.   (Input)
Ignored if J = 0; see IPATH for definition of J.

RNEW — Matrix of order NROW by NCOL containing the updated R matrix in the *QR* factorization.   (Output)
Only the upper trapezoidal part of RNEW is updated. R and RNEW may be the same.

LDRNEW — Leading dimension of RNEW exactly as specified in the dimension statement of the calling program.   (Input)

Comments

Automatic workspace usage is

LUPQR NROW + MIN(NROW – 1, NCOL) units, or
DLUPQR 2 * (NROW + MIN(NROW – 1, NCOL)) units.

Workspace may be explicitly provided, if desired, by use of L2PQR/DL2PQR. The reference is

```
CALL L2PQR (NROW, NCOL, ALPHA, W, Y, Q, LDQ, R, LDR, IPATH,
            QNEW, LDQNEW, RNEW, LDRNEW, Z, WORK)
```

The additional arguments are as follows:

Z — Work vector of length NROW.

WORK — Work vector of length MIN(NROW – 1, NCOL).

Algorithm

Let *A* be an $m \times n$ matrix and let $A = QR$ be its *QR* decomposition. (In the program, *m* is called NROW and *n* is called NCOL) Then

$$A + \alpha x y^T = QR + \alpha x y^T = Q(R + \alpha Q^T x y^T) = Q(R + \alpha w y^T)$$

where $w = Q^T x$. An orthogonal transformation J can be constructed, using a sequence of $m - 1$ Givens rotations, such that $Jw = \omega e_1$, where $\omega = \pm\|w\|_2$ and $e_1 = (1, 0, \ldots, 0)^T$. Then

$$A + \alpha xy^T = (QJ^T)(JR + \alpha\omega e_1 y^T)$$

Since JR is an upper Hessenberg matrix, $H = JR + \alpha\omega e_1 y^T$ is also an upper Hessenberg matrix. Again using $m - 1$ Givens rotations, an orthogonal transformation G can be constructed such that GH is an upper triangular matrix. Then

$$A + \alpha xy^T = \tilde{Q}\tilde{R}, \text{ where } \tilde{Q} = QJ^T G^T$$

is orthogonal and

$$\tilde{R} = GH$$

is upper triangular.

If the last k components of w are zero, then the number of Givens rotations needed to construct J or G is $m - k - 1$ instead of $m - 1$.

For further information, see Dennis and Schnabel (1983, pages 55−58 and 311−313), or Golub and Van Loan (1983, pages 437−439).

Example

The QR factorization of A is found. It is then used to find the QR factorization of $A + xy^T$. Since pivoting is used, the QR factorization routine finds $AP = QR$, where P is a permutation matrix determined by IPVT. We compute

$$AP + \alpha xy^T = \left(A + \alpha x(Py)^T\right)P = \tilde{Q}\tilde{R}$$

The IMSL routine PERMU (page 1138) is used to compute Py. As a check

$$\tilde{Q}\tilde{R}$$

is computed and printed. It can also be obtained from $A + xy^T$ by permuting its columns using the order given by IPVT.

```
C                                 Declare variables
      INTEGER    LDA, LDAQR, LDQ, LDQNEW, LDQR, LDR, LDRNEW, NCOL, NROW
      PARAMETER  (NCOL=3, NROW=4, LDA=NROW, LDAQR=NROW, LDQ=NROW,
     &           LDQNEW=NROW, LDQR=NROW, LDR=NROW, LDRNEW=NROW)
C
      INTEGER    IPATH, IPVT(NCOL), J, MIN0
      REAL       A(LDA,NCOL), ALPHA, AQR(LDAQR,NCOL), CONORM(NCOL),
     &           Q(LDQ,NROW), QNEW(LDQNEW,NROW), QR(LDQR,NCOL),
     &           QRAUX(NCOL), R(LDR,NCOL), RNEW(LDRNEW,NCOL), W(NROW),
     &           Y(NCOL)
      LOGICAL    PIVOT
      INTRINSIC  MIN0
      EXTERNAL   ISET, LQERR, LQRRR, LUPQR, MRRRR, PERMU, SCOPY, SSET,
     &           WRIRN, WRRRN
C
C                                 Set values for A
C
C                                 A = (  1    2     4   )
C                                     (  1    4    16   )
C                                     (  1    6    36   )
C                                     (  1    8    64   )
C
      DATA A/4*1.0, 2.0, 4.0, 6.0, 8.0, 4.0, 16.0, 36.0, 64.0/
C                                 Set values for W and Y
      DATA W/1., 2., 3., 4./
      DATA Y/3., 2., 1./
C
C                                 QR factorization
C                                 Set IPVT = 0 (all columns free)
      CALL ISET (NCOL, 0, IPVT, 1)
      PIVOT = .TRUE.
      CALL LQRRR (NROW, NCOL, A, LDA, PIVOT, IPVT, QR, LDQR, QRAUX,
     &            CONORM)
C                                 Accumulate Q
      CALL LQERR (NROW, NCOL, QR, LDQR, QRAUX, Q, LDQ)
C                                 Permute Y
      CALL PERMU (NCOL, Y, IPVT, 1, Y)
C                                 R is the upper trapezoidal part of QR
      CALL SSET (NROW*NCOL, 0.0, R, 1)
      DO 10  J=1, NCOL
         CALL SCOPY (MIN0(J,NROW), QR(1,J), 1, R(1,J), 1)
   10 CONTINUE
C                                 Update Q and R
      ALPHA = 1.0
      IPATH = 01
      CALL LUPQR (NROW, NCOL, ALPHA, W, Y, Q, LDQ, R, LDR, IPATH,
     &            QNEW, LDQNEW, RNEW, LDRNEW)
C                                 Compute AQR = Q*R
      CALL MRRRR (NROW, NROW, QNEW, LDQNEW, NROW, NCOL, RNEW, LDRNEW,
     &            NROW, NCOL, AQR, LDAQR)
C                                 Print results
      CALL WRIRN ('IPVT', 1, NCOL, IPVT, 1, 0)
      CALL WRRRN ('QNEW', NROW, NROW, QNEW, LDQNEW, 0)
      CALL WRRRN ('RNEW', NROW, NCOL, RNEW, LDRNEW, 0)
      CALL WRRRN ('QNEW*RNEW', NROW, NCOL, AQR, LDAQR, 0)
      END
```

Output

```
   IPVT
 1   2   3
 3   2   1

             QNEW
         1         2         3         4
1  -0.0620  -0.5412   0.8082  -0.2236
2  -0.2234  -0.6539  -0.2694   0.6708
3  -0.4840  -0.3379  -0.4490  -0.6708
4  -0.8438   0.4067   0.2694   0.2236
```

```
          RNEW
        1        2        3
1  -80.59   -21.34   -17.62
2    0.00    -4.94    -4.83
3    0.00     0.00     0.36
4    0.00     0.00     0.00

         QNEW*RNEW
        1        2        3
1    5.00     4.00     4.00
2   18.00     8.00     7.00
3   39.00    12.00    10.00
4   68.00    16.00    13.00
```

LCHRG/DLCHRG (Single/Double precision)

Compute the Cholesky decomposition of a symmetric positive semidefinite matrix with optional column pivoting.

Usage

CALL LCHRG (N, A, LDA, PIVOT, IPVT, FAC, LDFAC)

Arguments

N — Order of the matrix A. (Input)

A — N by N symmetric positive semidefinite matrix to be decomposed. (Input)
Only the upper triangle of A is referenced.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

PIVOT — Logical variable. (Input)
PIVOT = .TRUE. means column pivoting is done. PIVOT = .FALSE. means no pivoting is done.

IPVT — Integer vector of length N containing information that controls the selection of the pivot columns. (Input/Output)
On input, if IPVT(*K*) > 0, then the *K*-th column of A is an initial column; if IPVT(*K*) = 0, then the *K*-th column of A is a free column; if IPVT(*K*) < 0, then the *K*-th column of A is a final column. See Comments. On output, IPVT(*K*) contains the index of the diagonal element of A that was moved into the *K*-th position. IPVT is only referenced when PIVOT is equal to .TRUE..

FAC — N by N matrix containing the Cholesky factor of the permuted matrix in its upper triangle. (Output)
If A is not needed, A and FAC can share the same storage locations.

LDFAC — Leading dimension of FAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Informational error

Type	Code	
4	1	The input matrix is not positive semidefinite.

2. Before the decomposition is computed, initial elements are moved to the leading part of A and final elements to the trailing part of A. During the decomposition only rows and columns corresponding to the free elements are moved. The result of the decomposition is an upper triangular matrix R and a permutation matrix P that satisfy $P^T AP = R^T R$, where P is represented by IPVT.

3. LCHRG can be used together with subroutines PERMU and LSLDS to solve the positive semidefinite linear system AX = B with the solution X overwriting the right-hand side B as follows:

```
CALL ISET  (N, 0, IPVT, 1)
CALL LCHRG (N, A, LDA, TOL, .TRUE., IPVT, FAC,
            LDFAC, IRANK)
CALL PERMU (N, B, IPVT, 1, B)
CALL LSLDS (N, FAC, LDFAC, B, B)
CALL PERMU (N, B, IPVT, 2, B)
```

Algorithm

Routine LCHRG is based on the LINPACK routine SCHDC; see Dongarra et al. (1979).

Before the decomposition is computed, initial elements are moved to the leading part of A and final elements to the trailing part of A. During the decomposition only rows and columns corresponding to the free elements are moved. The result of the decomposition is an upper triangular matrix R and a permutation matrix P that satisfy $P^T AP = R^T R$, where P is represented by IPVT.

Example

Routine LCHRG can be used together with the IMSL routines PERMU (page 1138) and LFSDS (page 65) to solve a positive definite linear system $Ax = b$. Since $A = PR^T RP^{-1}$, the system $Ax = b$ is equivalent to $R^T R(P^{-1} x) = P^{-1} b$. LFSDS is used to solve $R^T Ry = P^{-1} b$ for y. The routine PERMU is used to compute both $P^{-1} b$ and $x = Py$.

```
C                                 Declare variables
      PARAMETER  (N=3, LDA=N, LDFAC=N)
      INTEGER    IPVT(N)
      REAL       A(LDA,N), FAC(LDFAC,N), B(N), X(N)
      LOGICAL    PIVOT
C
C                                 Set values for A and B
C
C                                 A = (   1   -3   2  )
```

```
C                                    (  -3   10  -5  )
C                                    (   2   -5   6  )
C
C                               B = (  27  -78  64  )
C
      DATA A/1.,-3.,2.,-3.,10.,-5.,2.,-5.,6./
      DATA B/27.,-78.,64./
C                               Pivot using all columns
      PIVOT = .TRUE.
      CALL ISET (N, 0, IPVT, 1)
C                               Compute Cholesky factorization
      CALL LCHRG (N, A, LDA, PIVOT, IPVT, FAC, LDFAC)
C                               Permute B and store in X
      CALL PERMU (N, B, IPVT, 1, X)
C                               Solve for X
      CALL LFSDS (N, FAC, LDFAC, X, X)
C                               Inverse permutation
      CALL PERMU (N, X, IPVT, 2, X)
C                               Print X
      CALL WRRRN (’X’, 1, N, X, 1, 0)
C
      END
```

Output

```
           X
   1       2       3
1.000  -4.000   7.000
```

LUPCH/DLUPCH (Single/Double precision)

Update the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is added.

Usage

```
CALL LUPCH (N, R, LDR, X, RNEW, LDRNEW, CS, SN)
```

Arguments

N — Order of the matrix.   (Input)

R — N by N upper triangular matrix containing the upper triangular factor to be updated.   (Input)
Only the upper triangle of R is referenced.

LDR — Leading dimension of R exactly as specified in the dimension statement of the calling program.   (Input)

X — Vector of length N determining the rank-one matrix to be added to the factorization $R^T R$.   (Input)

RNEW — N by N upper triangular matrix containing the updated triangular factor of $R^T R + XX^T$.   (Output)

Only the upper triangle of RNEW is referenced. If R is not needed, R and RNEW can share the same storage locations.

LDRNEW — Leading dimension of RNEW exactly as specified in the dimension statement of the calling program. (Input)

CS — Vector of length N containing the cosines of the rotations. (Output)

SN — Vector of length N containing the sines of the rotations. (Output)

Algorithm

The routine LUPCH is based on the LINPACK routine SCHUD; see Dongarra et al. (1979).

The Cholesky factorization of a matrix is $A = R^T R$, where R is an upper triangular matrix. Given this factorization, LUPCH computes the factorization

$$A + xx^T = \tilde{R}^T \tilde{R}$$

In the program

$$\tilde{R}$$

is called RNEW.

LUPCH determines an orthogonal matrix U as the product $G_N \ldots G_1$ of Givens rotations, such that

$$U \begin{bmatrix} R \\ x^T \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}$$

By multiplying this equation by its transpose, and noting that $U^T U = I$, the desired result

$$R^T R + xx^T = \tilde{R}^T \tilde{R}$$

is obtained.

Each Givens rotation, G_i, is chosen to zero out an element in x^T. The matrix G_i is $(N + 1) \times (N + 1)$ and has the form

$$G_i = \begin{bmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & c_i & 0 & s_i \\ 0 & 0 & I_{N-i} & 0 \\ 0 & -s_i & 0 & c_i \end{bmatrix}$$

where I_k is the identity matrix of order k and $c_i = \cos\theta_i =$ CS(I), $s_i = \sin\theta_i =$ SN(I) for some θ_i.

Example

A linear system $Az = b$ is solved using the Cholesky factorization of A. This factorization is then updated and the system $(A + xx^T)\ z = b$ is solved using this updated factorization.

```
C                                  Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), FAC(LDFAC,LDFAC), FACNEW(LDFAC,LDFAC),
     &           X(N), B(N), CS(N), SN(N), Z(N)
C
C                                  Set values for A
C                                  A = (  1.0  -3.0   2.0)
C                                      ( -3.0  10.0  -5.0)
C                                      (  2.0  -5.0   6.0)
C
      DATA A/1.0, -3.0, 2.0, -3.0, 10.0, -5.0, 2.0, -5.0, 6.0/
C
C                                  Set values for X and B
      DATA X/3.0, 2.0, 1.0/
      DATA B/53.0, 20.0, 31.0/
C                                  Factor the matrix A
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
C                                  Solve the original system
      CALL LFSDS (N, FAC, LDFAC, B, Z)
C                                  Print the results
      CALL WRRRN ('FAC', N, N, FAC, LDFAC, 1)
      CALL WRRRN ('Z', 1, N, Z, 1, 0)
C                                  Update the factorization
      CALL LUPCH (N, FAC, LDFAC, X, FACNEW, LDFAC, CS, SN)
C                                  Solve the updated system
      CALL LFSDS (N, FACNEW, LDFAC, B, Z)
C                                  Print the results
      CALL WRRRN ('FACNEW', N, N, FACNEW, LDFAC, 1)
      CALL WRRRN ('Z', 1, N, Z, 1, 0)
C
      END
```

Output

```
         FAC
        1       2       3
1   1.000  -3.000   2.000

2           1.000   1.000
3                   1.000
          Z
     1        2        3
1860.0    433.0   -254.0

      FACNEW
     1       2       3
1   3.162   0.949   1.581
2           3.619  -1.243
3                  -1.719
```

```
          Z
    1        2        3
4.000    1.000    2.000
```

LDNCH/DLDNCH (Single/Double precision)

Downdate the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is removed.

Usage

`CALL LDNCH (N, R, LDR, X, RNEW, LDRNEW, CS, SN)`

Arguments

N — Order of the matrix. (Input)

R — `N` by `N` upper triangular matrix containing the upper triangular factor to be downdated. (Input)
Only the upper triangle of `R` is referenced.

LDR — Leading dimension of `R` exactly as specified in the dimension statement of the calling program. (Input)

X — Vector of length `N` determining the rank-one matrix to be subtracted from the factorization $R^T R$. (Input)

RNEW — `N` by `N` upper triangular matrix containing the downdated triangular factor of $R^T R - X X^T$. (Output)
Only the upper triangle of `RNEW` is referenced. If `R` is not needed, `R` and `RNEW` can share the same storage locations.

LDRNEW — Leading dimension of `RNEW` exactly as specified in the dimension statement of the calling program. (Input)

CS — Vector of length `N` containing the cosines of the rotations. (Output)

SN — Vector of length `N` containing the sines of the rotations. (Output)

Comments

Informational error

Type	Code	
4	1	$R^T R - X X^T$ is not positive definite. `R` cannot be downdated.

Algorithm

The routine `LDNCH` is based on the LINPACK routine `SCHDD`; see Dongarra et al. (1979).

The Cholesky factorization of a matrix is $A = R^T R$, where R is an upper triangular matrix. Given this factorization, `LDNCH` computes the factorization

$$A - xx^T = \tilde{R}^T \tilde{R}$$

In the program

$$\tilde{R}$$

is called `RNEW`. This is not always possible, since $A - xx^T$ may not be positive definite.

`LDNCH` determines an orthogonal matrix U as the product $G_N \ldots G_1$ of Givens rotations, such that

$$U \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ x^T \end{bmatrix}$$

By multiplying this equation by its transpose and noting that $U^T U = I$, the desired result

$$R^T R - xx^T = \tilde{R}^T \tilde{R}$$

is obtained.

Let a be the solution of the linear system $R^T a = x$ and let

$$\alpha = \sqrt{1 - \|a\|_2^2}$$

The Givens rotations, G_i, are chosen such that

$$G_1 \cdots G_N \begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The $G_{i,}$ are $(N + 1) \times (N + 1)$ matrices of the form

$$G_i = \begin{bmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & c_i & 0 & -s_i \\ 0 & 0 & I_{N-i} & 0 \\ 0 & s_i & 0 & c_i \end{bmatrix}$$

where I_k is the identity matrix of order k; and c_i= cosθ_i = `CS(I)`, s_i= sinθ_i = `SN(I)` for some θ_i.

The Givens rotations are then used to form

$$\tilde{R},\ G_1 \cdots G_N \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ \tilde{x}^T \end{bmatrix}$$

The matrix

$$\tilde{R}$$

is upper triangular and

$$\tilde{x} = x$$

because

$$x = \left(R^T 0\right)\begin{bmatrix} a \\ \alpha \end{bmatrix} = \left(R^T 0\right)U^T U\begin{bmatrix} a \\ \alpha \end{bmatrix} = \left(\tilde{R}^T \tilde{x}\right)\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \tilde{x}$$

Example

A linear system $Az = b$ is solved using the Cholesky factorization of A. This factorization is then downdated, and the system $(A - xx^T)z = b$ is solved using this downdated factorization.

```
C                                 Declare variables
      INTEGER    LDA, LDFAC, N
      PARAMETER  (LDA=3, LDFAC=3, N=3)
      REAL       A(LDA,LDA), FAC(LDFAC,LDFAC), FACNEW(LDFAC,LDFAC),
     &           X(N), B(N), CS(N), SN(N), Z(N)
C
C                                 Set values for A
C                                 A = ( 10.0   3.0   5.0)
C                                     (  3.0  14.0  -3.0)
C                                     (  5.0  -3.0   7.0)
C
      DATA A/10.0, 3.0, 5.0, 3.0, 14.0, -3.0, 5.0, -3.0, 7.0/
C
C                                 Set values for X and B
      DATA X/3.0, 2.0, 1.0/
      DATA B/53.0, 20.0, 31.0/
C                                 Factor the matrix A
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
C                                 Solve the original system
      CALL LFSDS (N, FAC, LDFAC, B, Z)
C                                 Print the results
      CALL WRRRN ('FAC', N, N, FAC, LDFAC, 1)
      CALL WRRRN ('Z', 1, N, Z, 1, 0)
C                                 Downdate the factorization
      CALL LDNCH (N, FAC, LDFAC, X, FACNEW, LDFAC, CS, SN)
C                                 Solve the updated system
      CALL LFSDS (N, FACNEW, LDFAC, B, Z)
C                                 Print the results
      CALL WRRRN ('FACNEW', N, N, FACNEW, LDFAC, 1)
      CALL WRRRN ('Z', 1, N, Z, 1, 0)
C
      END
```

Output

```
          FAC
        1       2       3
1   3.162   0.949   1.581
2           3.619  -1.243
3                   1.719
            Z
      1       2       3
  4.000   1.000   2.000

          FACNEW
        1       2       3
1   1.000  -3.000   2.000
2           1.000   1.000
3                   1.000

             Z
     1        2        3
1859.9    433.0   -254.0
```

LSVRR/DLSVRR (Single/Double precision)

Compute the singular value decomposition of a real matrix.

Usage

```
CALL LSVRR (NRA, NCA, A, LDA, IPATH, TOL, IRANK, S, U, LDU,
            V, LDV)
```

Arguments

NRA — Number of rows in the matrix `A`. (Input)

NCA — Number of columns in the matrix `A`. (Input)

A — `NRA` by `NCA` matrix whose singular value decomposition is to be computed. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

IPATH — Flag used to control the computation of the singular vectors. (Input)
`IPATH` has the decimal expansion `IJ` such that:
`I` = 0 means do not compute the left singular vectors;
`I` = 1 means return the `NCA` left singular vectors in `U`;
`I` = 2 means return only the min(`NRA`, `NCA`) left singular vectors in `U`;
`J` = 0 means do not compute the right singular vectors,
`J` = 1 means return the right singular vectors in `V`.

For example, `IPATH` = 20 means `I` = 2 and `J` = 0.

TOL — Scalar containing the tolerance used to determine when a singular value is negligible. (Input)
If `TOL` is positive, then a singular value σ_i considered negligible if $\sigma_i \le$ `TOL` . If

TOL is negative, then a singular value σ_i considered negligible if $\sigma_i \le$ |TOL| * ||A||$_\infty$. In this case, |TOL| generally contains an estimate of the level of the relative error in the data.

IRANK — Scalar containing an estimate of the rank of A.   (Output)

S — Vector of length min(NRA + 1, NCA) containing the singular values of A in descending order of magnitude in the first min(NRA, NCA) positions.   (Output)

U — NRA by NCU matrix containing the left singular vectors of A.   (Output)
NCU must be equal to NRA if I is equal to 1. NCU must be equal to min(NRA, NCA) if I is equal to 2. U will not be referenced if I is equal to zero. If NRA is less than or equal to NCU, then U can share the same storage locations as A. See Comments.

LDU — Leading dimension of U exactly as specified in the dimension statement of the calling program.   (Input)

V — NCA by NCA matrix containing the right singular vectors of A.   (Output)
V will not be referenced if J is equal to zero. V can share the same storage location as A, however, U and V cannot both coincide with A simultaneously.

LDV — Leading dimension of V exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   LSVRR NRA * NCA + NRA + NCA + max(NRA, NCA) – 1 units, or
   DLSVRR 2 * (NRA * NCA + NRA + NCA + max(NRA, NCA) – 1) units.

   Workspace may be explicitly provided, if desired, by use of L2VRR/DL2VRR. The reference is

   ```
   CALL L2VRR (NRA, NCA, A, LDA, IPATH, TOL, IRANK, S,
               U, LDU, V, LDV, ACOPY, WK)
   ```

   The additional arguments are as follows:

   ACOPY — Work vector of length NRA * NCA for the matrix A. If A is not needed, then A and ACOPY may share the same storage locations.

   WK — Work vector of length NRA + NCA + max(NRA, NCA) – 1.

2. Informational error

Type	Code	
4	1	Convergence cannot be achieved for all the singular values and their corresponding singular vectors.

3. When NRA is much greater than NCA, it might not be reasonable to store the whole matrix U. In this case, IPATH with I = 2 allows a singular value factorization of A to be computed in which only the first

NCA columns of U are computed, and in many applications those are all that are needed.

4. Integer Options with Chapter 10 Options Manager

16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2VRR the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSVRR. Additional memory allocation for FAC and option value restoration are done automatically in LSVRR. Users directly calling L2VRR can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSVRR or L2VRR. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSVRR temporarily replaces IVAL(2) by IVAL(1). The routine L2CRG computes the condition number if IVAL(2) = 2. Otherwise L2CRG skips this computation. LSVRR restores the option. Default values for the option are IVAL(*) = 1, 2.

Algorithm

The routine LSVRR is based on the LINPACK routine SSVDC; see Dongarra et al. (1979).

Let n = NRA (the number of rows in A) and let p = NCA (the number of columns in A). For any $n \times p$ matrix A, there exists an $n \times n$ orthogonal matrix U and a $p \times p$ orthogonal matrix V such that

$$U^T AV = \begin{cases} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} & \text{if } n \ge p \\ \begin{bmatrix} \Sigma\, 0 \end{bmatrix} & \text{if } n \le p \end{cases}$$

where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_m)$, and $m = \min(n, p)$. The scalars $\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_m \ge 0$ are called the *singular values* of A. The columns of U are called the *left singular vectors* of A. The columns of V are called the *right singular vectors* of A.

The estimated rank of A is the number of σ_k that is larger than a tolerance η. If τ is the parameter TOL in the program, then

$$\eta = \begin{cases} \tau & \text{if } \tau > 0 \\ |\tau|\, \|A\|_\infty & \text{if } \tau < 0 \end{cases}$$

Example

This example computes the singular value decomposition of a 6 × 4 matrix *A*. The matrices *U* and *V* containing the left and right singular vectors, respectively, and the diagonal of Σ, containing singular values, are printed. On some systems, the signs of some of the columns of *U* and *V* may be reversed.

```
C                                 Declare variables
      PARAMETER  (NRA=6, NCA=4, LDA=NRA, LDU=NRA, LDV=NCA)
      REAL       A(LDA,NCA), U(LDU,NRA), V(LDV,NCA), S(NCA)
C
C                                 Set values for A
C
C                                 A = (  1    2    1    4  )
C                                     (  3    2    1    3  )
C                                     (  4    3    1    4  )
C                                     (  2    1    3    1  )
C                                     (  1    5    2    2  )
C                                     (  1    2    2    3  )
C
      DATA A/1., 3., 4., 2., 1., 1., 2., 2., 3., 1., 5., 2., 3*1.,
     &       3., 2., 2., 4., 3., 4., 1., 2., 3./
C
C                                 Compute all singular vectors
      IPATH = 11
      TOL   = 10.*AMACH(4)
      CALL LSVRR(NRA, NCA, A, LDA, IPATH, TOL, IRANK, S, U, LDU, V, LDV)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT, *) 'IRANK = ', IRANK
      CALL WRRRN ('U', NRA, NRA, U, LDU, 0)
      CALL WRRRN ('S', 1, NCA, S, 1, 0)
      CALL WRRRN ('V', NCA, NCA, V, LDV, 0)
C
      END
```

Output

```
IRANK =   4
                             U
         1        2        3        4        5        6
1  -0.3805   0.1197   0.4391  -0.5654   0.0243  -0.5726
2  -0.4038   0.3451  -0.0566   0.2148   0.8089   0.1193
3  -0.5451   0.4293   0.0514   0.4321  -0.5723   0.0403
4  -0.2648  -0.0683  -0.8839  -0.2153  -0.0625  -0.3062
5  -0.4463  -0.8168   0.1419   0.3213   0.0621  -0.0799
6  -0.3546  -0.1021  -0.0043  -0.5458  -0.0988   0.7457

                S
      1       2       3       4
  11.49    3.27    2.65    2.09

                  V
         1        2        3        4
1  -0.4443   0.5555  -0.4354   0.5518
2  -0.5581  -0.6543   0.2775   0.4283
3  -0.3244  -0.3514  -0.7321  -0.4851
4  -0.6212   0.3739   0.4444  -0.5261
```

LSVCR/DLSVCR (Single/Double precision)

Compute the singular value decomposition of a complex matrix.

Usage

```
CALL LSVCR (NRA, NCA, A, LDA, IPATH, TOL, IRANK, S, U, LDU,
            V, LDV)
```

Arguments

NRA — Number of rows in the matrix `A`. (Input)

NCA — Number of columns in the matrix `A`. (Input)

A — Complex `NRA` by `NCA` matrix whose singular value decomposition is to be computed. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

IPATH — Integer flag used to control the computation of the singular vectors. (Input)
`IPATH` has the decimal expansion `IJ` such that:

`I`= 0 means do not compute the left singular vectors;
`I` = 1 means return the `NCA` left singular vectors in `U`;
`I` = 2 means return only the min(`NRA`, `NCA`) left singular vectors in `U`;
J = 0 means do not compute the right singular vectors;
J = 1 means return the right singular vectors in V.

For example, `IPATH` = 20 means `I` = 2 and `J` = 0.

TOL — Real scalar containing the tolerance used to determine when a singular value is negligible. (Input)
If `TOL` is positive, then a singular value `SI` is considered negligible if `SI` ≤ `TOL` .
If `TOL` is negative, then a singular value `SI` is considered negligible if
`SI` ≤ |`TOL`|`*`(Infinity norm of `A`). In this case |`TOL`| should generally contain an estimate of the level of relative error in the data.

IRANK — Integer scalar containing an estimate of the rank of `A`. (Output)

S — Complex vector of length min(`NRA` + 1, `NCA`) containing the singular values of `A` in descending order of magnitude in the first min(`NRA`, `NCA`) positions. (Output)

U — Complex `NRA` by `NRA` if `I` = 1 or `NRA` by min(`NRA`, `NCA`) if `I` = 2 matrix containing the left singular vectors of `A`. (Output)
`U` will not be referenced if `I` is equal to zero. If `NRA` is less than or equal to `NCA` or `IPATH` = 2, then `U` can share the same storage locations as `A`.

LDU — Leading dimension of `U` exactly as specified in the dimension statement of the calling program. (Input)

V — Complex NCA by NCA matrix containing the right singular vectors of A. (Output)
V will not be referenced if J is equal to zero. If NCA is less than or equal to NRA, then V can share the same storage locations as A; however U and V cannot both coincide with A simultaneously.

LDV — Leading dimension of V exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   LSVCR 2 * (NRA * NCA + NRA + NCA + max(NRA, NCA) – 1) units, or
   DLSVCR 4 * (NRA * NCA + NRA + NCA + max(NRA, NCA) – 1) units.

   Workspace may be explicitly provided, if desired, by use of L2VCR/DL2VCR. The reference is

   ```
   CALL L2VCR (NRA, NCA, A, LDA, IPATH, TOL, IRANK, S,
               U, LDU, V, LDV, ACOPY, WK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work vector of length NRA * NCA for the matrix A. If A is not needed, then A and ACOPY can share the same storage locations.

   WK — Complex work vector of length NRA + NCA + max(NRA, NCA) – 1.

2. Informational error

Type	Code	
4	1	Convergence cannot be achieved for all the singular values and their corresponding singular vectors.

3. When NRA is much greater than NCA, it might not be reasonable to store the whole matrix U. In this case IPATH with I = 2 allows a singular value factorization of A to be computed in which only the first NCA columns of U are computed, and in many applications those are all that are needed.

4. Integer Options with Chapter 10 Options Manager

   16 This option uses four values to solve memory bank conflict (access inefficiency) problems. In routine L2VCR the leading dimension of FAC is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in LSVCR. Additional memory allocation for FAC and option value restoration are done automatically in LSVCR. Users directly calling L2VCR can allocate additional space for FAC and set IVAL(3) and IVAL(4) so that memory bank conflicts no longer cause inefficiencies. There is no requirement that users change existing applications that use LSVCR or L2VCR. Default values for the option are IVAL(*) = 1, 16, 0, 1.

17 This option has two values that determine if the L_1 condition number is to be computed. Routine LSVCR temporarily replaces IVAL(2) by IVAL(1). The routine L2CCG computes the condition number if IVAL(2) = 2. Otherwise L2CCG skips this computation. LSVCR restores the option. Default values for the option are IVAL(*) = 1, 2

Algorithm

The IMSL routine LSVCR is based on the LINPACK routine CSVDC; see Dongarra et al. (1979).

Let n = NRA (the number of rows in A) and let p = NCA (the number of columns in A).For any $n \times p$ matrix A there exists an $n \times n$ orthogonal matrix U and a $p \times p$ orthogonal matrix V such that

$$U^T AV = \begin{cases} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} & \text{if } n \geq p \\ \left[\Sigma \, 0\right] & \text{if } n \leq p \end{cases}$$

where Σ = diag($\sigma_{1,}$ …, σ_m), and m = min(n, p). The scalars $\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$ are called the *singular values* of A. The columns of U are called the *left singular vectors* of A. The columns of V are called the *right singular vectors* of A.

The estimated rank of A is the number of σ_k which are larger than a tolerance η. If τ is the parameter TOL in the program, then

$$\eta = \begin{cases} \tau & \text{if } \tau > 0 \\ |\tau| \, \|A\|_\infty & \text{if } \tau < 0 \end{cases}$$

Example

This example computes the singular value decomposition of a 6 × 3 matrix A. The matrices U and V containing the left and right singular vectors, respectively, and the diagonal of Σ, containing singular values, are printed. On some systems, the signs of some of the columns of U and V may be reversed.

```
C                                 Declare variables
      PARAMETER  (NRA=6, NCA=3, LDA=NRA, LDU=NRA, LDV=NCA)
      COMPLEX    A(LDA,NCA), U(LDU,NRA), V(LDV,NCA), S(NCA)
C
C                                 Set values for A
C
C                                 A = (  1+2i    3+2i    1-4i  )
C                                     (  3-2i    2-4i    1+3i  )
C                                     (  4+3i   -2+1i    1+4i  )
```

```
C                                     (  2-1i    3+0i    3-1i  )
C                                     (  1-5i    2-5i    2+2i  )
C                                     (  1+2i    4-2i    2-3i  )
C
      DATA A/(1.0,2.0), (3.0,-2.0), (4.0,3.0), (2.0,-1.0), (1.0,-5.0),
     &       (1.0,2.0), (3.0,2.0), (2.0,-4.0), (-2.0,1.0), (3.0,0.0),
     &       (2.0,-5.0), (4.0,-2.0), (1.0,-4.0), (1.0,3.0), (1.0,4.0),
     &       (3.0,-1.0), (2.0,2.0), (2.0,-3.0)/
C
C                                 Compute all singular vectors
      IPATH = 11
      TOL   = 10.*AMACH(4)
      CALL LSVCR(NRA, NCA, A, LDA, IPATH, TOL, IRANK, S, U, LDU, V, LDV)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT, *) 'IRANK = ', IRANK
      CALL WRCRN ('U', NRA, NRA, U, LDU, 0)
      CALL WRCRN ('S', 1, NCA, S, 1, 0)
      CALL WRCRN ('V', NCA, NCA, V, LDV, 0)
C
      END
```

Output

```
IRANK =   3
                                      U
                  1                 2                  3                 4
1 ( 0.1968, 0.2186) ( 0.5011, 0.0217)  (-0.2007,-0.1003)  (-0.2036, 0.0405)
2 ( 0.3443,-0.3542) (-0.2933, 0.0248)  ( 0.1155,-0.2338)  (-0.2316, 0.0287)
3 ( 0.1457, 0.2307) (-0.5424, 0.1381)  (-0.4361,-0.4407)  ( 0.0281,-0.3088)
4 ( 0.3016,-0.0844) ( 0.2157, 0.2659)  (-0.0523,-0.0894)  ( 0.8617, 0.0223)
5 ( 0.2283,-0.6008) (-0.1325, 0.1433)  ( 0.3152,-0.0090)  (-0.0392,-0.0145)
6 ( 0.2876,-0.0350) ( 0.4377,-0.0400)  ( 0.0458,-0.6205)  (-0.2303, 0.0924)

                   5                  6
1  ( 0.4132,-0.0985)  (-0.6017, 0.1612)
2  (-0.5061, 0.0198)  (-0.5380,-0.0317)
3  ( 0.2043,-0.1853)  ( 0.1012, 0.2132)
4  (-0.1272,-0.0866)  (-0.0808,-0.0266)
5  ( 0.6482,-0.1033)  ( 0.0995,-0.0837)
6  (-0.1412, 0.1121)  ( 0.4897,-0.0436)

                       S
              1                2                3
( 11.77,  0.00)  (  9.30,  0.00)  (  4.99,  0.00)

                            V
                   1                  2                  3
1  ( 0.6616, 0.0000)  (-0.2651, 0.0000)  (-0.7014, 0.0000)
2  ( 0.7355, 0.0379)  ( 0.3850,-0.0707)  ( 0.5482, 0.0624)
3  ( 0.0507,-0.1317)  ( 0.1724, 0.8642)  (-0.0173,-0.4509)
```

LSGRR/DLSGRR (Single/Double precision)

Compute the generalized inverse of a real matrix.

Usage

```
CALL LSGRR (NRA, NCA, A, LDA, TOL, IRANK, GINVA, LDGINV)
```

Arguments

NRA — Number of rows in the matrix A. (Input)

NCA — Number of columns in the matrix A. (Input)

A — NRA by NCA matrix whose generalized inverse is to be computed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

TOL — Scalar containing the tolerance used to determine when a singular value (from the singular value decomposition of A) is negligible. (Input)
If TOL is positive, then a singular value σ_i considered negligible if $\sigma_i \le$ TOL . If TOL is negative, then a singular value σ_i considered negligible if $\sigma_i \le$ |TOL| * $||A||_\infty$. In this case, |TOL| generally contains an estimate of the level of the relative error in the data.

IRANK — Scalar containing an estimate of the rank of A. (Output)

GINVA — NCA by NRA matrix containing the generalized inverse of A. (Output)

LDGINV — Leading dimension of GINVA exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   LSGRR NRA * NCA + NRA^2 + NCA^2 + min(NRA + 1, NCA) + NRA + NCA + max(NRA, NCA) – 1 units, or

   DLSGRR 2 * (NRA * NCA + NRA^2 + NCA^2 + min (NRA + 1, NCA) + min (NRA + 1, NCA) + NRA + NCA + max(NRA, NCA) – 1) units.

   Workspace may be explicitly provided, if desired, by use of L2GRR/DL2GRR. The reference is

   ```
   CALL L2GRR (NRA, NCA, A, LDA, TOL, IRANK, GINVA,
               LDGINV, WKA, WK)
   ```

   The additional arguments are as follows:

WKA — Work vector of length `NRA * NCA` used as workspace for the matrix `A`. If `A` is not needed, `WKA` and `A` can share the same storage locations.

WK — Work vector of length `LWK` where `LWK` is equal to $\texttt{NRA}^2 + \texttt{NCA}^2 +$ min(`NRA` + 1, `NCA`) + `NRA` + `NCA` + max(`NRA`, `NCA`) – 2.

2. Informational error

Type	Code	
4	1	Convergence cannot be achieved for all the singular values and their corresponding singular vectors.

Algorithm

Let k = `IRANK`, the rank of A; let n = `NRA`, the number of rows in A; let p = `NCA`, the number of columns in A; and let

$$A^{\dagger} = \mathrm{GINV}$$

be the generalized inverse of A.

To compute the *Moore-Penrose generalized inverse*, the routine `LSVRR` (page 307) is first used to compute the singular value decomposition of A. A singular value decomposition of A consists of an $n \times n$ orthogonal matrix U, a $p \times p$ orthogonal matrix V and a diagonal matrix $\Sigma = \mathrm{diag}(\sigma_{1,}\ldots, \sigma_m)$, $m = \min(n, p)$, such that $U^T AV = [\Sigma, 0]$ if $n \le p$ and $U^T AV = [\Sigma, 0]^T$ if $n \ge p$. Only the first p columns of U are computed. The rank k is estimated by counting the number of nonnegligible σ_i.

The matrices U and V can be partitioned as $U = (U_1, U_2)$ and $V = (V_1, V_2)$ where both U_1 and V_1 are $k \times k$ matrices. Let $\Sigma_1 = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$. The Moore-Penrose generalized inverse of A is

$$A^{\dagger} = \mathrm{V}_1 \Sigma_1^{-1} U_1^T$$

Example

This example computes the generalized inverse of a 3×2 matrix A. The rank k = `IRANK` and the inverse

$$A^{\dagger} = \mathrm{GINV}$$

are printed.

```
C                                 Declare variables
      PARAMETER  (NRA=3, NCA=2, LDA=NRA, LDGINV=NCA)
      REAL       A(LDA,NCA), GINV(LDGINV,NRA)
C
C                                 Set values for A
C
C                                 A = (   1    0   )
C                                     (   1    1   )
C                                     ( 100  -50   )
```

```
C
      DATA A/1., 1., 100., 0., 1., -50./
C
C                                Compute generalized inverse
      TOL = 10.*AMACH(4)
      CALL LSGRR (NRA, NCA, A, LDA, TOL, IRANK, GINV, LDGINV)
C                                Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT, *) ’IRANK = ’, IRANK
      CALL WRRRN (’GINV’, NCA, NRA, GINV, LDGINV, 0)
C
      END
```

Output

```
IRANK =   2
             GINV
         1         2         3
1   0.1000    0.3000    0.0060
2   0.2000    0.6000   -0.0080
```

Chapter 2: Eigensystem Analysis

Routines

2.1. Eigenvalues and (Optionally) Eigenvectors of $Ax = \lambda x$

2.1.1 Real General Problem $Ax = \lambda x$

2.1.2 Complex General Problem $Ax = \lambda x$

2.1.3 Real Symmetric Problem $Ax = \lambda x$

2.1.4 Real Band Symmetric Matrices in Band Storage Mode

Usage Notes

This chapter includes routines for linear eigensystem analysis. Many of these are for matrices with special properties. Some routines compute just a portion of the eigensystem. Use of the appropriate routine can substantially reduce computing time and storage requirements compared to computing a full eigensystem for a general complex matrix.

An ordinary linear eigensystem problem is represented by the equation $Ax = \lambda x$ where A denotes an $n \times n$ matrix. The value λ is an *eigenvalue* and $x \neq 0$ is the corresponding *eigenvector*. The eigenvector is determined up to a scalar factor. In all routines, we have chosen this factor so that x has Euclidean length with value one, and the component of x of smallest index and largest magnitude is positive. In case x is a complex vector, this largest component is real and positive.

In contrast to Version 1.1 of the IMSL Libraries, in Version 2.0 the eigenvalues and corresponding eigenvectors are sorted and returned in the order of largest to smallest complex magnitude. Users who require access to specific eigenvalues may need to alter their application codes to adjust for this change. If the order returned by Version 1.1 eigensystem codes is required, a use of Level 2 routines can replace the use of Level 1 routines. The Level 2 routines are documented in Version 1.1.

For example, the single-precision routine for computing the complex eigenvalues of a non-symmetric real matrix should now be obtained with the statement:

```
CALL E2LRG (N,A,LDA,EVAL,ACOPY,RWK)
```

This will replace the Level 1 statement:

```
CALL EVLRG (N,A,LDA,EVAL)
```

The arrays `A(*,*)`, `EVAL(*)`, `ACOPY(*)`, and `RWK(*)` are documented on page 293 of Version 1.1 (IMSL 1989).

Similar comments hold for the use of the remaining Level 1 routines in the following tables in those cases where the second character of the Level 2 routine name is no longer the character "2".

A generalized linear eigensystem problem is represented by $Ax = \lambda Bx$ where A and B are $n \times n$ matrices. The value λ is an eigenvalue, and x is the corresponding eigenvector. The eigenvectors are normalized in the same manner as for the ordinary eigensystem problem. The linear eigensystem routines have names that begin with the letter “`E`”. The generalized linear eigensystem routines have names that begin with the letter “`G`”. This prefix is followed by a two-letter code for the type of analysis that is performed. That is followed by another two-letter suffix for the form of the coefficient matrix. The following tables summarize the names of the eigensystem routines.

Symmetric and Hermitian Eigensystems			
	Symmetric Full	**Symmetric Band**	**Hermitian Full**
All eigenvalues	EVLSF p. 337	EVLSB p. 351	EVLHF p. 367
All eigenvalues and eigenvectors	EVCSF p. 339	EVCSB p. 353	EVCHF p. 369
Extreme eigenvalues	EVASF p. 341	EVASB p. 356	EVAHF p. 372
Extreme eigenvalues and eigenvectors	EVESF p. 343	EVESB p. 358	EVEHF p.373
Eigenvalues in an interval	EVBSF p. 345	EVBSB p. 361	EVBHF p. 376
Eigenvalues and eigevectors in an interval	EVFSF p. 347	EVFSB p. 363	EVFHF p 379
Performance index	EPISF p. 350	EPISB p. 366	EPIHF p. 382

General Eigensystems				
	Real General	**Complex General**	**Real Hessenberg**	**Complex Hessenberg**
All eigenvalues	EVLRG p. 325	EVLCG p. 331	EVLRH p. 383	EVLCH p. 387
All eigenvalues and eigenvectors	EVCRG p. 327	EVCCG p. 333	EVCRH p. 385	EVCCH p. 388
Performance index	EPIRG p. 330	EPICG p. 336	EPIRG p. 330	EPICG p. 336

Generalized Eigensystems Ax = λBx			
	Real General	**Complex General**	**A Symmetric B Positive Definite**
All eigenvalues	GVLRG p. 391	GVLCG p. 398	GVLSP p. 405
All eigenvalues and eigenvectors	GVCRG p. 393	GVCCG p. 400	GVCSP p. 407
Performance index	GPIRG p. 396	GPICG p. 403	GPISP p. 409

Error Analysis and Accuracy

The remarks in this section are for the ordinary eigenvalue problem. Except in special cases, routines will not return the exact eigenvalue-eigenvector pair for the ordinary eigenvalue problem $Ax = \lambda x$. The computed pair

$$\tilde{x}, \tilde{\lambda}$$

is an exact eigenvector-eigenvalue pair for a "nearby" matrix $A + E$. Information about E is known only in terms of bounds of the form $\|E\|_2 \le f(n) \, \|A\|_2 \, \varepsilon$. The value of $f(n)$ depends on the algorithm but is typically a small fractional power of n. The parameter ε is the machine precision. By a theorem due to Bauer and Fike (see Golub and Van Loan [1989, page 342],

$$\min\left|\tilde{\lambda} - \lambda\right| \le \kappa(X)\|E\|_2 \qquad \text{for all } \lambda \text{ in } \sigma(\mathrm{A})$$

where σ (A) is the set of all eigenvalues of A (called the *spectrum* of A), X is the matrix of eigenvectors, $\| \cdot \|_2$ is the 2-norm, and $\kappa(X)$ is the condition number of X defined as $\kappa(X) = \| X \|_2 \, \| X^{-1} \|_2$. If A is a real symmetric or complex Hermitian matrix, then its eigenvector matrix X is respectively orthogonal or unitary. For these matrices,$\kappa(X) = 1$.

The eigenvalues

$$\tilde{\lambda}_j$$

and eigenvectors

$$\tilde{x}_j$$

computed by `EVC**` can be checked by computing their performance index τ using `EPI**`. The performance index is defined by Smith et al. (1976, pages 124–126) to be

$$\tau = \max_{1 \le j \le n} \frac{\left\| A\tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j \right\|_1}{10 n \varepsilon \left\| A \right\|_1 \left\| \tilde{x}_j \right\|_1}$$

No significance should be attached to the factor of 10 used in the denominator. For a real vector x, the symbol $\| x \|_1$ represents the usual 1-norm of x. For a complex vector x, the symbol $\| x \|_1$ is defined by

$$\|x\|_1 = \sum_{k=1}^{N} \left(\left|\Re x_k\right| + \left|\Im x_k\right| \right)$$

The performance index τ is related to the error analysis because

$$\left\| E\tilde{x}_j \right\|_2 \doteq \left\| A\tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j \right\|_2$$

where E is the "nearby" matrix discussed above.

While the exact value of τ is machine and precision dependent, the performance of an eigensystem analysis routine is defined as excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. This is an arbitrary definition, but large values of τ can serve as a warning that there is a blunder in the calculation. There are also similar routines `GPI**` to compute the performance index for generalized eigenvalue problems.

If the condition number $\kappa(X)$ of the eigenvector matrix X is large, there can be large errors in the eigenvalues even if τ is small. In particular, it is often difficult to recognize near multiple eigenvalues or unstable mathematical problems from numerical results. This facet of the eigenvalue problem is difficult to understand: A user often asks for the accuracy of an individual eigenvalue. This can be answered approximately by computing the *condition number of an individual eigenvalue*. See Golub and Van Loan (1989, pages 344-345). For matrices A such that the computed array of normalized eigenvectors X is invertible, the condition number of λ_j is $\kappa_j \equiv$ the Euclidean length of row j of the inverse matrix X^{-1} . Users can choose to compute this matrix with routine `LINCG`, page 43. An approximate bound for the accuracy of a computed eigenvalue is then given by $\kappa_j \varepsilon \| A \|$ To compute an approximate bound for the relative accuracy of an eigenvalue, divide this bound by $| \lambda_j |$.

Reformulating Generalized Eigenvalue Problems

The generalized eigenvalue problem $Ax = \lambda Bx$ is often difficult for users to analyze because it is frequently ill-conditioned. There are occasionally changes of variables that can be performed on the given problem to ease this ill-conditioning. Suppose that B is singular but A is nonsingular. Define the reciprocal $\mu = \lambda^{-1}$. Then, the roles of A and B are interchanged so that the reformulated problem $Bx = \mu Ax$ is solved. Those generalized eigenvalues $\mu_j = 0$ correspond to eigenvalues $\lambda_j = \infty$. The remaining

$$\lambda_j = \mu_j^{-1}$$

The generalized eigenvectors for λ_j correspond to those for μ_j. Other reformulations can be made: If B is nonsingular, the user can solve the ordinary eigenvalue problem $Cx \equiv B^{-1} Ax = \lambda x$. This is not recommended as a computational algorithm for two reasons. First, it is generally less efficient than solving the generalized problem directly. Second, the matrix C will be subject to perturbations due to ill-conditioning and rounding errors when computing $B^{-1}A$. Computing the condition numbers of the eigenvalues for C may, however, be helpful for analyzing the accuracy of results for the generalized problem.

There is another method that users can consider to reduce the generalized problem to an alternate ordinary problem. This technique is based on first computing a matrix decomposition $B = PQ$, where both P and Q are matrices that are “simple” to invert. Then, the given generalized problem is equivalent to the ordinary eigenvalue problem $Fy = \lambda y$. The matrix $F \equiv P^{-1}AQ^{-1}$. The unnormalized eigenvectors of the generalized problem are given by $x = Q^{-1}y$. An example of this reformulation is used in the case where A and B are real and symmetric with B positive definite. The IMSL routines `GVLSP`, page 405 and `GVCSP`, page 407, use $P = R^T$ and $Q = R$ where R is an upper triangular matrix obtained from a Cholesky decomposition, $B = R^TR$. The matrix $F = R^{-\mathrm{T}} AR^{-1}$ is symmetric and real. Computation of the eigenvalue-eigenvector expansion for F is based on routine `EVCSF`, page 339.

EVLRG/DEVLRG (Single/Double precision)

Compute all of the eigenvalues of a real matrix.

Usage

```
CALL EVLRG (N, A, LDA, EVAL)
```

Arguments

N — Order of the matrix.   (Input)

A — Real full matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

EVAL — Complex vector of length `N` containing the eigenvalues of `A` in decreasing order of magnitude.   (Output)

Comments

1. Automatic workspace usage is

   `EVLRG` `N`(`N` + 6) units, or

   `DEVLRG` 2`N`(`N` + 4) + 2*n* units.

   Workspace may be explicitly provided, if desired, by use of `E3LRG`/`DE3LRG`. The reference is

   ```
   CALL E3LRG (N, A, LDA, EVAL, ACOPY, WK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Real work array of length N^2. `A` and `ACOPY` may be the same, in which case the first N^2 elements of `A` will be destroyed.

   WK — Floating-point work array of size 4`N`.

   IWK — Integer work array of size 2`N`.

2. Informational error

Type	Code	
4	1	The iteration for an eigenvalue failed to converge.

3. Integer Options with Chapter 10 Options Manager

   1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E3LRG, the internal or working leading dimension of ACOPY is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVLRG . Additional memory allocation and option value restoration are automatically done in EVLRG. There is no requirement that users change existing applications that use EVLRG or E3LRG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5−8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVLRG.

Algorithm

Routine EVLRG computes the eigenvalues of a real matrix. The matrix is first balanced. Elementary or Gauss similarity transformations with partial pivoting are used to reduce this balanced matrix to a real upper Hessenberg matrix. A hybrid double−shifted LR−QR algorithm is used to compute the eigenvalues of the Hessenberg matrix, Watkins and Elsner (1990).

The balancing routine is based on the EISPACK routine BALANC. The reduction routine is based on the EISPACK routine ELMHES. See Smith et al. (1976) for the EISPACK routines. The LR−QR algorithm is based on software work of Watkins and Haag. Further details, some timing data, and credits are given in Hanson et al. (1990).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 85). The eigenvalues of this real matrix are computed and printed. The exact eigenvalues are known to be {4, 3, 2, 1}.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=4, LDA=N)
C
      REAL       A(LDA,N)
      COMPLEX    EVAL(N)
      EXTERNAL   EVLRG, WRCRN
C                                 Set values of A
C
C                                 A = ( -2.0    2.0    2.0    2.0  )
C                                     ( -3.0    3.0    2.0    2.0  )
C                                     ( -2.0    0.0    4.0    2.0  )
C                                     ( -1.0    0.0    0.0    5.0  )
```

```
      DATA A/-2.0, -3.0, -2.0, -1.0, 2.0, 3.0, 0.0, 0.0, 2.0, 2.0,
     &     4.0, 0.0, 2.0, 2.0, 2.0, 5.0/
C
C                                 Find eigenvalues of A
      CALL EVLRG (N, A, LDA, EVAL)
C                                 Print results
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
                              EVAL
              1                 2                 3                 4
( 4.000, 0.000)  ( 3.000, 0.000)  ( 2.000, 0.000)  ( 1.000, 0.000)
```

EVCRG/DEVCRG (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a real matrix.

Usage

```
CALL EVCRG (N, A, LDA, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix.   (Input)

A — Floating-point array containing the matrix.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

EVAL — Complex array of size N containing the eigenvalues of A in decreasing order of magnitude.   (Output)

EVEC — Complex array containing the matrix of eigenvectors.   (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   EVCRG 2N * (N + 1) + 8N units, or

   DEVCRG 4N * (N + 1) + 13N + N

   Workspace may be explicitly provided, if desired, by use of E8CRG/DE8CRG. The reference is:

   ```
   CALL E8CRG (N, A, LDA, EVAL, EVEC, LDEVEC, ACOPY,
               ECOPY WK,IWK)
   ```

The additional arguments are as follows:

ACOPY — Floating-point work array of size N by N. The arrays A and ACOPY may be the same, in which case the first N^2 elements of A will be destroyed. The array ACOPY can have its working row dimension increased from N to a larger value. An optional usage is required. See Item 3 below for further details.

ECOPY — Floating-point work array of default size N by N + 1. The working, leading dimension of ECOPY is the same as that for ACOPY. To increase this value, an optional usage is required. See Item 3 below for further details.

WK — Floating-point work array of size 6N.

IWK — Integer work array of size N.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge. No eigenvalues or eigenvectors are computed.

3. Integer Options with Chapter 10 Options Manager

1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E8CRG, the internal or working leading dimensions of ACOPY and ECOPY are both increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVCRG. Additional memory allocation and option value restoration are automatically done in EVCRG. There is no requirement that users change existing applications that use EVCRG or E8CRG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5–8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVCRG.

Algorithm

Routine EVCRG computes the eigenvalues and eigenvectors of a real matrix. The matrix is first balanced. Orthogonal similarity transformations are used to reduce the balanced matrix to a real upper Hessenberg matrix. The implicit double−shifted QR algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix. The eigenvectors are normalized such that each has Euclidean length of value one. The largest component is real and positive.

The balancing routine is based on the EISPACK routine BALANC. The reduction routine is based on the EISPACK routines ORTHES and ORTRAN. The QR algorithm routine is based on the EISPACK routine HQR2. See Smith et al.

(1976) for the EISPACK routines. Further details, some timing data, and credits are given in Hanson et al. (1990).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 82). The eigenvalues and eigenvectors of this real matrix are computed and printed. The performance index is also computed and printed. This serves as a check on the computations. For more details, see IMSL routine EPIRG, page 330.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDEVEC=N)

      INTEGER    NOUT
      REAL       PI
      COMPLEX    EVAL(N), EVEC(LDEVEC,N)

      REAL       A(LDA,N)
      EXTERNAL   EVCRG, UMACH, WRCRN, EPIRG
      REAL       EPIRG
C                                 Define values of A:
C
C                                 A = (  8.0   -1.0   -5.0  )
C                                     ( -4.0    4.0   -2.0  )
C                                     ( 18.0   -5.0   -7.0  )
C
      DATA A/8.0, -4.0, 18.0, -1.0, 4.0, -5.0, -5.0, -2.0, -7.0/
C
C                                 Find eigenvalues and vectors of A
      CALL EVCRG (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPIRG(N,N,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
                     EVAL
              1                 2                 3
( 2.000, 4.000)  ( 2.000,-4.000)  ( 1.000, 0.000)

                           EVEC
                   1                  2                  3
1  ( 0.3162, 0.3162)  ( 0.3162,-0.3162)  ( 0.4082, 0.0000)
2  ( 0.0000, 0.6325)  ( 0.0000,-0.6325)  ( 0.8165, 0.0000)
3  ( 0.6325, 0.0000)  ( 0.6325, 0.0000)  ( 0.4082, 0.0000)

Performance index =  0.037
```

EPIRG/DEPIRG (Single/Double precision)

Compute the performance index for a real eigensystem.

Usage

```
EPIRG(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs on which the performance index computation is based. (Input)

A — Matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Complex vector of length `NEVAL` containing eigenvalues of `A`. (Input)

EVEC — Complex `N` by `NEVAL` array containing eigenvectors of `A`. (Input)
The eigenvector corresponding to the eigenvalue `EVAL(J)` must be in the `J`-th column of `EVEC`.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

EPIRG — Performance index. (Output)

Comments

1. Automatic workspace usage is

   `EPIRG` 2`N` units, or
   `DEPIRG` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `E2IRG`/`DE2IRG`. The reference is

   ```
   E2IRG(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC, CWK)
   ```

   The additional argument is

   CWK — Complex work array of length `N`.

2. Informational errors

Type	Code	
3	1	The performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix is zero.

Algorithm

Let M = NEVAL, λ = EVAL, x_j = EVEC(*,J), the j-th column of EVEC. Also, let ε be the machine precision given by AMACH(4). The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| Ax_j - \lambda_j x_j \right\|_1}{10 N \varepsilon \left\| A \right\|_1 \left\| x_j \right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector v is

$$\left\| v \right\|_1 = \sum_{i=1}^{N} \left\{ \left| \Re v_i \right| + \left| \Im v_i \right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of EVCSF is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$.

The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124−125).

Example

For an example of EPIRG, see IMSL routine EVCRG, page 327.

EVLCG/DEVLCG (Single/Double precision)

Compute all of the eigenvalues of a complex matrix.

Usage

```
CALL EVLCG (N, A, LDA, EVAL)
```

Arguments

N — Order of the matrix A.   (Input)

A — Complex matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Complex vector of length N containing the eigenvalues of A in decreasing order of magnitude.   (Output)

Comments

1.   Automatic workspace usage is

EVLCG $2N^2 + 6N$ units, or

DEVLCG $4N^2 + 11N$ units.

Workspace may be explicitly provided, if desired, by use of E3LCG/DE3LCG. The reference is

```
CALL E3LCG (N, A, LDA, EVAL, ACOPY, RWK, CWK, IWK)
```

The additional arguments are as follows:

ACOPY — Complex work array of length N^2. A and ACOPY may be the same, in which case the first N^2 elements of A will be destroyed.

RWK — Work array of length N.

CWK — Complex work array of length 2N.

IWK — Integer work array of length N.

2. Informational error

Type	Code	
4	1	The iteration for an eigenvalue failed to converge.

3. Integer Options with Chapter 10 Options Manager

1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E3LCG, the internal or working, leading dimension of ACOPY is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL (4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVLCG . Additional memory allocation and option value restoration are automatically done in EVLCG. There is no requirement that users change existing applications that use EVLCG or E3LCG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5−8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVLCG.

Algorithm

Routine EVLCG computes the eigenvalues of a complex matrix. The matrix is first balanced. Unitary similarity transformations are used to reduce this balanced matrix to a complex upper Hessenberg matrix. The shifted QR algorithm is used to compute the eigenvalues of this Hessenberg matrix.

The balancing routine is based on the EISPACK routine CBAL. The reduction routine is based on the EISPACK routine CORTH. The QR routine used is based on the EISPACK routine COMQR2. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a `DATA` statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 115). The program computes the eigenvalues of this matrix.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=3, LDA=N)
C
      COMPLEX    A(LDA,N), EVAL(N)
      EXTERNAL   EVLCG, WRCRN
C                                 Set values of A
C
C                                 A = ( 1+2i    3+4i   21+22i)
C                                     (43+44i  13+14i  15+16i)
C                                     ( 5+6i    7+8i   25+26i)
C
      DATA A/(1.0,2.0), (43.0,44.0), (5.0,6.0), (3.0,4.0),
     &     (13.0,14.0), (7.0,8.0), (21.0,22.0), (15.0,16.0),
     &     (25.0,26.0)/
C
C                                 Find eigenvalues of A
      CALL EVLCG (N, A, LDA, EVAL)
C                                 Print results
      CALL WRCRN (’EVAL’, 1, N, EVAL, 1, 0)
      END
```

Output

```
                          EVAL
              1                 2                 3
( 39.78, 43.00)  (  6.70, -7.88)  ( -7.48,  6.88)
```

EVCCG/DEVCCG (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a complex matrix.

Usage

`CALL EVCCG (N, A, LDA, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrix `A`.   (Input)

A — Complex matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Complex vector of length `N` containing the eigenvalues of `A` in decreasing order of magnitude.   (Output)

EVEC — Complex matrix of order N. (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVCCG $2N^2 + 6N$ units, or
   DEVCCG $4N^2 + 11N$ units.

   Workspace may be explicitly provided, if desired, by use of E6CCG/DE6CCG. The reference is

   ```
   CALL E6CCG (N, A, LDA, EVAL, EVEC, LDEVEC, ACOPY,
               RWK, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work array of length N^2. The arrays A and ACOPY may be the same, in which case the first N^2 elements of A will be destroyed.

   RWK — Work array of length N.

   CWK — Complex work array of length 2N.

   IWK — Integer work array of length N.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge. No eigenvalues or eigenvectors are computed.

3. Integer Options with Chapter 10 Options Manager

   1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E6CCG, the internal or working leading dimensions of ACOPY and ECOPY are both increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVCCG. Additional memory allocation and option value restoration are automatically done in EVCCG. There is no requirement that users change existing applications that use EVCCG or E6CCG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5−8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVCCG.

Algorithm

Routine EVCCG computes the eigenvalues and eigenvectors of a complex matrix. The matrix is first balanced. Unitary similarity transformations are used to reduce this balanced matrix to a complex upper Hessenberg matrix. The QR algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix. The eigenvectors of the original matrix are computed by transforming the eigenvectors of the complex upper Hessenberg matrix.

The balancing routine is based on the EISPACK routine CBAL. The reduction routine is based on the EISPACK routine CORTH. The QR algorithm routine used is based on the EISPACK routine COMQR2. The back transformation routine is based on the EISPACK routine CBABK2 . See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 116). Its eigenvalues are known to be {1 + 5*i*, 2 + 6*i*, 3 + 7*i*, 4 + 8*i*}. The program computes the eigenvalues and eigenvectors of this matrix. The performance index is also computed and printed. This serves as a check on the computations; for more details, see IMSL routine EPICG, page 336.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=4, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       EPICG, PI
      COMPLEX    A(LDA,N), EVAL(N), EVEC(LDEVEC,N)
      EXTERNAL   EPICG, EVCCG, UMACH, WRCRN
C                                 Set values of A
C
C                                 A = (5+9i  5+5i  -6-6i -7-7i)
C                                     (3+3i  6+10i -5-5i -6-6i)
C                                     (2+2i  3+3i  -1+3i -5-5i)
C                                     (1+i   2+2i  -3-3i     4i)
C
      DATA A/(5.0,9.0), (3.0,3.0), (2.0,2.0), (1.0,1.0), (5.0,5.0),
     &     (6.0,10.0), (3.0,3.0), (2.0,2.0), (-6.0,-6.0), (-5.0,-5.0),
     &     (-1.0,3.0), (-3.0,-3.0), (-7.0,-7.0), (-6.0,-6.0),
     &     (-5.0,-5.0), (0.0,4.0)/
C
C                                 Find eigenvalues and vectors of A
      CALL EVCCG (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPICG(N,N,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN (’EVAL’, 1, N, EVAL, 1, 0)
      CALL WRCRN (’EVEC’, N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,’(/,A,F6.3)’) ’ Performance index = ’, PI
      END
```

Output

```
                              EVAL
              1                2                3                4
( 4.000, 8.000)  ( 3.000, 7.000)  ( 2.000, 6.000)  ( 1.000, 5.000)

                              EVEC
         1                  2                  3                  4
1 ( 0.5774, 0.0000) ( 0.5774, 0.0000) ( 0.3780, 0.0000) ( 0.7559, 0.0000)
2 ( 0.5774, 0.0000) ( 0.5773, 0.0000) ( 0.7559, 0.0000) ( 0.3780, 0.0000)
3 ( 0.5774, 0.0000) ( 0.0000, 0.0000) ( 0.3780, 0.0000) ( 0.3780, 0.0000)
4 ( 0.0000, 0.0000) ( 0.5774, 0.0000) ( 0.3780, 0.0000) ( 0.3780, 0.0000)

Performance index =  0.016
```

EPICG/DEPICG (Single/Double precision)

Compute the performance index for a complex eigensystem.

Usage

`EPICG(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs on which the performance index computation is based. (Input)

A — Complex matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Complex vector of length `N` containing the eigenvalues of `A`. (Input)

EVEC — Complex matrix of order `N` containing the eigenvectors of `A`. (Input) The `J`-th eigenvalue/eigenvector pair should be in `EVAL(J)` and in the `J`-th column of `EVEC`.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

EPICG — Performance index. (Output)

Comments

1. Automatic workspace usage is

   `EPICG` 2`N` units, or
   `DEPICG` 4`N` units.

   Workspace may be explicitly provided, if desired, by use of `E2ICG`/`DE2ICG`. The reference is

```
E2ICG(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC, WK)
```

The additional argument is

WK — Complex work array of length `N`.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix is zero.

Algorithm

Let M = `NEVAL`, λ = `EVAL`, x_j = `EVEC(*, J)`, the `j`-th column of `EVEC`. Also, let ε be the machine precision given by `AMACH`(4). The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| Ax_j - \lambda_j x_j \right\|_1}{10 N\varepsilon \left\| A \right\|_1 \left\| x_j \right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector v is

$$\left\| v \right\|_1 = \sum_{i=1}^{N} \left\{ \left| \Re v_i \right| + \left| \Im v_i \right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of `EVCSF` (page 339) is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124−125).

Example

For an example of `EPICG`, see IMSL routine `EVCCG` on page 333.

EVLSF/DEVLSF (Single/Double precision)

Compute all of the eigenvalues of a real symmetric matrix.

Usage

```
CALL EVLSF (N, A, LDA, EVAL)
```

Arguments

N — Order of the matrix `A`. (Input)

A — Real symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Real vector of length `N` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

Comments

1. Automatic workspace usage is

   `EVLSF` 3`N` units, or
   `DEVLSF` 4`N` + `N` units.

   Workspace may be explicitly provided, if desired, by use of `E4LSF`/`DE4LSF`. The reference is

   `CALL E4LSF (N, A, LDA, EVAL,WORK, IWORK)`

   The additional arguments are as follows:

   WORK — Work array of length 2`N`.

   IWORK — Integer array of length `N`.

2. Informational error

Type	Code	
3	1	The iteration for the eigenvalue failed to converge in 100 iterations before deflating.

Algorithm

Routine `EVLSF` computes the eigenvalues of a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. Then, an implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine `TRED2`. See Smith et al. (1976). The rational QR algorithm is called the PWK algorithm. It is given in Parlett (1980, page 169). Further details, some timing data, and credits are given in Hanson et al. (1990).

Example

In this example, the eigenvalues of a real symmetric matrix are computed and printed. This matrix is given by Gregory and Karney (1969, page 56).

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=4, LDA=N)
C
      REAL       A(LDA,N), EVAL(N)
      EXTERNAL   EVLSF, WRRRN
C                                 Set values of A
C
C                                 A = (  6.0    4.0    4.0    1.0)
C                                     (  4.0    6.0    1.0    4.0)
```

```
C                                    (  4.0    1.0    6.0    4.0)
C                                    (  1.0    4.0    4.0    6.0)
C
      DATA A /6.0, 4.0, 4.0, 1.0, 4.0, 6.0, 1.0, 4.0, 4.0, 1.0, 6.0,
     &        4.0, 1.0, 4.0, 4.0, 6.0 /
C
C                                Find eigenvalues of A
      CALL EVLSF (N, A, LDA, EVAL)
C                                Print results
      CALL WRRRN (’EVAL’, 1, N, EVAL, 1, 0)
      END
```

Output

```
              EVAL
    1       2       3       4
15.00    5.00    5.00   -1.00
```

EVCSF/DEVCSF (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a real symmetric matrix.

Usage

CALL EVCSF (N, A, LDA, EVAL, EVEC, LDEVEC)

Arguments

N — Order of the matrix A.   (Input)

A — Real symmetric matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Real vector of length N containing the eigenvalues of A in decreasing order of magnitude.   (Output)

EVEC — Real matrix of order N.   (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program.   (Input)

Comments

1. Automatic workspace usage is

   EVCSF   4N units, or
   DEVCSF 7N units.

   Workspace may be explicitly provided, if desired, by use of E5CSF/DE5CSF. The reference is

```
CALL E5CSF (N, A, LDA, EVAL, EVEC, LDEVEC, WORK,
            IWK)
```

The additional argument is

WORK — Work array of length 3`N`.

IWK — Integer array of length `N`.

2. Informational error

Type	Code	
3	1	The iteration for the eigenvalue failed to converge in 100 iterations before deflating.

Algorithm

Routine `EVCSF` computes the eigenvalues and eigenvectors of a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. These transformations are accumulated. An implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix. The eigenvectors are computed using the eigenvalues as perfect shifts, Parlett (1980, pages 169, 172). The reduction routine is based on the EISPACK routine TRED2. See Smith et al. (1976) for the EISPACK routines. Further details, some timing data, and credits are given in Hanson et al. (1990).

Example

The eigenvalues and eigenvectors of this real symmetric matrix are computed and printed. The performance index is also computed and printed. This serves as a check on the computations. For more details, see `EPISF` on page 350.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N), EPISF, EVAL(N), EVEC(LDEVEC,N), PI
      EXTERNAL   EPISF, EVCSF, UMACH, WRRRN
C
C                                 Set values of A
C
C                                 A = (  7.0   -8.0   -8.0)
C                                     ( -8.0  -16.0  -18.0)
C                                     ( -8.0  -18.0   13.0)
C
      DATA A/7.0, -8.0, -8.0, -8.0, -16.0, -18.0, -8.0, -18.0, 13.0/
C
C                                 Find eigenvalues and vectors of A
      CALL EVCSF (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPISF (N, N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT, '(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
           EVAL
      1       2       3
 -27.90   22.68    9.22

            EVEC
         1        2        3
1   0.2945  -0.2722   0.9161
2   0.8521  -0.3591  -0.3806
3   0.4326   0.8927   0.1262

Performance index =  0.044
```

EVASF/DEVASF (Single/Double precision)

Compute the largest or smallest eigenvalues of a real symmetric matrix.

Usage

```
CALL EVASF (N, NEVAL, A, LDA, SMALL, EVAL)
```

Arguments

N — Order of the matrix `A`.   (Input)

NEVAL — Number of eigenvalues to be computed.   (Input)

A — Real symmetric matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

SMALL — Logical variable.   (Input)
If `.TRUE.`, the smallest `NEVAL` eigenvalues are computed. If `.FALSE.`, the largest `NEVAL` eigenvalues are computed.

EVAL — Real vector of length `NEVAL` containing the eigenvalues of `A` in decreasing order of magnitude.   (Output)

Comments

1. Automatic workspace usage is

   `EVASF`   5`N` units, or
   `DEVASF` 9`N` units.

   Workspace may be explicitly provided, if desired, by use of `E4ASF`/`DE4ASF`. The reference is

   ```
   CALL E4ASF (N, NEVAL, A, LDA, SMALL, EVAL, WORK,
               IWK)
   ```

WORK — Work array of length `4N`.
IWK — Integer work array of length `N`.

2. Informational error

Type	Code	
3	1	The iteration for an eigenvalue failed to converge. The best estimate will be returned.

Algorithm

Routine `EVASF` computes the largest or smallest eigenvalues of a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. Then, an implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine `TRED2`. See Smith et al. (1976). The rational QR algorithm is called the PWK algorithm. It is given in Parlett (1980, page 169).

Example

In this example, the three largest eigenvalues of the computed Hilbert matrix $a_{ij} = 1/(i + j - 1)$ of order $N = 10$ are computed and printed.

```
C                                  Declare variables
      INTEGER    LDA, N, NEVAL
      PARAMETER  (N=10, NEVAL=3, LDA=N)
C
      INTEGER    I, J
      REAL       A(LDA,N), EVAL(NEVAL), REAL
      LOGICAL    SMALL
      INTRINSIC  REAL
      EXTERNAL   EVASF, WRRRN
C                                  Set up Hilbert matrix
      DO 20  J=1, N
         DO 10  I=1, N
            A(I,J) = 1.0/REAL(I+J-1)
   10    CONTINUE
   20 CONTINUE
C                                  Find the 3 largest eigenvalues
      SMALL = .FALSE.
      CALL EVASF (N, NEVAL, A, LDA, SMALL, EVAL)
C                                  Print results
      CALL WRRRN (’EVAL’, 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
       EVAL
    1       2       3
1.752   0.343   0.036
```

EVESF/DEVESF (Single/Double precision)

Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix.

Usage

```
CALL EVESF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVEC — Number of eigenvalues to be computed. (Input)

A — Real symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

SMALL — Logical variable. (Input)
If `.TRUE.`, the smallest `NEVEC` eigenvalues are computed. If `.FALSE.`, the largest `NEVEC` eigenvalues are computed.

EVAL — Real vector of length `NEVEC` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

EVEC — Real matrix of dimension `N` by `NEVEC`. (Output)
The `J`-th eigenvector, corresponding to `EVAL(J)`, is stored in the `J`-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   `EVESF` 10`N` units, or
   `DEVESF` 19`N` units.

   Workspace may be explicitly provided, if desired, by use of `E5ESF/DE5ESF`. The reference is

   ```
   CALL E5ESF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC,
               LDEVEC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 9`N`.

   IWK — Integer array of length `N`.

2. Informational errors

Type	Code	
3	1	The iteration for an eigenvalue failed to converge. The best estimate will be returned.
3	2	Inverse iteration did not converge. Eigenvector is not correct for the specified eigenvalue.
3	3	The eigenvectors have lost orthogonality.

Algorithm

Routine EVESF computes the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. Then, an implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix. Inverse iteration is used to compute the eigenvectors of the tridiagonal matrix. This is followed by orthogonalization of these vectors. The eigenvectors of the original matrix are computed by back transforming those of the tridiagonal matrix.

The reduction routine is based on the EISPACK routine TRED2. See Smith et al. (1976). The rational QR algorithm is called the PWK algorithm. It is given in Parlett (1980, page 169). The inverse iteration and orthogonalization computation is discussed in Hanson et al. (1990). The back transformation routine is based on the EISPACK routine TRBAK1.

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 55). The largest two eigenvalues and their eigenvectors are computed and printed. The performance index is also computed and printed. This serves as a check on the computations. For more details, see IMSL routine EPISF on page 350.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=4, LDA=N, LDEVEC=N)
C
      INTEGER    NEVEC, NOUT
      REAL       A(LDA,N), EPISF, EVAL(N), EVEC(LDEVEC,N), PI
      LOGICAL    SMALL
      EXTERNAL   EPISF, EVESF, UMACH, WRRRN
C
C                                 Set values of A
C
C                                 A = (  5.0    4.0    1.0    1.0)
C                                     (  4.0    5.0    1.0    1.0)
C                                     (  1.0    1.0    4.0    2.0)
C                                     (  1.0    1.0    2.0    4.0)
C
      DATA A/5.0, 4.0, 1.0, 1.0, 4.0, 5.0, 1.0, 1.0, 1.0, 1.0, 4.0,
     &     2.0, 1.0, 1.0, 2.0, 4.0/
C
C                                 Find eigenvalues and vectors of A
```

```
      NEVEC = 2
      SMALL = .FALSE.
      CALL EVESF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPISF(N,NEVEC,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN ('EVAL', 1, NEVEC, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, NEVEC, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
       EVAL
     1       2
 10.00    5.00

       EVEC
         1         2
 1   0.6325  -0.3162
 2   0.6325  -0.3162
 3   0.3162   0.6325
 4   0.3162   0.6325

 Performance index =  0.026
```

EVBSF/DEVBSF (Single/Double precision)

Compute selected eigenvalues of a real symmetric matrix.

Usage

`CALL EVBSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL)`

Arguments

N — Order of the matrix `A`.   (Input)

MXEVAL — Maximum number of eigenvalues to be computed.   (Input)

A — Real symmetric matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought.   (Input)

EHIGH — Upper limit of the interval in which the eigenvalues are sought.   (Input)

NEVAL — Number of eigenvalues found.   (Output)

EVAL — Real vector of length MXEVAL containing the eigenvalues of A in the interval (ELOW, EHIGH) in decreasing order of magnitude. (Output)
Only the first NEVAL elements of EVAL are significant.

Comments

1. Automatic workspace usage is

   EVBSF 6N units, or
   DEVBSF 11N units.

   Workspace may be explicitly provided, if desired, by use of E5BSF/DE5BSF. The reference is

   ```
   CALL E5BSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL,
               EVAL, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 5N.

   IWK — Integer work array of length 1N.

2. Informational error

Type	Code	
3	1	The number of eigenvalues in the specified interval exceeds MXEVAL. NEVAL contains the number of eigenvalues in the interval. No eigenvalues will be returned.

Algorithm

Routine EVBSF computes the eigenvalues in a given interval for a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. Then, an implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix. The reduction step is based on the EISPACK routine TRED1. See Smith et al. (1976). The rational QR algorithm is called the PWK algorithm. It is given in Parlett (1980, page 169).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 56). The eigenvalues of *A* are known to be −1, 5, 5 and 15. The eigenvalues in the interval [1.5, 5.5] are computed and printed. As a test, this example uses MXEVAL = 4. The routine EVBSF computes NEVAL, the number of eigenvalues in the given interval. The value of NEVAL is 2.

```
C                                 Declare variables
      INTEGER    LDA, MXEVAL, N
      PARAMETER  (MXEVAL=4, N=4, LDA=N)
C
      INTEGER    NEVAL, NOUT
      REAL       A(LDA,N), EHIGH, ELOW, EVAL(MXEVAL)
```

```
      EXTERNAL   EVBSF, UMACH, WRRRN
C
C                                 Set values of A
C
C                                 A = (  6.0    4.0    4.0    1.0)
C                                     (  4.0    6.0    1.0    4.0)
C                                     (  4.0    1.0    6.0    4.0)
C                                     (  1.0    4.0    4.0    6.0)
C
      DATA A/6.0, 4.0, 4.0, 1.0, 4.0, 6.0, 1.0, 4.0, 4.0, 1.0, 6.0,
     &     4.0, 1.0, 4.0, 4.0, 6.0/
C
C                                 Find eigenvalues of A
      ELOW  = 1.5
      EHIGH = 5.5
      CALL EVBSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I2)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
NEVAL =  2

     EVAL
    1        2
5.000   5.000
```

EVFSF/DEVFSF (Single/Double precision)

Compute selected eigenvalues and eigenvectors of a real symmetric matrix.

Usage

```
CALL EVFSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL,
            EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

MXEVAL — Maximum number of eigenvalues to be computed. (Input)

A — Real symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought. (Input)

EHIGH — Upper limit of the interval in which the eigenvalues are sought. (Input)

NEVAL — Number of eigenvalues found. (Output)

EVAL — Real vector of length MXEVAL containing the eigenvalues of A in the interval (ELOW, EHIGH) in decreasing order of magnitude. (Output)
Only the first NEVAL elements of EVAL are significant.

EVEC — Real matrix of dimension N by MXEVAL. (Output)
The J-th eigenvector corresponding to EVAL(J), is stored in the J-th column. Only the first NEVAL columns of EVEC are significant. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVFSF 10N units, or
   DEVFSF 18N + N units.

   Workspace may be explicitly provided, if desired, by use of E3FSF/DE3FSF. The reference is

   ```
   CALL E3FSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL,
               EVAL, EVEC, LDEVEC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 9N.

   IWK — Integer work array of length N.

2. Informational errors

Type	Code	
3	1	The number of eigenvalues in the specified range exceeds MXEVAL. NEVAL contains the number of eigenvalues in the range. No eigenvalues will be computed.
3	2	Inverse iteration did not converge. Eigenvector is not correct for the specified eigenvalue.
3	3	The eigenvectors have lost orthogonality.

Algorithm

Routine EVFSF computes the eigenvalues in a given interval and the corresponding eigenvectors of a real symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. Then, an implicit rational QR algorithm is used to compute the eigenvalues of this tridiagonal matrix. Inverse iteration is used to compute the eigenvectors of the tridiagonal matrix. This is followed by orthogonalization of these vectors. The eigenvectors of the original matrix are computed by back transforming those of the tridiagonal matrix.

The reduction step is based on the EISPACK routine TRED1. The rational QR algorithm is called the PWK algorithm. It is given in Parlett (1980, page 169). The inverse iteration and orthogonalization processes are discussed in Hanson et al. (1990). The transformation back to the users's input matrix is based on the EISPACK routine TRBAK1. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, *A* is set to the computed Hilbert matrix. The eigenvalues in the interval [0.001, 1] and their corresponding eigenvectors are computed and printed. This example uses MXEVAL = 3. The routine EVFSF computes the number of eigenvalues NEVAL in the given interval. The value of NEVAL is 2. The performance index is also computed and printed. For more details, see IMSL routine EPISF on page 350.

```
C                                  Declare variables
      INTEGER    LDA, LDEVEC, MXEVAL, N
      PARAMETER  (MXEVAL=3, N=3, LDA=N, LDEVEC=N)
C
      INTEGER    NEVAL, NOUT
      REAL       A(LDA,N), EHIGH, ELOW, EPISF, EVAL(MXEVAL),
     &           EVEC(LDEVEC,MXEVAL), PI
      EXTERNAL   EPISF, EVFSF, UMACH, WRRRN
C                                  Compute Hilbert matrix
      DO 20 J=1,N
         DO 10 I=1,N
            A(I,J) = 1.0/FLOAT(I+J-1)
   10    CONTINUE
   20 CONTINUE
C                                  Find eigenvalues and vectors
      ELOW  = 0.001
      EHIGH = 1.0
      CALL EVFSF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL, EVEC,
     &            LDEVEC)
C                                  Compute performance index
      PI = EPISF(N,NEVAL,A,LDA,EVAL,EVEC,LDEVEC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I2)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, NEVAL, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
NEVAL =  2

      EVAL
     1        2
0.1223   0.0027

        EVEC
         1        2
1  -0.5474  -0.1277
2   0.5283   0.7137
```

```
3   0.6490  -0.6887

Performance index =  0.008
```

EPISF/DEPISF (Single/Double precision)

Compute the performance index for a real symmetric eigensystem.

Usage

`EPISF(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs on which the performance index computation is based on. (Input)

A — Symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Vector of length `NEVAL` containing eigenvalues of `A`. (Input)

EVEC — `N` by `NEVAL` array containing eigenvectors of `A`. (Input)
The eigenvector corresponding to the eigenvalue `EVAL(J)` must be in the `J`-th column of `EVEC`.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

EPISF — Performance index. (Output)

Comments

1. Automatic workspace usage is

   `EPISF` `N` units, or
   `DEPISF` 2`N` units.

   Workspace may be explicitly provided, if desired, by use of `E2ISF/DE2ISF`. The reference is

   `E2ISF(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC, WORK)`

   The additional argument is

   WORK — Work array of length `N`.

   E2ISF — Performance Index.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix is zero.

Algorithm

Let M = `NEVAL`, λ = `EVAL`, x_j = `EVEC(*,J)`, the `j`-th column of `EVEC`. Also, let ε be the machine precision, given by `AMACH`(4) (page 1201). The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| Ax_j - \lambda_j x_j \right\|_1}{10 N \varepsilon \left\| A \right\|_1 \left\| x_j \right\|_1}$$

While the exact value of τ is highly machine dependent, the performance of `EVCSF` (page 339) is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124−125).

Example

For an example of `EPISF`, see routine `EVCSF`, page 339.

EVLSB/DEVLSB (Single/Double precision)

Compute all of the eigenvalues of a real symmetric matrix in band symmetric storage mode.

Usage

```
CALL EVLSB (N, A, LDA, NCODA, EVAL)
```

Arguments

N — Order of the matrix `A`. (Input)

A — Band symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

NCODA — Number of codiagonals in `A`. (Input)

EVAL — Vector of length `N` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

Comments

1. Automatic workspace usage is

   EVLSB N(NCODA + 2) units, or
   DEVLSB 2N(NCODA + 2) units.

   Workspace may be explicitly provided, if desired, by use of E3LSB/DE3LSB. The reference is

   CALL E3LSB (N, A, LDA, NCODA, EVAL, ACOPY, WK)

   The additional arguments are as follows:

   ACOPY — Work array of length N(NCODA + 1). The arrays A and ACOPY may be the same, in which case the first N(NCODA + 1) elements of A will be destroyed.

   WK — Work array of length N.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

Algorithm

Routine EVLSB computes the eigenvalues of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. The implicit QL algorithm is used to compute the eigenvalues of the resulting tridiagonal matrix.

The reduction routine is based on the EISPACK routine BANDR; see Garbow et al. (1977). The QL routine is based on the EISPACK routine IMTQL1; see Smith et al. (1976).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 77). The eigenvalues of this matrix are given by

$$\lambda_k = \left(1 - 2\cos\frac{k\pi}{N+1}\right)^2 - 3$$

Since the eigenvalues returned by EVLSB are in decreasing magnitude, the above formula for $k = 1, \ldots, N$ gives the the values in a different order. The eigenvalues of this real band symmetric matrix are computed and printed.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N, NCODA
      PARAMETER  (N=5, NCODA=2, LDA=NCODA+1, LDEVEC=N)
C
      REAL       A(LDA,N), EVAL(N)
      EXTERNAL   EVLSB, WRRRN
C                                 Define values of A:
C                                 A = (-1  2  1       )
```

```
C                                      ( 2  0  2  1    )
C                                      ( 1  2  0  2  1 )
C                                      (    1  2  0  2 )
C                                      (       1  2 -1 )
C                                  Represented in band symmetric
C                                  form this is:
C                                  A = ( 0  0  1  1  1 )
C                                      ( 0  2  2  2  2 )
C                                      (-1  0  0  0 -1 )
C
      DATA A/0.0, 0.0, -1.0, 0.0, 2.0, 0.0, 1.0, 2.0, 0.0, 1.0, 2.0,
     &     0.0, 1.0, 2.0, -1.0/
C
      CALL EVLSB (N, A, LDA, NCODA, EVAL)
C                                  Print results
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
                  EVAL
    1       2       3       4       5
4.464  -3.000  -2.464  -2.000   1.000
```

EVCSB/DEVCSB (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a real symmetric matrix in band symmetric storage mode.

Usage

CALL EVCSB (N, A, LDA, NCODA, EVAL, EVEC, LDEVEC)

Arguments

N — Order of the matrix A.   (Input)

A — Band symmetric matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

NCODA — Number of codiagonals in A.   (Input)

EVAL — Vector of length N containing the eigenvalues of A in decreasing order of magnitude.   (Output)

EVEC — Matrix of order N containing the eigenvectors.   (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program.   (Input)

Comments

1. Automatic workspace usage is

   `EVCSB  N`(`NCODA` + 3) units, or
   `DEVCSB` 2`N`(`NCODA` + 2) + `N` units.

   Workspace may be explicitly provided, if desired, by use of `E4CSB`/`DE4CSB`. The reference is

   ```
   CALL E4CSB (N, A, LDA, NCODA, EVAL, EVEC, LDEVEC,
                ACOPY, WK,IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Work array of length `N`(`NCODA` + 1). `A` and `ACOPY` may be the same, in which case the first `N * NCODA` elements of `A` will be destroyed.

   WK — Work array of length `N`.

   IWK — Integer work array of length `N`.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

3. The success of this routine can be checked using `EPISB` (page 366).

Algorithm

Routine `EVCSB` computes the eigenvalues and eigenvectors of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. These transformations are accumulated. The implicit QL algorithm is used to compute the eigenvalues and eigenvectors of the resulting tridiagonal matrix.

The reduction routine is based on the EISPACK routine `BANDR`; see Garbow et al. (1977). The QL routine is based on the EISPACK routine `IMTQL2`; see Smith et al. (1976).

Example

In this example, a `DATA` statement is used to set A to a band matrix given by Gregory and Karney (1969, page 75). The eigenvalues, λ_k, of this matrix are given by

$$\lambda_k = 16\sin^4\left(\frac{k\pi}{2N+2}\right)$$

The eigenvalues and eigenvectors of this real band symmetric matrix are computed and printed. The performance index is also computed and printed. This serves as a check on the computations; for more details, see IMSL routine `EPISB`, page 366.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N, NCODA
      PARAMETER  (N=6, NCODA=2, LDA=NCODA+1, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N), EPISB, EVAL(N), EVEC(LDEVEC,N), PI
      EXTERNAL   EPISB, EVCSB, UMACH, WRRRN
C                                 Define values of A:
C                                 A = (  5  -4   1             )
C                                     ( -4   6  -4   1         )
C                                     (  1  -4   6  -4   1     )
C                                     (      1  -4   6  -4   1 )
C                                     (          1  -4   6  -4 )
C                                     (              1  -4   5 )
C                                 Represented in band symmetric
C                                 form this is:
C                                 A = (  0   0   1   1   1   1 )
C                                     (  0  -4  -4  -4  -4  -4 )
C                                     (  5   6   6   6   6   5 )
C
      DATA A/0.0, 0.0, 5.0, 0.0, -4.0, 6.0, 1.0, -4.0, 6.0, 1.0, -4.0,
     &     6.0, 1.0, -4.0, 6.0, 1.0, -4.0, 5.0/
C
C                                 Find eigenvalues and vectors
      CALL EVCSB (N, A, LDA, NCODA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPISB(N,N,A,LDA,NCODA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN (’EVAL’, 1, N, EVAL, 1, 0)
      CALL WRRRN (’EVEC’, N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,’(/,A,F6.3)’) ’ Performance index = ’, PI
      END
```

Output

```
                       EVAL
    1       2       3       4       5       6
14.45   10.54    5.98    2.42    0.57    0.04

                          EVEC
          1         2         3         4         5         6
1  -0.2319  -0.4179  -0.5211   0.5211  -0.4179   0.2319
2   0.4179   0.5211   0.2319   0.2319  -0.5211   0.4179
3  -0.5211  -0.2319   0.4179  -0.4179  -0.2319   0.5211
4   0.5211  -0.2319  -0.4179  -0.4179   0.2319   0.5211
5  -0.4179   0.5211  -0.2319   0.2319   0.5211   0.4179
6   0.2319  -0.4179   0.5211   0.5211   0.4179   0.2319

Performance index =  0.029
```

EVASB/DEVASB (Single/Double precision)

Compute the largest or smallest eigenvalues of a real symmetric matrix in band symmetric storage mode.

Usage

```
CALL EVASB (N, NEVAL, A, LDA, NCODA, SMALL, EVAL)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalues to be computed. (Input)

A — Band symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

NCODA — Number of codiagonals in `A`. (Input)

SMALL — Logical variable. (Input)
If `.TRUE.`, the smallest `NEVAL` eigenvalues are computed. If `.FALSE.`, the largest `NEVAL` eigenvalues are computed.

EVAL — Vector of length `NEVAL` containing the computed eigenvalues in decreasing order of magnitude. (Output)

Comments

1. Automatic workspace usage is

   `EVASB` `N`(`NCODA` + 4) units, or
   `DEVASB` 2`N`(`NCODA` + 4) units.

   Workspace may be explicitly provided, if desired, by use of `E3ASB`/`DE3ASB`. The reference is

   ```
   CALL E3ASB (N, NEVAL, A, LDA, NCODA, SMALL, EVAL,
               ACOPY, WK)
   ```

   The additional arguments are as follows:

   ACOPY — Work array of length `N`(`NCODA` + 1). `A` and `ACOPY` may be the same, in which case the first `N`(`NCODA` + 1) elements of `A` will be destroyed.

   WK — Work array of length 3`N`.

2. Informational error

Type	Code	
3	1	The iteration for an eigenvalue failed to converge. The best estimate will be returned.

Algorithm

Routine EVASB computes the largest or smallest eigenvalues of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. The rational QR algorithm with Newton corrections is used to compute the extreme eigenvalues of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine BANDR; see Garbow et al. (1978). The QR routine is based on the EISPACK routine RATQR; see Smith et al. (1976).

Example

The following example is given in Gregory and Karney (1969, page 63). The smallest four eigenvalues of the matrix

$$A = \begin{bmatrix} 5 & 2 & 1 & 1 & & & & & & & \\ 2 & 6 & 3 & 1 & 1 & & & & & & \\ 1 & 3 & 6 & 3 & 1 & 1 & & & & & \\ 1 & 1 & 3 & 6 & 3 & 1 & 1 & & & & \\ & 1 & 1 & 3 & 6 & 3 & 1 & 1 & & & \\ & & 1 & 1 & 3 & 6 & 3 & 1 & 1 & & \\ & & & 1 & 1 & 3 & 6 & 3 & 1 & 1 & \\ & & & & 1 & 1 & 3 & 6 & 3 & 1 & 1 \\ & & & & & 1 & 1 & 3 & 6 & 3 & 1 \\ & & & & & & 1 & 1 & 3 & 6 & 2 \\ & & & & & & & 1 & 1 & 2 & 5 \end{bmatrix}$$

are computed and printed.

```
C                                 Declare variables
      INTEGER    LDA, N, NCODA, NEVAL
      PARAMETER  (N=11, NCODA=3, NEVAL=4, LDA=NCODA+1)
C
      REAL       A(LDA,N), EVAL(NEVAL)
      LOGICAL    SMALL
      EXTERNAL   EVASB, SSET, WRRRN
C                                 Set up matrix in band symmetric
C                                 storage mode
      CALL SSET (N, 6.0, A(4,1), LDA)
      CALL SSET (N-1, 3.0, A(3,2), LDA)
      CALL SSET (N-2, 1.0, A(2,3), LDA)
      CALL SSET (N-3, 1.0, A(1,4), LDA)
      CALL SSET (NCODA, 0.0, A(1,1), 1)
      CALL SSET (NCODA-1, 0.0, A(1,2), 1)
      CALL SSET (NCODA-2, 0.0, A(1,3), 1)
      A(4,1) = 5.0
      A(4,N) = 5.0
      A(3,2) = 2.0
      A(3,N) = 2.0
C                                  Find the 4 smallest eigenvalues
      SMALL = .TRUE.
      CALL EVASB (N, NEVAL, A, LDA, NCODA, SMALL, EVAL)
C                                  Print results
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
            EVAL
    1       2       3       4
4.000   3.172   1.804   0.522
```

EVESB/DEVESB (Single/Double precision)

Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix in band symmetric storage mode.

Usage

```
CALL EVESB (N, NEVEC, A, LDA, NCODA, SMALL, EVAL, EVEC,
            LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVEC — Number of eigenvectors to be calculated. (Input)

A — Band symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

NCODA — Number of codiagonals in `A`. (Input)

SMALL — Logical variable. (Input)
If `.TRUE.` , the smallest `NEVEC` eigenvectors are computed. If `.FALSE.` , the largest `NEVEC` eigenvectors are computed.

EVAL — Vector of length `NEVEC` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

EVEC — Real matrix of dimension `N` by `NEVEC`. (Output)
The `J`-th eigenvector, corresponding to `EVAL(J)`, is stored in the `J`-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVESB N(3NCODA + 7) units, or
   DEVESB 2N(3NCODA + 6) + N units.

   Workspace may be explicitly provided, if desired, by use of E4ESB/DE4ESB. The reference is

   ```
   CALL E4ESB (N,NEVEC, A, LDA, NCODA,SMALL,EVAL, EVEC,
               LDEVEC, ACOPY, WK, IWK)
   ```

   The additional argument is

   ACOPY — Work array of length N(NCODA + 1).

   WK — Work array of length N(2NCODA + 5).

   IWK — Integer work array of length N.

2. Informational errors

Type	Code	
3	1	Inverse iteration did not converge. Eigenvector is not correct for the specified eigenvalue.
3	2	The eigenvectors have lost orthogonality.

3. The success of this routine can be checked using EPISB.

Algorithm

Routine EVESB computes the largest or smallest eigenvalues and the corresponding eigenvectors of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. The rational QR algorithm with Newton corrections is used to compute the extreme eigenvalues of this tridiagonal matrix. Inverse iteration and orthogonalization are used to compute the eigenvectors of the given band matrix. The reduction routine is based on the EISPACK routine BANDR; see Garbow et al. (1977). The QR routine is based on the EISPACK routine RATQR; see Smith et al. (1976). The inverse iteration and orthogonalization steps are based on EISPACK routine BANDV using the additional steps given in Hanson et al. (1990).

Example

The following example is given in Gregory and Karney (1969, page 75). The largest three eigenvalues and the corresponding eigenvectors of the matrix are computed and printed.

```
C                                Declare variables
      INTEGER    LDA, LDEVEC, N, NCODA, NEVEC
      PARAMETER  (N=6, NCODA=2, NEVEC=3, LDA=NCODA+1, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N), EPISB, EVAL(NEVEC), EVEC(LDEVEC,NEVEC), PI
```

```
      LOGICAL    SMALL
      EXTERNAL   EPISB, EVESB, UMACH, WRRRN
C                                  Define values of A:
C                                  A = (  5  -4   1             )
C                                      ( -4   6  -4   1         )
C                                      (  1  -4   6  -4   1     )
C                                      (      1  -4   6  -4   1 )
C                                      (          1  -4   6  -4 )
C                                      (              1  -4   5 )
C                                  Represented in band symmetric
C                                  form this is:
C                                  A = (  0   0   1   1   1   1 )
C                                      (  0  -4  -4  -4  -4  -4 )
C                                      (  5   6   6   6   6   5 )
C
      DATA A/0.0, 0.0, 5.0, 0.0, -4.0, 6.0, 1.0, -4.0, 6.0, 1.0, -4.0,
     &     6.0, 1.0, -4.0, 6.0, 1.0, -4.0, 5.0/
C
C                                  Find the 3 largest eigenvalues
C                                  and their eigenvectors.
      SMALL = .FALSE.
      CALL EVESB (N, NEVEC, A, LDA, NCODA, SMALL, EVAL, EVEC, LDEVEC)
C                                  Compute performance index
      PI = EPISB(N,NEVEC,A,LDA,NCODA,EVAL,EVEC,LDEVEC)
C                                  Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN ('EVAL', 1, NEVEC, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, NEVEC, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
        EVAL
    1       2       3
14.45   10.54    5.98

             EVEC
          1         2         3
1   0.2319  -0.4179   0.5211
2  -0.4179   0.5211  -0.2319
3   0.5211  -0.2319  -0.4179
4  -0.5211  -0.2319   0.4179
5   0.4179   0.5211   0.2319
6  -0.2319  -0.4179  -0.5211

Performance index =  0.172
```

EVBSB/DEVBSB (Single/Double precision)

Compute the eigenvalues in a given interval of a real symmetric matrix stored in band symmetric storage mode.

Usage

```
CALL EVBSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH, NEVAL,
            EVAL)
```

Arguments

N — Order of the matrix `A`. (Input)

MXEVAL — Maximum number of eigenvalues to be computed. (Input)

A — Band symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

NCODA — Number of codiagonals in `A`. (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought. (Input)

EHIGH — Upper limit of the interval in which the eigenvalues are sought. (Input)

NEVAL — Number of eigenvalues found. (Output)

EVAL — Real vector of length `MXEVAL` containing the eigenvalues of `A` in the interval (`ELOW`, `EHIGH`) in decreasing order of magnitude. (Output)
Only the first `NEVAL` elements of `EVAL` are set.

Comments

1. Automatic workspace usage is

   `EVBSB` `N`(`NCODA` + 6)units, or

   `DEVBSB` 2`N`(`NCODA` + 6) units.

   Workspace may be explicitly provided, if desired, by use of `E3BSB`/`DE3BSB`. The reference is

   ```
   CALL E3BSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH,
               NEVAL,EVAL, ACOPY, WK)
   ```

   The additional arguments are as follows:

   ACOPY — Work matrix of size `NCODA` + 1 by `N`. `A` and `ACOPY` may be the same, in which case the first `N`(`NCODA` + 1) elements of `A` will be destroyed.

   WK — Work array of length 5`N`.

2. Informational error

Type	Code	
3	1	The number of eigenvalues in the specified interval exceeds MXEVAL. NEVAL contains the number of eigenvalues in the interval. No eigenvalues will be returned.

Algorithm

Routine EVBSB computes the eigenvalues in a given range of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. A bisection algorithm is used to compute the eigenvalues of the tridiagonal matrix in a given range.

The reduction routine is based on the EISPACK routine BANDR; see Garbow et al. (1977). The bisection routine is based on the EISPACK routine BISECT; see Smith et al. (1976).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 77). The eigenvalues in the range (-2.5, 1.5) are computed and printed. As a test, this example uses MXEVAL = 5. The routine EVBSB computes NEVAL, the number of eigenvalues in the given range, has the value 3.

```
C                                 Declare variables
      INTEGER    LDA, MXEVAL, N, NCODA
      PARAMETER  (MXEVAL=5, N=5, NCODA=2, LDA=NCODA+1)
C
      INTEGER    NEVAL, NOUT
      REAL       A(LDA,N), EHIGH, ELOW, EVAL(MXEVAL)
      EXTERNAL   EVBSB, WRRRN
C
C                                 Define values of A:
C                                 A = ( -1   2   1          )
C                                     (  2   0   2   1      )
C                                     (  1   2   0   2   1  )
C                                     (      1   2   0   2  )
C                                     (          1   2  -1  )
C                                 Representedin band symmetric
C                                 form this is:
C                                 A = (  0   0   1   1   1 )
C                                     (  0   2   2   2   2 )
C                                     ( -1   0   0   0  -1 )
      DATA A/0.0, 0.0, -1.0, 0.0, 2.0, 0.0, 1.0, 2.0, 0.0, 1.0, 2.0,
     &     0.0, 1.0, 2.0, -1.0/
C
      ELOW  = -2.5
      EHIGH = 1.5
      CALL EVBSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH, NEVAL, EVAL)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I1)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
NEVAL = 3

          EVAL
     1       2       3
-2.464  -2.000   1.000
```

EVFSB/DEVFSB (Single/Double precision)

Compute the eigenvalues in a given interval and the corresponding eigenvectors of a real symmetric matrix stored in band symmetric storage mode.

Usage

```
CALL EVFSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH, NEVAL,
            EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix A.   (Input)

MXEVAL — Maximum number of eigenvalues to be computed.   (Input)

A — Band symmetric matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

NCODA — Number of codiagonals in A.   (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought. (Input)

EHIGH — Upper limit of the interval in which the eigenvalues are sought. (Input)

NEVAL — Number of eigenvalues found.   (Output)

EVAL — Real vector of length MXEVAL containing the eigenvalues of A in the interval (ELOW, EHIGH) in decreasing order of magnitude.   (Output)
Only the first NEVAL elements of EVAL are significant.

EVEC — Real matrix containing in its first NEVAL columns the eigenvectors associated with the eigenvalues found and stored in EVAL. Eigenvector J corresponds to eigenvalue J for J = 1 to NEVAL. Each vector is normalized to have Euclidean length equal to the value one.   (Output)

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program.   (Input)

Comments

1. Automatic workspace usage is

   EVFSB 3N * NCODA + 9N units, or
   DEVFSB 6N * NCODA + 17N units.

   Workspace may be explicitly provided, if desired, by use of E3FSB/DE3FSB. The reference is

   ```
   CALL E3FSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH,
               NEVAL, EVAL, EVEC, LDEVEC, ACOPY, WK1,
               WK2, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Work matrix of size NCODA + 1 by N.

   WK1 — Work array of length 6N.

   WK2 — Work array of length 2N * NCODA + N

   IWK — Integer work array of length N.

2. Informational errors

Type	Code	
3	1	The number of eigenvalues in the specified interval exceeds MXEVAL. NEVAL contains the number of eigenvalues in the interval. No eigenvalues will be returned.
3	2	Inverse iteration did not converge. Eigenvector is not correct for the specified eigenvalue.
3	3	The eigenvectors have lost orthogonality.

Algorithm

Routine EVFSB computes the eigenvalues in a given range and the corresponding eigenvectors of a real band symmetric matrix. Orthogonal similarity transformations are used to reduce the matrix to an equivalent tridiagonal matrix. A bisection algorithm is used to compute the eigenvalues of the tridiagonal matrix in the required range. Inverse iteration and orthogonalization are used to compute the eigenvectors of the given band symmetric matrix.

The reduction routine is based on the EISPACK routine BANDR; see Garbow et al. (1977). The bisection routine is based on the EISPACK routine BISECT; see Smith et al. (1976). The inverse iteration and orthogonalization steps are based on the EISPACK routine BANDV using remarks from Hanson et al. (1990).

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 75). The eigenvalues in the range [1, 6] and their corresponding eigenvectors are computed and printed. As a test, this example uses MXEVAL = 4. The routine EVFSB computes NEVAL, the number of eigenvalues in the given range has the value 2. As a check on the computations, the performance index is also computed and printed. For more details, see IMSL routine EPISB on page 366.

```
C                                  Declare variables
      INTEGER    LDA, LDEVEC, MXEVAL, N, NCODA
      PARAMETER  (MXEVAL=4, N=6, NCODA=2, LDA=NCODA+1, LDEVEC=N)
C
      INTEGER    NEVAL, NOUT
      REAL       A(LDA,N), EHIGH, ELOW, EPISB, EVAL(MXEVAL),
     &           EVEC(LDEVEC,MXEVAL), PI
      EXTERNAL   EPISB, EVFSB, UMACH, WRRRN
C                                  Define values of A:
C                                  A = (  5  -4   1             )
C                                      ( -4   6  -4   1         )
C                                      (  1  -4   6  -4   1     )
C                                      (      1  -4   6  -4   1 )
C                                      (          1  -4   6  -4 )
C                                      (              1  -4   5 )
C                                  Represented in band symmetric
C                                  form this is:
C                                  A = (  0   0   1   1   1   1 )
C                                      (  0  -4  -4  -4  -4  -4 )
C                                      (  5   6   6   6   6   5 )
      DATA A/0.0, 0.0, 5.0, 0.0, -4.0, 6.0, 1.0, -4.0, 6.0, 1.0, -4.0,
     &     6.0, 1.0, -4.0, 6.0, 1.0, -4.0, 5.0/
C
C                                  Find eigenvalues and vectors
      ELOW  = 1.0
      EHIGH = 6.0
      CALL EVFSB (N, MXEVAL, A, LDA, NCODA, ELOW, EHIGH, NEVAL, EVAL,
     &            EVEC, LDEVEC)
C                                  Compute performance index
      PI = EPISB(N,NEVAL,A,LDA,NCODA,EVAL,EVEC,LDEVEC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I1)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, NEVAL, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
NEVAL = 2

    EVAL
    1       2
5.978   2.418

       EVEC
         1        2
1   0.5211   0.5211
2  -0.2319   0.2319
3  -0.4179  -0.4179
4   0.4179  -0.4179
5   0.2319   0.2319
6  -0.5211   0.5211

    Performance index =  0.082
```

EPISB/DEPISB (Single/Double precision)

Compute the performance index for a real symmetric eigensystem in band symmetric storage mode.

Usage

```
EPISB(N, NEVAL, A, LDA, NCODA, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix A. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs on which the performance is based. (Input)

A — Band symmetric matrix of order N. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

NCODA — Number of codiagonals in A. (Input)

EVAL — Vector of length NEVAL containing eigenvalues of A. (Input)

EVEC — N by NEVAL array containing eigenvectors of A. (Input)
The eigenvector corresponding to the eigenvalue EVAL(J) must be in the J-th column of EVEC.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

EPISB — Performance index. (Output)

Comments

1. Automatic workspace usage is

   EPISB N units, or
   DEPISB 2N units.

   Workspace may be explicitly provided, if desired, by use of E2ISB/DE2ISB. The reference is

   ```
   E2ISB(N, NEVAL, A, LDA, NCODA, EVAL, EVEC, LDEVEC, WK)
   ```

   The additional argument is

   WK — Work array of length N.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix is zero.

Algorithm

Let M = NEVAL, λ = EVAL, x_j = EVEC(*,J), the j-th column of EVEC. Also, let ε be the machine precision, given by AMACH(4), page 1201. The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| Ax_j - \lambda_j x_j \right\|_1}{10 N \varepsilon \left\| A \right\|_1 \left\| x_j \right\|_1}$$

While the exact value of τ is highly machine dependent, the performance of EVCSF (page 339) is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124−125).

Example

For an example of EPISB, see IMSL routine EVCSB on page 353.

EVLHF/DEVLHF (Single/Double precision)

Compute all of the eigenvalues of a complex Hermitian matrix.

Usage

```
CALL EVLHF (N, A, LDA, EVAL)
```

Arguments

N — Order of the matrix A.   (Input)

A — Complex Hermitian matrix of order N.   (Input)
Only the upper triangle is used.

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Real vector of length N containing the eigenvalues of A in decreasing order of magnitude.   (Output)

Comments

1. Automatic workspace usage is

   EVLHF   2 N^2 + 6N units, or
   DEVLHF 2(2 N^2 + 5N) + N units.

   Workspace may be explicitly provided, if desired, by use of E3LHF/DE3LHF. The reference is

   ```
   CALL E3LHF (N, A, LDA, EVAL, ACOPY, RWK, CWK, IWK)
   ```

The additional arguments are as follows:

ACOPY — Complex work array of length N^2. A and ACOPY may be the same in which case A will be destroyed.

RWK — Work array of length N.

CWK — Complex work array of length 2N.

IWK — Integer work array of length N.

2. Informational errors

Type	Code	
3	1	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	1	The iteration for an eigenvalue failed to converge.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. Integer Options with Chapter 10 Options Manager

1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E3LHF, the internal or working leading dimensions of ACOPY and ECOPY are both increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVLHF. Additional memory allocation and option value restoration are automatically done in EVLHF. There is no requirement that users change existing applications that use EVLHF or E3LHF. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5 – 8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVLHF.

Algorithm

Routine EVLHF computes the eigenvalues of a complex Hermitian matrix. Unitary similarity transformations are used to reduce the matrix to an equivalent real symmetric tridiagonal matrix. The implicit QL algorithm is used to compute the eigenvalues of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine HTRIDI. The QL routine is based on the EISPACK routine IMTQL1. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 114). The eigenvalues of this complex Hermitian matrix are computed and printed.

```
C                                Declare variables
```

```
      INTEGER    LDA, N
      PARAMETER  (N=2, LDA=N)
C
      REAL       EVAL(N)
      COMPLEX    A(LDA,N)
      EXTERNAL   EVLHF, WRRRN
C                                 Set values of A
C
C                                 A = (  1      -i  )
C                                     (  i       1  )
C
      DATA A/(1.0,0.0), (0.0,1.0), (0.0,-1.0), (1.0,0.0)/
C
C                                 Find eigenvalues of A
      CALL EVLHF (N, A, LDA, EVAL)
C                                 Print results
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
       EVAL
    1       2
2.000   0.000
```

EVCHF/DEVCHF (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a complex Hermitian matrix.

Usage

```
CALL EVCHF (N, A, LDA, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix A. (Input)

A — Complex Hermitian matrix of order N. (Input)
Only the upper triangle is used.

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Real vector of length N containing the eigenvalues of A in decreasing order of magnitude. (Output)

EVEC — Complex matrix of order N. (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVCHF $3N^2 + 6N$ units, or
   DEVCHF $6N^2 + 11N$ units.

   Workspace may be explicitly provided, if desired, by use of E5CHF/DE5CHF. The reference is

   ```
   CALL E5CHF (N, A, LDA, EVAL, EVEC, LDEVEC, ACOPY,
               RWK, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work array of length N^2. A and ACOPY may be the same, in which case A will be destroyed.

   RWK — Work array of length $N^2 + N$.

   CWK — Complex work array of length 2N.

   IWK — Integer work array of length N.

2. Informational error

Type	Code	
3	1	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	1	The iteration for an eigenvalue failed to converge.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. The success of this routine can be checked using EPIHF (page 382).

4. Integer Options with Chapter 10 Options Manager

   1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine E5CHF, the internal or working leading dimensions of ACOPY and ECOPY are both increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine EVCHF. Additional memory allocation and option value restoration are automatically done in EVCHF. There is no requirement that users change existing applications that use EVCHF or E5CHF. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5–8 in IVAL(*) are for the generalized eigenvalue problem and are not used in EVCHF.

Algorithm

Routine EVCHF computes the eigenvalues and eigenvectors of a complex Hermitian matrix. Unitary similarity transformations are used to reduce the matrix to an equivalent real symmetric tridiagonal matrix. The implicit QL algorithm is used to compute the eigenvalues and eigenvectors of this tridiagonal matrix. These eigenvectors and the transformations used to reduce the matrix to tridiagonal form are combined to obtain the eigenvectors for the user's problem. The reduction routine is based on the EISPACK routine `HTRIDI`. The QL routine is based on the EISPACK routine `IMTQL2`. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a `DATA` statement is used to set *A* to a complex Hermitian matrix. The eigenvalues and eigenvectors of this matrix are computed and printed. The performance index is also computed and printed. This serves as a check on the computations; for more details, see routine `EPIHF` on page 382.

```
C                                  Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       EPIHF, EVAL(N), PI
      COMPLEX    A(LDA,N), EVEC(LDEVEC,N)
      EXTERNAL   EPIHF, EVCHF, UMACH, WRCRN, WRRRN
C                                  Set values of A
C
C                                  A = ((1, 0)  (  1,-7i)  ( 0,- i))
C                                      ((1,7i)  (  5,  0)  (10,-3i))
C                                      ((0, i)  ( 10, 3i)  (-2,  0))
C
      DATA A/(1.0,0.0), (1.0,7.0), (0.0,1.0), (1.0,-7.0), (5.0,0.0),
     &     (10.0, 3.0), (0.0,-1.0), (10.0,-3.0), (-2.0,0.0)/
C
C                                  Find eigenvalues and vectors of A
      CALL EVCHF (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                  Compute performance index
      PI = EPIHF(N,N,A,LDA,EVAL,EVEC,LDEVEC)
C                                  Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
         EVAL
    1       2       3
15.38  -10.63   -0.75

                            EVEC
                   1                   2                   3
1  ( 0.0631,-0.4075)  (-0.0598,-0.3117)  ( 0.8539, 0.0000)
2  ( 0.7703, 0.0000)  (-0.5939, 0.1841)  (-0.0313,-0.1380)
3  ( 0.4668, 0.1366)  ( 0.7160, 0.0000)  ( 0.0808,-0.4942)

Performance index =  0.093
```

EVAHF/DEVAHF (Single/Double precision)

Compute the largest or smallest eigenvalues of a complex Hermitian matrix.

Usage

```
CALL EVAHF (N, NEVAL, A, LDA, SMALL, EVAL)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalues to be calculated. (Input)

A — Complex Hermitian matrix of order `N`. (Input)
Only the upper triangle is used.

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

SMALL — Logical variable. (Input)
If `.TRUE.`, the smallest `NEVAL` eigenvalues are computed. If `.FALSE.`, the largest `NEVAL` eigenvalues are computed.

EVAL — Real vector of length `NEVAL` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

Comments

1. Automatic workspace usage is

   `EVAHF` 2N^2 + 7`N` units, or
   `DEVAHF` 4N^2 + 13`N` units.

   Workspace may be explicitly provided, if desired, by use of `E3AHF`/`DE3AHF`. The reference is

   ```
   CALL E3AHF (N, NEVAL, A, LDA, SMALL, EVAL, ACOPY,
               RWK, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work array of length N^2. `A` and `ACOPY` may be the same in which case `A` will be destroyed.

   RWK — Work array of length 2`N`.

   CWK — Complex work array of length 2`N`.

   IWK — Work array of length `N`.

2. Informational errors

Type	Code	
3	1	The iteration for an eigenvalue failed to converge. The best estimate will be returned.

3	2	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine EVAHF computes the largest or smallest eigenvalues of a complex Hermitian matrix. Unitary transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. The rational QR algorithm with Newton corrections is used to compute the extreme eigenvalues of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine HTRIDI. The QR routine is based on the EISPACK routine RATQR. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a DATA statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 114). Its largest eigenvalue is computed and printed.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=2, LDA=N)
C
      INTEGER    NEVAL
      REAL       EVAL(N)
      COMPLEX    A(LDA,N)
      LOGICAL    SMALL
      EXTERNAL   EVAHF, WRRRN
C                                 Set values of A
C
C                                 A = (  1      -i  )
C                                     (  i       1  )
C
      DATA A/(1.0,0.0), (0.0,1.0), (0.0,-1.0), (1.0,0.0)/
C
C                                 Find the largest eigenvalue of A
      NEVAL = 1
      SMALL = .FALSE.
      CALL EVAHF (N, NEVAL, A, LDA, SMALL, EVAL)
C                                 Print results
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
EVAL
2.000
```

EVEHF/DEHF (Single/Double precision)

Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix.

Usage

```
CALL EVEHF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

NEVEC — Number of eigenvectors to be computed. (Input)

A — Complex Hermitian matrix of order `N`. (Input)
Only the upper triangle is used.

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

SMALL — Logical variable. (Input)
If `.TRUE.`, the smallest `NEVEC` eigenvectors are computed. If `.FALSE.`, the largest `NEVEC` eigenvectors are computed.

EVAL — Real vector of length `NEVEC` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

EVEC — Complex matrix of dimension `N` by `NEVEC`. (Output)
The `J`-th eigenvector corresponding to `EVAL(J)`, is stored in the `J`-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVEHF $2N^2$ + N * NEVEC + 13N units, or
   DEVEHF $4N^2$ + 2N * NEVEC + 25N units.

   Workspace may be explicitly provided, if desired, by use of `E3EHF/DE3EHF`. The reference is

   ```
   CALL E3EHF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC,
               LDEVEC, ACOPY, RW1, RW2, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work array of length N^2. `A` and `ACOPY` may be the same, in which case `A` will be destroyed.

RW1 — Work array of length `N * NEVEC`. Used to store the real eigenvectors of a symmetric tridiagonal matrix.

RW2 — Work array of length 8`N`.

CWK — Complex work array of length 2`N`.

IWK — Work array of length `N`.

2. Informational errors

Type	Code	
3	1	The iteration for an eigenvalue failed to converge. The best estimate will be returned.
3	2	The iteration for an eigenvector failed to converge. The eigenvector will be set to 0.
3	3	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

3. The success of this routine can be checked using `EPIHF` (page 382).

Algorithm

Routine `EVEHF` computes the largest or smallest eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix. Unitary transformations are used to reduce the matrix to an equivalent real symmetric tridiagonal matrix. The rational QR algorithm with Newton corrections is used to compute the extreme eigenvalues of the tridiagonal matrix. Inverse iteration is used to compute the eigenvectors of the tridiagonal matrix. Eigenvectors of the original matrix are found by back transforming the eigenvectors of the tridiagonal matrix.

The reduction routine is based on the EISPACK routine `HTRIDI`. The QR routine used is based on the EISPACK routine `RATQR`. The inverse iteration routine is based on the EISPACK routine `TINVIT`. The back transformation routine is based on the EISPACK routine `HTRIBK`. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a `DATA` statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 115). The smallest eigenvalue and its corresponding eigenvector is computed and printed. The performance index is also computed and printed. This serves as a check on the computations. For more details, see IMSL routine `EPIHF` on page 382.

```
C                                  Declare variables
      INTEGER    LDA, LDEVEC, N, NEVEC
      PARAMETER  (N=3, NEVEC=1, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       EPIHF, EVAL(N), PI
      COMPLEX    A(LDA,N), EVEC(LDEVEC,NEVEC)
      LOGICAL    SMALL
      EXTERNAL   EPIHF, EVEHF, UMACH, WRCRN, WRRRN
C                                 Set values of A
C
C                                 A = (  2      -i      0  )
C                                     (  i       2      0  )
C                                     (  0       0      3  )
C
      DATA A/(2.0,0.0), (0.0,1.0), (0.0,0.0), (0.0,-1.0), (2.0,0.0),
     &     (0.0,0.0), (0.0,0.0), (0.0,0.0), (3.0,0.0)/
C
C                                 Find smallest eigenvalue and its
C                                 eigenvectors
      SMALL = .TRUE.
      CALL EVEHF (N, NEVEC, A, LDA, SMALL, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPIHF(N,NEVEC,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN (’EVAL’, 1, NEVEC, EVAL, 1, 0)
      CALL WRCRN (’EVEC’, N, NEVEC, EVEC, LDEVEC, 0)
      WRITE (NOUT,’(/,A,F6.3)’) ’ Performance index = ’, PI
      END
```

Output

```
EVAL
1.000

         EVEC
1  ( 0.0000, 0.7071)
2  ( 0.7071, 0.0000)
3  ( 0.0000, 0.0000)

Performance index =  0.031
```

EVBHF/DEVBHF (Single/Double precision)

Compute the eigenvalues in a given range of a complex Hermitian matrix.

Usage

`CALL EVBHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL)`

Arguments

N — Order of the matrix `A`. (Input)

MXEVAL — Maximum number of eigenvalues to be computed. (Input)

A — Complex Hermitian matrix of order `N`. (Input)
Only the upper triangle is used.

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought. (Input)

EHIGH – Upper limit of the interval in which the eigenvalues are sought. (Input)

NEVAL — Number of eigenvalues found. (Output)

EVAL — Real vector of length MXEVAL containing the eigenvalues of A in the interval (ELOW, EHIGH) in decreasing order of magnitude. (Output)
Only the first NEVAL elements of EVAL are significant.

Comments

1. Automatic workspace usage is

   EVBHF $2N^2 + 9N$ + MXEVAL units, or
   DEVBHF $4N^2 + 18N$ + MXEVAL units.

   Workspace may be explicitly provided, if desired, by use of E3BHF/DE3BHF. The reference is

   ```
   CALL E3BHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL,
               EVAL, ACOPY, RWK, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work matrix of size N by N. A and ACOPY may be the same, in which case the first N^2 elements of A will be destroyed.

   RWK — Work array of length 5N.

   CWK — Complex work array of length 2N.

   IWK — Work array of length MXEVAL.

2. Informational errors

Type	Code	
3	1	The number of eigenvalues in the specified range exceeds MXEVAL. NEVAL contains the number of eigenvalues in the range. No eigenvalues will be computed.
3	2	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine EVBHF computes the eigenvalues in a given range of a complex Hermitian matrix. Unitary transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. A bisection algorithm is used to compute the eigenvalues in the given range of this tridiagonal matrix.

The reduction routine is based on the EISPACK routine `HTRIDI`. The bisection routine used is based on the EISPACK routine `BISECT`. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a `DATA` statement is used to set *A* to a matrix given by Gregory and Karney (1969, page 114). The eigenvalues in the range [1.5, 2.5] are computed and printed. This example allows a maximum number of eigenvalues `MXEVAL` = 2. The routine computes that there is one eigenvalue in the given range. This value is returned in `NEVAL`.

```
C                                  Declare variables
      INTEGER    LDA, MXEVAL, N
      PARAMETER  (MXEVAL=2, N=2, LDA=N)
C
      INTEGER    NEVAL, NOUT
      REAL       EHIGH, ELOW, EVAL(MXEVAL)
      COMPLEX    A(LDA,N)
      EXTERNAL   EVBHF, UMACH, WRRRN
C                                  Set values of A
C
C                                  A = (  1      -i  )
C                                      (  i       1  )
C
      DATA A/(1.0,0.0), (0.0,1.0), (0.0,-1.0), (1.0,0.0)/
C
C                                  Find eigenvalue
      ELOW  = 1.5
      EHIGH = 2.5
      CALL EVBHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL)
C
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I3)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      END
```

Output

```
NEVAL =   1

EVAL
2.000
```

EVFHF/DEVFHF (Single/Double precision)

Compute the eigenvalues in a given range and the corresponding eigenvectors of a complex Hermitian matrix.

Usage

```
CALL EVFHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL,
           EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`.   (Input)

MXEVAL — Maximum number of eigenvalues to be computed.   (Input)

A — Complex Hermitian matrix of order `N`.   (Input)
Only the upper triangle is used.

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

ELOW — Lower limit of the interval in which the eigenvalues are sought. (Input)

EHIGH — Upper limit of the interval in which the eigenvalues are sought. (Input)

NEVAL — Number of eigenvalues found.   (Output)

EVAL — Real vector of length `MXEVAL` containing the eigenvalues of `A` in the interval (`ELOW`, `EHIGH`) in decreasing order of magnitude.   (Output)
Only the first `NEVAL` elements of `EVAL` are significant.

EVEC — Complex matrix containing in its first `NEVAL` columns the eigenvectors associated with the eigenvalues found stored in `EVAL`. Each vector is normalized to have Euclidean length equal to the value one.   (Output)

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program.   (Input)

Comments

1. Automatic workspace usage is

   `EVFHF` $2N^2$ + `N` * `MXEVAL` + 12`N` + `MXEVAL` units, or

   `DEVFHF` $4N^2$ + 2`N` * `MXEVAL` + 24`N` + `MXEVAL` units.

   Workspace may be explicitly provided, if desired, by use of `E3FHF`/`DE3FHF`. The reference is

   ```
   CALL E3FHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL,
               EVAL, EVEC,LDEVEC, ACOPY, ECOPY, RWK,
               CWK, IWK)
   ```

The additional arguments are as follows:

ACOPY — Complex work matrix of size `N` by `N`. `A` and `ACOPY` may be the same, in which case the first N^2 elements of `A` will be destroyed.

ECOPY — Work matrix of size `N` by `MXEVAL`. Used to store eigenvectors of a real tridiagonal matrix.

RWK — Work array of length 8`N`.

CWK — Complex work array of length 2`N`.

IWK — Work array of length `MXEVAL`.

2. Informational errors

Type	Code	
3	1	The number of eigenvalues in the specified range exceeds `MXEVAL`. `NEVAL` contains the number of eigenvalues in the range. No eigenvalues will be computed.
3	2	The iteration for an eigenvector failed to converge. The eigenvector will be set to 0.
3	3	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

Routine `EVFHF` computes the eigenvalues in a given range and the corresponding eigenvectors of a complex Hermitian matrix. Unitary transformations are used to reduce the matrix to an equivalent symmetric tridiagonal matrix. A bisection algorithm is used to compute the eigenvalues in the given range of this tridiagonal matrix. Inverse iteration is used to compute the eigenvectors of the tridiagonal matrix. The eigenvectors of the original matrix are computed by back transforming the eigenvectors of the tridiagonal matrix.

The reduction routine is based on the EISPACK routine `HTRIDI`. The bisection routine is based on the EISPACK routine `BISECT`. The inverse iteration routine is based on the EISPACK routine `TINVIT`. The back transformation routine is based on the EISPACK routine `HTRIBK`. See Smith et al. (1976) for the EISPACK routines.

Example

In this example, a `DATA` statement is used to set A to a complex Hermitian matrix. The eigenvalues in the range [–15, 0] and their corresponding eigenvectors are computed and printed. As a test, this example uses `MXEVAL` = 3. The routine `EVFHF` computes the number of eigenvalues in the given range. That value, `NEVAL`, is two. As a check on the computations, the performance index is also computed and printed. For more details, see routine EPIHF on page 382.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, MXEVAL, N
      PARAMETER  (MXEVAL=3, N=3, LDA=N, LDEVEC=N)
C
      INTEGER    NEVAL, NOUT
      REAL       EHIGH, ELOW, EPIHF, EVAL(MXEVAL), PI
      COMPLEX    A(LDA,N), EVEC(LDEVEC,MXEVAL)
      EXTERNAL   EPIHF, EVFHF, UMACH, WRCRN, WRRRN
C                                 Set values of A
C
C                                 A = ((1, 0)  (  1,-7i)  ( 0,- i))
C                                     ((1,7i)  (  5,  0)  (10,-3i))
C                                     ((0, i)  ( 10, 3i)  (-2,  0))
C
      DATA A/(1.0,0.0), (1.0,7.0), (0.0,1.0), (1.0,-7.0), (5.0,0.0),
     &     (10.0,3.0), (0.0,-1.0), (10.0,-3.0), (-2.0,0.0)/
C
C                                 Find eigenvalues and vectors
      ELOW  = -15.0
      EHIGH = 0.0
      CALL EVFHF (N, MXEVAL, A, LDA, ELOW, EHIGH, NEVAL, EVAL, EVEC,
     &            LDEVEC)
C                                 Compute performance index
      PI = EPIHF(N,NEVAL,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(/,A,I3)') ' NEVAL = ', NEVAL
      CALL WRRRN ('EVAL', 1, NEVAL, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, NEVAL, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
NEVAL =   2

      EVAL
     1       2
-10.63   -0.75

                 EVEC
                   1                  2
1  (-0.0598,-0.3117)  ( 0.8539, 0.0000)
2  (-0.5939, 0.1841)  (-0.0313,-0.1380)
3  ( 0.7160, 0.0000)  ( 0.0808,-0.4942)

    Performance index =  0.057
```

EPIHF/DEPIHF (Single/Double precision)

Compute the performance index for a complex Hermitian eigensystem.

Usage

`EPIHF(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrix `A`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs on which the performance index computation is based. (Input)

A — Complex Hermitian matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Vector of length `NEVAL` containing eigenvalues of `A`. (Input)

EVEC — Complex `N` by `NEVAL` array containing eigenvectors of `A`. (Input)
The eigenvector corresponding to the eigenvalue `EVAL(J)` must be in the `J`-th column of `EVEC`.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

EPIHF — Performance index. (Output)

Comments

1 Automatic workspace usage is

`EPIHF` 2`N` units, or
`DEPIHF` 4`N` units.

Workspace may be explicitly provided, if desired, by use of `E2IHF/DE2IHF`. The reference is

`E2IHF(N, NEVAL, A, LDA, EVAL, EVEC, LDEVEC, WK)`

The additional argument is

WK — Complex work array of length `N`.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix is zero.

Algorithm

Let M = NEVAL, λ = EVAL, x_j = EVEC(*, J), the j-th column of EVEC. Also, let ε be the machine precision, given by AMACH(4), page 1201. The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\|Ax_j - \lambda_j x_j\right\|_1}{10 N \varepsilon \left\|A\right\|_1 \left\|x_j\right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector v is

$$\left\|v\right\|_1 = \sum_{i=1}^{N} \left\{ \left|\Re v_i\right| + \left|\Im v_i\right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of EVCSF (page 339) is considered excellent if τ < 1, good if 1 ≤ τ ≤ 100, and poor if τ > 100. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Smith et al. (1976, pages 124−125).

Example

For an example of EPIHF, see IMSL routine EVCHF, page 369.

EVLRH/DEVLRH (Single/Double precision)

Compute all of the eigenvalues of a real upper Hessenberg matrix.

Usage

```
CALL EVLRH (N, A, LDA, EVAL)
```

Arguments

N — Order of the matrix A.   (Input)

A — Real upper Hessenberg matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Complex vector of length N containing the eigenvalues in decreasing order of magnitude.   (Output)

Comments

1. Automatic workspace usage is

   EVLRH N^2 + 4N units, or
   DEVLRH $2N^2$ + 7N units.

Workspace may be explicitly provided, if desired, by use of E3LRH/DE3LRH. The reference is

CALL E3LRH (N, A, LDA, EVAL, ACOPY, WK, IWK)

The additional argument is

ACOPY — Real N by N work matrix.

WK — Real vector of length 3*n*.

IWK — Integer vector of length *n*.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

Algorithm

Routine EVLRH computes the eigenvalues of a real upper Hessenberg matrix by using the QR algorithm. The QR Algorithm routine is based on the EISPACK routine HQR, Smith et al. (1976).

Example

In this example, a DATA statement is used to set A to an upper Hessenberg matrix of integers. The eigenvalues of this matrix are computed and printed.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (N=4, LDA=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N)
      COMPLEX    EVAL(N)
      EXTERNAL   EVLRH, UMACH, WRCRN
C                                 Set values of A
C
C                                 A = (  2.0    1.0    3.0    4.0  )
C                                     (  1.0    0.0    0.0    0.0  )
C                                     (         1.0    0.0    0.0  )
C                                     (                1.0    0.0  )
C
      DATA A/2.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 3.0, 0.0, 0.0,
     &     1.0, 4.0, 0.0, 0.0, 0.0/
C
C                                 Find eigenvalues of A
      CALL EVLRH (N, A, LDA, EVAL)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
                                EVAL
              1                  2                  3                  4
( 2.878, 0.000)  ( 0.011, 1.243)  ( 0.011,-1.243)  (-0.900, 0.000)
```

EVCRH/DEVCRH (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a real upper Hessenberg matrix.

Usage

```
CALL EVCRH (N, A, LDA, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrix `A`. (Input)

A — Real upper Hessenberg matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Complex vector of length `N` containing the eigenvalues in decreasing order of magnitude. (Output)

EVEC — Complex matrix of order `N`. (Output)
The `J`-th eigenvector, corresponding to `EVAL(J)`, is stored in the `J`-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   `EVCRH` $2N^2 + 4N$ units, or
   `DEVCRH` $4N^2 + 7N$ units.

   Workspace may be explicitly provided, if desired, by use of `E6CRH`/`DE6CRH`. The reference is

   ```
   CALL E6CRH (N, A, LDA, EVAL, EVEC, LDEVEC, ACOPY,
               ECOPY,RWK,IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Real `N` by `N` work matrix.

   ECOPY — Real `N` by `N` work matrix.

   RWK — Real array of length 3`N`.

   IWK — Integer array of length `N`.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

Algorithm

Routine EVCRH computes the eigenvalues and eigenvectors of a real upper Hessenberg matrix by using the QR algorithm. The QR algorithm routine is based on the EISPACK routine HQR2; see Smith et al. (1976).

Example

In this example, a DATA statement is used to set *A* to a Hessenberg matrix with integer entries. The values are returned in decreasing order of magnitude. The eigenvalues, eigenvectors and performance index of this matrix are computed and printed. See routine EPIRG on page 330 for details.

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=4, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N), EPIRG, PI
      COMPLEX    EVAL(N), EVEC(LDEVEC,N)
      EXTERNAL   EPIRG, EVCRH, UMACH, WRCRN
C                                 Define values of A:
C
C                                 A = ( -1.0   -1.0   -1.0   -1.0  )
C                                     (  1.0    0.0    0.0    0.0  )
C                                     (         1.0    0.0    0.0  )
C                                     (                1.0    0.0  )
C
      DATA A/-1.0, 1.0, 0.0, 0.0, -1.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0,
     &     1.0, -1.0, 0.0, 0.0, 0.0/
C
C                                 Find eigenvalues and vectors of A
      CALL EVCRH (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPIRG(N,N,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
                                     EVAL
                1                  2                  3                  4
(-0.8090, 0.5878)  (-0.8090,-0.5878)  ( 0.3090, 0.9511)  ( 0.3090,-0.9511)
```

```
                                     EVEC
                 1                  2                  3                  4
1  (-0.4045, 0.2939)  (-0.4045,-0.2939)  ( 0.1545, 0.4755)  ( 0.1545,-0.4755)
2  ( 0.5000, 0.0000)  ( 0.5000, 0.0000)  ( 0.5000, 0.0000)  ( 0.5000, 0.0000)
3  (-0.4045,-0.2939)  (-0.4045, 0.2939)  ( 0.1545,-0.4755)  ( 0.1545, 0.4755)
4  ( 0.1545, 0.4755)  ( 0.1545,-0.4755)  (-0.4045,-0.2939)  (-0.4045, 0.2939)

Performance index =  0.051
```

EVLCH/DEVLCH (Single/Double precision)

Compute all of the eigenvalues of a complex upper Hessenberg matrix.

Usage

`CALL EVLCH (N, A, LDA, EVAL)`

Arguments

N — Order of the matrix `A`. (Input)

A — Complex upper Hessenberg matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Complex vector of length `N` containing the eigenvalues of `A` in decreasing order of magnitude. (Output)

Comments

1. Automatic workspace usage is

   `EVLCH` $2N^2 + 2N$ units, or
   `DEVLCH` $4N^2 + 3N$ units.

   Workspace may be explicitly provided, if desired, by use of `E3LCH`/`DE3LCH`. The reference is

   `CALL E3LCH (N, A, LDA, EVAL, ACOPY, RWK, IWK)`

   The additional arguments are

   ACOPY — Complex `N` by `N` work array. `A` and `ACOPY` may be the same, in which case `A` is destroyed.

   RWK — Real work array of length `N`.

   IWK — Integer work array of length `N`.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

Algorithm

Routine `EVLCH` computes the eigenvalues of a complex upper Hessenberg matrix using the QR algorithm. This routine is based on the EISPACK routine `COMQR2`; see Smith et al. (1976).

Example

In this example, a `DATA` statement is used to set the matrix `A`. The program computes and prints the eigenvalues of this matrix.

```
C                                 Declare variables
      INTEGER LDA, N
      PARAMETER (N=4, LDA=N)
      COMPLEX A(LDA,N), EVAL(N)
C                                 Set values of A
C
C                                 A = (5+9i  5+5i  -6-6i  -7-7i)
C                                     (3+3i  6+10i -5-5i  -6-6i)
C                                     ( 0    3+3i  -1+3i  -5-5i)
C                                     ( 0     0    -3-3i     4i)
C
      DATA A /(5.0,9.0), (3.0,3.0), (0.0,0.0), (0.0,0.0),
     &        (5.0,5.0), (6.0,10.0), (3.0,3.0), (0.0,0.0),
     &        (-6.0,-6.0), (-5.0,-5.0), (-1.0,3.0), (-3.0,-3.0),
     &        (-7.0,-7.0), (-6.0,-6.0), (-5.0,-5.0), (0.0,4.0)/
C
C                                 Find the eigenvalues of A
      CALL EVLCH (N, A, LDA, EVAL)
C                                 Print results
      CALL WRCRN (’EVAL’, 1, N, EVAL, 1, 0)
      END
```

Output

```
                               EVAL
              1                2                3                4
(  8.22, 12.22)  (  3.40,  7.40)  (  1.60,  5.60)  ( -3.22,  0.78)
```

EVCCH/DEVCCH (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a complex upper Hessenberg matrix.

Usage

`CALL EVCCH (N, A, LDA, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrix `A`. (Input)

A — Complex upper Hessenberg matrix of order `N`. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Complex vector of length N containing the eigenvalues of A in decreasing order of magnitude. (Output)

EVEC — Complex matrix of order N. (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

Comments

1. Automatic workspace usage is

   EVCCH 4N(N + 2) units, or
   DEVCCH 7N(N + 2) units.

   Workspace may be explicitly provided, if desired, by use of E4CCH/DE4CCH. The reference is

   ```
   CALL E4CCH (N, A, LDA, EVAL, EVEC, LDEVEC, ACOPY,
               CWORK, RWK,IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex N by N work array. A and ACOPY may be the same, in which case A is destroyed.

   CWORK — Complex work array of length 2N.

   RWK — Real work array of length N.

   IWK — Integer work array of length N.

2 Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

3. The results of EVCCH can be checked using EPICG (page 336). This requires that the matrix A explicitly contains the zeros in A(I, J) for (I − 1) > J which are assumed by EVCCH.

Algorithm

Routine EVCCH computes the eigenvalues and eigenvectors of a complex upper Hessenberg matrix using the QR algorithm. This routine is based on the EISPACK routine COMQR2; see Smith et al. (1976).

Example

In this example, a DATA statement is used to set the matrix *A*. The program computes the eigenvalues and eigenvectors of this matrix. The performance index is also computed and printed. This serves as a check on the computations; for more details, see IMSL routine EPICG, page 336. The zeros in the lower part of the matrix are not referenced by EVCCH, but they are required by EPICG (page 336).

```
C                                 Declare variables
      INTEGER    LDA, LDEVEC, N
      PARAMETER  (N=4, LDA=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       EPICG, PI
      COMPLEX    A(LDA,N), EVAL(N), EVEC(LDEVEC,N)
      EXTERNAL   EPICG, EVCCH, UMACH, WRCRN
C                                 Set values of A
C
C                                 A = (5+9i  5+5i  -6-6i  -7-7i)
C                                     (3+3i  6+10i -5-5i  -6-6i)
C                                     ( 0    3+3i  -1+3i  -5-5i)
C                                     ( 0     0    -3-3i     4i)
C
      DATA A/(5.0,9.0), (3.0,3.0), (0.0,0.0), (0.0,0.0), (5.0,5.0),
     &     (6.0,10.0), (3.0,3.0), (0.0,0.0), (-6.0,-6.0), (-5.0,-5.0),
     &     (-1.0,3.0), (-3.0,-3.0), (-7.0,-7.0), (-6.0,-6.0),
     &     (-5.0,-5.0), (0.0,4.0)/
C
C                                 Find eigenvalues and vectors of A
      CALL EVCCH (N, A, LDA, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = EPICG(N,N,A,LDA,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
                                 EVAL
              1                 2                 3                 4
(  8.22, 12.22)  (  3.40,  7.40)  (  1.60,  5.60)  ( -3.22,  0.78)

                                    EVEC
                  1                  2                   3                   4
1 ( 0.7167, 0.0000) (-0.0704, 0.0000)  (-0.3678, 0.0000)  ( 0.5429, 0.0000)
2 ( 0.6402, 0.0000) (-0.0046, 0.0000)  ( 0.6767, 0.0000)  ( 0.4298, 0.0000)
3 ( 0.2598, 0.0000) ( 0.7477, 0.0000)  (-0.3005, 0.0000)  ( 0.5277, 0.0000)
4 (-0.0948, 0.0000) (-0.6603, 0.0000)  ( 0.5625, 0.0000)  ( 0.4920, 0.0000)

Performance index =  0.020
```

GVLRG/DGVLRG (Single/Double precision)

Compute all of the eigenvalues of a generalized real eigensystem $Az = \lambda Bz$.

Usage

```
CALL GVLRG (N, A, LDA, B, LDB, ALPHA, BETA)
```

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Real matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

B — Real matrix of order `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program. (Input)

ALPHA — Complex vector of size `N` containing scalars α_i, $i = 1, \ldots, n$. If $\beta_i \neq 0$, $\lambda_i = \alpha_i / \beta_i$ the eigenvalues of the system in decreasing order of magnitude. (Output)

BETA — Real vector of size `N`. (Output)

Comments

1. Automatic workspace usage is

   `GVLRG` $2N^2 + 4N$ units, or
   `DGVLRG` $4N^2 + 7N$ units.

   Workspace may be explicitly provided, if desired, by use of `G3LRG`/`DG3LRG`. The reference is

   ```
   CALL G3LRG (N, A, LDA, B, LDB, ALPHA, BETA, ACOPY,
                BCOPY, RWK, CWK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Work array of size N^2 . The arrays `A` and `ACOPY` may be the same, in which case the first N^2 elements of `A` will be destroyed.

   BCOPY — Work array of size N^2 . The arrays `B` and `BCOPY` may be the same, in which case the first N^2 elements of `B` will be destroyed.

   RWK — Real work array of size `N`.

   CWK — Complex work array of size `N`.

   IWK — Integer work array of size `N`.

2. Integer Options with Chapter 10 Options Manager

1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine G3LRG, the internal or working leading dimension of ACOPY is increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL (4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine GVLRG . Analogous comments hold for BCOPY and the values IVAL(5) – IVAL(8) . Additional memory allocation and option value restoration are automatically done in GVLRG. There is no requirement that users change existing applications that use GVLRG or G3LRG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1.

Algorithm

Routine GVLRG computes the eigenvalues of the generalized eigensystem $Ax = \lambda Bx$ where A and B are real matrices of order N. The eigenvalues for this problem can be infinite; so instead of returning λ, GVLRG returns α and β. If β is nonzero, then $\lambda = \alpha/\beta$.

The first step of the QZ algorithm is to simultaneously reduce A to upper Hessenberg form and B to upper triangular form. Then, orthogonal transformations are used to reduce A to quasi-upper-triangular form while keeping B upper triangular. The generalized eigenvalues are then computed.

The routine GVLRG uses the QZ algorithm due to Moler and Stewart (1973), as implemented by the EISPACK routines QZHES, QZIT and QZVAL; see Garbow et al. (1977).

Example

In this example, DATA statements are used to set A and B. The eigenvalues are computed and printed.

```
      INTEGER    LDA, LDB, N
      PARAMETER  (N=3, LDA=N, LDB=N)
C
      INTEGER    I
      REAL       A(LDA,N), AMACH, B(LDB,N), BETA(N)
      COMPLEX    ALPHA(N), EVAL(N)
      EXTERNAL   AMACH, GVLRG, WRCRN
C
C                                  Set values of A and B
C                                  A = (  1.0     0.5    0.0  )
C                                      (-10.0     2.0    0.0  )
C                                      (  5.0     1.0    0.5  )
C
C                                  B = (  0.5     0.0    0.0  )
C                                      (  3.0     3.0    0.0  )
C                                      (  4.0     0.5    1.0  )
C
```

```
C                                 Declare variables
      DATA A/1.0, -10.0, 5.0, 0.5, 2.0, 1.0, 0.0, 0.0, 0.5/
      DATA B/0.5, 3.0, 4.0, 0.0, 3.0, 0.5, 0.0, 0.0, 1.0/
C
      CALL GVLRG (N, A, LDA, B, LDB, ALPHA, BETA)
C                                 Compute eigenvalues
      DO 10  I=1, N
            EVAL(I) = ALPHA(I)/BETA(I)
   10 CONTINUE
C                                 Print results
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
                         EVAL
              1                 2                 3
( 0.833, 1.993)  ( 0.833,-1.993)  ( 0.500, 0.000)
```

GVCRG/DGVCRG (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a generalized real eigensystem $Az = \lambda Bz$.

Usage

`CALL GVCRG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)`

Arguments

N — Order of the matrices `A` and `B`.   (Input)

A — Real matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

B — Real matrix of order `N`.   (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program.   (Input)

ALPHA — Complex vector of size `N` containing scalars $\alpha_{i.}$ If $\beta_i \neq 0$, $\lambda_i = \alpha_i / \beta_{i,}$ $i = 1, \ldots, n$ are the eigenvalues of the system.

BETA — Vector of size `N` containing scalars β_i.   (Output)

EVEC — Complex matrix of order `N`.   (Output)
The *J*-th eigenvector, corresponding to λ_J, is stored in the *J*-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program.   (Input)

Comments

1. Automatic workspace usage is

   GVCRG $3N^2 + 4N$ units, or
   DGVCRG $6N^2 + 7N$ units.

   Workspace may be explicitly provided, if desired, by use of G8CRG/DG8CRG. The reference is

```
CALL G8CRG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC,
            LDEVEC, ACOPY, BCOPY, ECOPY, RWK, CWK,
            IWK)
```

   The additional arguments are as follows:

   ACOPY — Work array of size N^2. The arrays A and ACOPY may be the same, in which case the first N^2 elements of A will be destroyed.

   BCOPY — Work array of size N^2. The arrays B and BCOPY may be the same, in which case the first N^2 elements of B will be destroyed.

   ECOPY — Work array of size N^2.

   RWK — Work array of size N.

   CWK — Complex work array of size N.

   IWK — Integer work array of size N.

2. Integer Options with Chapter 10 Options Manager

   1 This option uses eight values to solve memory bank conflict (access inefficiency) problems. In routine G8CRG, the internal or working leading dimensions of ACOPY and ECOPY are both increased by IVAL(3) when N is a multiple of IVAL(4). The values IVAL(3) and IVAL(4) are temporarily replaced by IVAL(1) and IVAL(2), respectively, in routine GVCRG. Analogous comments hold for the array BCOPY and the option values IVAL(5) – IVAL(8). Additional memory allocation and option value restoration are automatically done in GVCRG. There is no requirement that users change existing applications that use GVCRG or G8CRG. Default values for the option are IVAL(*) = 1, 16, 0, 1, 1, 16, 0, 1. Items 5–8 in IVAL(*) are for the generalized eigenvalue problem and are not used in GVCRG.

Algorithm

Routine GVCRG computes the complex eigenvalues and eigenvectors of the generalized eigensystem $Ax = \lambda Bx$ where A and B are real matrices of order N. The eigenvalues for this problem can be infinite; so instead of returning λ, GVCRG returns complex numbers α and real numbers β. If β is nonzero, then

$\lambda = \alpha/\beta$. For problems with small $|\beta|$ users can choose to solve the mathematically equivalent problem $Bx = \mu Ax$ where $\mu = \lambda^{-1}$.

The first step of the QZ algorithm is to simultaneously reduce *A* to upper Hessenberg form and *B* to upper triangular form. Then, orthogonal transformations are used to reduce *A* to quasi-upper-triangular form while keeping *B* upper triangular. The generalized eigenvalues and eigenvectors for the reduced problem are then computed.

The routine `GVCRG` is based on the QZ algorithm due to Moler and Stewart (1973), as implemented by the EISPACK routines `QZHES`, `QZIT` and `QZVAL`; see Garbow et al. (1977).

Example

In this example, `DATA` statements are used to set *A* and *B*. The eigenvalues, eigenvectors and performance index are computed and printed for the systems $Ax = \lambda Bx$ and $Bx = \mu Ax$ where $\mu = \lambda^{-1}$. For more details about the performance index, see routine `GPIRG` (page 396).

```
      INTEGER    LDA, LDB, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDB=N, LDEVEC=N)
C
      INTEGER    I, NOUT
      REAL       A(LDA,N), AMACH, B(LDB,N), BETA(N), GPIRG, PI
      COMPLEX    ALPHA(N), EVAL(N), EVEC(LDEVEC,N)
      EXTERNAL   AMACH, GPIRG, GVCRG, UMACH, WRCRN
C
C                                 Define values of A and B:
C                                 A = (  1.0     0.5    0.0  )
C                                     (-10.0     2.0    0.0  )
C                                     (  5.0     1.0    0.5  )
C
C                                 B = (  0.5     0.0    0.0  )
C                                     (  3.0     3.0    0.0  )
C                                     (  4.0     0.5    1.0  )
C
C                                 Declare variables
      DATA A/1.0, -10.0, 5.0, 0.5, 2.0, 1.0, 0.0, 0.0, 0.5/
      DATA B/0.5, 3.0, 4.0, 0.0, 3.0, 0.5, 0.0, 0.0, 1.0/
C
      CALL GVCRG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)
C                                 Compute eigenvalues
      DO 10  I=1, N
            EVAL(I) = ALPHA(I)/BETA(I)
   10 CONTINUE
C                                 Compute performance index
      PI = GPIRG(N,N,A,LDA,B,LDB,ALPHA,BETA,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN (’EVAL’, 1, N, EVAL, 1, 0)
      CALL WRCRN (’EVEC’, N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,’(/,A,F6.3)’) ’ Performance index = ’, PI
C                                 Solve for reciprocals of values
      CALL GVCRG (N,  B, LDB,A, LDA, ALPHA, BETA, EVEC, LDEVEC)
```

```
C                                  Compute reciprocals
      DO 20  I=1, N
            EVAL(I) = ALPHA(I)/BETA(I)
   20 CONTINUE
C                                  Compute performance index
      PI = GPIRG(N,N,B,LDB,A,LDA,ALPHA,BETA,EVEC,LDEVEC)
C                                  Print results
      CALL WRCRN ('EVAL reciprocals', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
                        EVAL
              1                 2                 3
( 0.833, 1.993)  ( 0.833,-1.993)  ( 0.500, 0.000)

                          EVEC
                 1                 2                 3
1  (-0.197, 0.150)  (-0.197,-0.150)  ( 0.000, 0.000)
2  (-0.069,-0.568)  (-0.069, 0.568)  ( 0.000, 0.000)
3  ( 0.782, 0.000)  ( 0.782, 0.000)  ( 1.000, 0.000)

Performance index =  0.384

               EVAL reciprocals
              1                 2                 3
( 2.000, 0.000)  ( 0.179, 0.427)  ( 0.179,-0.427)

                            EVEC
                 1                 2                 3
1  ( 0.000, 0.000)  (-0.197,-0.150)  (-0.197, 0.150)
2  ( 0.000, 0.000)  (-0.069, 0.568)  (-0.069,-0.568)
3  ( 1.000, 0.000)  ( 0.782, 0.000)  ( 0.782, 0.000)

Performance index =  0.283
```

GPIRG/DGPIRG (Single/Double precision)

Compute the performance index for a generalized real eigensystem $Az = \lambda Bz$.

Usage

`GPIRG(N, NEVAL, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)`

Arguments

N — Order of the matrices `A` and `B`.   (Input)

NEVAL — Number of eigenvalue/eigenvector pairs performance index computation is based on.   (Input)

A — Real matrix of order `N`.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

B — Real matrix of order N. (Input)

LDB — Leading dimension of B exactly as specified in the dimension statement in the calling program. (Input)

ALPHA — Complex vector of length NEVAL containing the numerators of eigenvalues. (Input)

BETA — Real vector of length NEVAL containing the denominators of eigenvalues. (Input)

EVEC — Complex N by NEVAL array containing the eigenvectors. (Input)

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement in the calling program. (Input)

GPIRG — Performance index. (Output)

Comments

1. Automatic workspace usage is

   GPIRG 4N units, or
   DGPIRG 8N units.

   Workspace may be explicitly provided, if desired, by use of G2IRG/DG2IRG. The reference is

   ```
   G2IRG(N, NEVAL, A, LDA, B, LDB, ALPHA, BETA, EVEC,
         LDEVEC, WK)
   ```

   The additional argument is

   WK — Complex work array of length 2N.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix A is zero.
3	4	The matrix B is zero.

3. The J-th eigenvalue should be ALPHA(J)/BETA(J), its eigenvector should be in the J-th column of EVEC.

Algorithm

Let M = NEVAL, x_j = EVEC(*,J) , the j-th column of EVEC. Also, let ε be the machine precision given by AMACH(4), page 1201. The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| \beta_j A x_j - \alpha_j B x_j \right\|_1}{\varepsilon \left(\left| \beta_j \right| \left\| A \right\|_1 + \left| \alpha_j \right| \left\| B \right\|_1 \right) \left\| x_j \right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector *v* is

$$\left\| v \right\|_1 = \sum_{i=1}^{N} \left\{ \left| \Re v_i \right| + \left| \Im v_i \right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of `EVCSF` (page 339) is considered excellent if τ < 1, good if 1 ≤ τ ≤ 100, and poor if τ > 100. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Garbow et al. (1977, pages 77−79).

Example

For an example of `GPIRG`, see routine `GVCRG` on page 393.

GVLCG/DGVLCG (Single/Double precision)

Compute all of the eigenvalues of a generalized complex eigensystem $Az = \lambda Bz$.

Usage

```
CALL GVLCG (N, A, LDA, B, LDB, ALPHA, BETA)
```

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Complex matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

B — Complex matrix of order `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program. (Input)

ALPHA — Complex vector of length `N`. Ultimately, alpha(*i*)/beta(*i*) (for *i* = 1, *n*), will be the eigenvalues of the system in decreasing order of magnitude. (Output)

BETA — Complex vector of length `N`. (Output)

Comments

1. Automatic workspace usage is

GVLCG $4N^2 + 4N$ units, or

DGVLCG $8N^2 + 7N$ units.

Workspace may be explicitly provided, if desired, by use of G3LCG/DG3LCG. The reference is

```
CALL G3LCG (N, A, LDA, B, LDB, ALPHA, BETA, ACOPY,
            BCOPY, CWK, WK, IWK)
```

The additional arguments are as follows:

ACOPY — Complex work array of length N^2. A and ACOPY may be the same, in which case A will be destroyed.

BCOPY — Complex work array of length N^2. B and BCOPY may be the same, in which case B will be destroyed.

CWK — Complex work array of length N.

WK — Real work array of length N.

IWK — Integer work array of length N.

2. Informational error

Type	Code	
4	1	The iteration for the eigenvalues failed to converge.

Algorithm

Routine GVLCG computes the eigenvalues of the generalized eigensystem $Ax = \lambda Bx$, where A and B are complex matrices of order n. The eigenvalues for this problem can be infinite; so instead of returning λ, GVLCG returns α and β. If β is nonzero, then $\lambda = \alpha/\beta$. If the eigenvectors are needed, then use GVCCG. See page 400.

The routine GVLCG is based on routines for the generalized complex eigenvalue problem by Garbow (1978). The QZ algorithm is described by Moler and Stewart (1973). Some timing information is given in Hanson et al. (1990).

Example

In this example, DATA statements are used to set A and B. Then, the eigenvalues are computed and printed.

```
C                                  Declaration of variables
      INTEGER    LDA, LDB, N
      PARAMETER  (N=5, LDA=N, LDB=N)
C
      INTEGER    I
      COMPLEX    A(LDA,N), ALPHA(N), B(LDB,N), BETA(N), EVAL(N)
      EXTERNAL   GVLCG, WRCRN
C
C                                  Define values of A and B
C
      DATA A/(-238.0,-344.0), (76.0,152.0), (118.0,284.0),
     &     (-314.0,-160.0), (-54.0,-24.0), (86.0,178.0),
     &     (-96.0,-128.0), (55.0,-182.0), (132.0,78.0),
     &     (-205.0,-400.0), (164.0,240.0), (40.0,-32.0),
     &     (-13.0,460.0), (114.0,296.0), (109.0,148.0),
     &     (-166.0,-308.0), (60.0,184.0), (34.0,-192.0),
     &     (-90.0,-164.0), (158.0,312.0), (56.0,158.0),
     &     (-60.0,-136.0), (-176.0,-214.0), (-424.0,-374.0),
     &     (-38.0,-96.0)/
      DATA B/(388.0,94.0), (-304.0,-76.0), (-658.0,-136.0),
     &     (-640.0,-10.0), (-162.0,-72.0), (-386.0,-122.0),
     &     (384.0,64.0), (-73.0,100.0), (204.0,-42.0), (631.0,158.0),
     &     (-250.0,-14.0), (-160.0,16.0), (-109.0,-250.0),
     &     (-692.0,-90.0), (131.0,52.0), (556.0,130.0),
     &     (-240.0,-92.0), (-118.0,100.0), (288.0,66.0),
     &     (-758.0,-184.0), (-396.0,-62.0), (240.0,68.0),
     &     (406.0,96.0), (-192.0,154.0), (278.0,76.0)/
C
      CALL GVLCG (N, A, LDA, B, LDB, ALPHA, BETA)
C                                  Compute eigenvalues
      DO 10  I=1, N
            EVAL(I) = ALPHA(I)/BETA(I)
   10 CONTINUE
C                                  Print results
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)

      STOP
      END
```

Output

```
                                 EVAL
               1                  2                  3                  4
(-1.000,-1.333)  ( 0.765, 0.941)  (-0.353, 0.412)  (-0.353,-0.412)

               5
(-0.353,-0.412)
```

GVCCG/DGVCCG (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of a generalized complex eigensystem $Az = \lambda Bz$.

Usage

`CALL GVCCG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)`

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Complex matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex matrix of order `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

ALPHA — Complex vector of length `N`. Ultimately, alpha(*i*)/beta(*i*) (for *i* = 1, …, *n*), will be the eigenvalues of the system in decreasing order of magnitude. (Output)

BETA — Complex vector of length `N`. (Output)

EVEC — Complex matrix of order `N`. (Output)
The `J`-th eigenvector, corresponding to `ALPHA(J)` = `BETA` (`J`), is stored in the `J`-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   `GVCCG` $4N^2 + 4N$ units, or
   `DGVCCG` $8N^2 + 7N$ units.

   Workspace may be explicitly provided, if desired, by use of `G6CCG`/`DG6CCG`. The reference is

   ```
   CALL G6CCG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC,
               LDEVEC, ACOPY, BCOPY, CWK, WK, IWK)
   ```

   The additional arguments are as follows:

   ACOPY — Complex work array of length N^2. `A` and `ACOPY` may be the same in which case the first N^2 elements of `A` will be destroyed.

   BCOPY — Complex work array of length N^2. `B` and `BCOPY` may be the same in which case the first N^2 elements of `B` will be destroyed.

   CWK — Complex work array of length `N`.

   WK — Real work array of length `N`.

   IWK — Integer work array of length `N`.

2. Informational error

Type	Code	
4	1	The iteration for an eigenvalue failed to converge.

3. The success of this routine can be checked using `GPICG` (page 403).

Algorithm

Routine `GVCCG` computes the eigenvalues and eigenvectors of the generalized eigensystem $Ax = \lambda Bx$. Here, A and B, are complex matrices of order n. The eigenvalues for this problem can be infinite; so instead of returning λ, GVCCG returns α and β. If β is nonzero, then λ = α / β.

The routine GVCCG uses the QZ algorithm described by Moler and Stewart (1973). The implementation is based on routines of Garbow (1978). Some timing results are given in Hanson et al. (1990).

Example

In this example, DATA statements are used to set *A* and *B*. The eigenvalues and eigenvectors are computed and printed. The performance index is also computed and printed. This serves as a check on the computations. For more details, see routine GPICG on page 403.

```
      INTEGER    LDA, LDB, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDB=N, LDEVEC=N)
C
      INTEGER    I, NOUT
      REAL       GPICG, PI
      COMPLEX    A(LDA,N), ALPHA(N), B(LDB,N), BETA(N), EVAL(N),
     &           EVEC(LDEVEC,N)
      EXTERNAL   GPICG, GVCCG, UMACH
C
C                                 Define values of A and B
C                                 A = (  1+0i   0.5+i   0+5i   )
C                                     (-10+0i     2+i   0+0i   )
C                                     (  5+i     1+0i   0.5+3i )
C
C                                 B = ( 0.5+0i     0+0i  0+0i  )
C                                     (   3+3i     3+3i   0+i  )
C                                     (   4+2i    0.5+i   1+i  )
C
C                                 Declare variables
      DATA A/(1.0,0.0), (-10.0,0.0), (5.0,1.0), (0.5,1.0), (2.0,1.0),
     &     (1.0,0.0), (0.0,5.0), (0.0,0.0), (0.5,3.0)/
      DATA B/(0.5,0.0), (3.0,3.0), (4.0,2.0), (0.0,0.0), (3.0,3.0),
     &     (0.5,1.0), (0.0,0.0), (0.0,1.0), (1.0,1.0)/
C                                 Compute eigenvalues
      CALL GVCCG (N, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)

      DO 10  I=1, N
            EVAL(I) = ALPHA(I)/BETA(I)
   10 CONTINUE
C                                 Compute performance index
      PI = GPICG(N,N,A,LDA,B,LDB,ALPHA,BETA,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRCRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRCRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT, '(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
                        EVAL
              1                2                3
( -8.18,-25.38)  (  2.18,  0.61)  (  0.12, -0.39)
```

```
                           EVEC
                   1                   2                   3
1  (-0.3267,-0.1245)  (-0.3007,-0.2444)  ( 0.0371, 0.1518)
2  ( 0.1767, 0.0054)  ( 0.8959, 0.0000)  ( 0.9577, 0.0000)
3  ( 0.9201, 0.0000)  (-0.2019, 0.0801)  (-0.2215, 0.0968)

Performance index =  0.709
```

GPICG/DGPICG (Single/Double precision)

Compute the performance index for a generalized complex eigensystem $Az = \lambda Bz$.

Usage

`GPICG(N, NEVAL, A, LDA, B, LDB, ALPHA, BETA, EVEC, LDEVEC)`

Arguments

N — Order of the matrices `A` and `B`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs performance index computation is based on. (Input)

A — Complex matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

B — Complex matrix of order `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program. (Input)

ALPHA — Complex vector of length `NEVAL` containing the numerators of eigenvalues. (Input)

BETA — Complex vector of length `NEVAL` containing the denominators of eigenvalues. (Input)

EVEC — Complex `N` by `NEVAL` array containing the eigenvectors. (Input)

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

GPICG — Performance index. (Output)

Comments

1. Automatic workspace usage is

   `GPICG` 4`N` units, or
   `DGPICG` 8`N` units.

Workspace may be explicitly provided, if desired, by use of G2ICG/DG2ICG. The reference is

```
G2ICG(N, NEVAL, A, LDA, B, LDB, ALPHA, BETA, EVEC,
      LDEVEC, WK)
```

The additional argument is

WK — Complex work array of length 2N.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix A is zero.
3	4	The matrix B is zero.

3. The J-th eigenvalue should be ALPHA(J)/BETA (J), its eigenvector should be in the J-th column of EVEC.

Algorithm

Let M = NEVAL, x_j = EVEC(*, J) , the j-th column of EVEC. Also, let ε be the machine precision given by AMACH(4). The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| \beta_j A x_j - \alpha_j B x_j \right\|_1}{\varepsilon \left(\left| \beta_j \right| \left\| A \right\|_1 + \left| \alpha_j \right| \left\| B \right\|_1 \right) \left\| x_j \right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector v is

$$\left\| v \right\|_1 = \sum_{i=1}^{N} \left\{ \left| \Re v_i \right| + \left| \Im v_i \right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of EVCSF (page 339) is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$.

The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Garbow et al. (1977, pages 77−79).

Example

For an example of GPICG, see routine GVCCG on page 400.

GVLSP/DGVLSP (Single/Double precision)

Compute all of the eigenvalues of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.

Usage

```
CALL GVLSP (N, A, LDA, B, LDB, EVAL)
```

Arguments

N — Order of the matrices `A` and `B`.   (Input)

A — Real symmetric matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program.   (Input)

B — Positive definite symmetric matrix of order `N`.   (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program.   (Input)

EVAL — Vector of length `N` containing the eigenvalues in decreasing order of magnitude.   (Output)

Comments

1. Automatic workspace usage is

   `GVLSP` $N^2 + 4N$ units, or
   `DGVLSP` $2N^2 + 7N$ units.

   Workspace may be explicitly provided, if desired, by use of `G3LSP`/`DG3LSP`. The reference is

   ```
   CALL G3LSP (N, A, LDA, B, LDB, EVAL, IWK, WK1, WK2)
   ```

   The additional arguments are as follows:

   IWK — Integer work array of length `N`.

   WK1 — Work array of length 2`N`.

   WK2 — Work array of length $N^2 + N$.

2. Informational errors

Type	Code	
4	1	The iteration for an eigenvalue failed to converge.
4	2	Matrix `B` is not positive definite.

Algorithm

Routine GVLSP computes the eigenvalues of $Ax = \lambda Bx$ with A symmetric and B symmetric positive definite. The Cholesky factorization $B = R^T R$, with R a triangular matrix, is used to transform the equation $Ax = \lambda Bx$ to

$$(R^{-T} AR^{-1})(Rx) = \lambda\ (Rx)$$

The eigenvalues of $C = R^{-T} AR^{-1}$ are then computed. This development is found in Martin and Wilkinson (1968). The Cholesky factorization of B is computed based on IMSL routine LFTDS, page 64. The eigenvalues of C are computed based on routine EVLSF, page 337. Further discussion and some timing results are given Hanson et al. (1990).

Example

In this example, a DATA statement is used to set the matrices A and B. The eigenvalues of the system are computed and printed.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N
      PARAMETER  (N=3, LDA=N, LDB=N)
C
      REAL       A(LDA,N), B(LDB,N), EVAL(N)
      EXTERNAL   GVLSP, WRRRN
C                                  Define values of A:
C                                  A = (  2    3    5  )
C                                      (  3    2    4  )
C                                      (  5    4    2  )
      DATA A/2.0, 3.0, 5.0, 3.0, 2.0, 4.0, 5.0, 4.0, 2.0/
C
C                                  Define values of B:
C                                  B = (  3    1    0  )
C                                      (  1    2    1  )
C                                      (  0    1    1  )
      DATA B/3.0, 1.0, 0.0, 1.0, 2.0, 1.0, 0.0, 1.0, 1.0/
C
C                                  Find eigenvalues
      CALL GVLSP (N, A, LDA, B, LDB, EVAL)
C                                  Print results
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      END
```

Output

```
         EVAL
     1        2        3
-4.717   4.393   -0.676
```

GVCSP/DGVCSP (Single/Double precision)

Compute all of the eigenvalues and eigenvectors of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.

Usage

```
CALL GVCSP (N, A, LDA, B, LDB, EVAL, EVEC, LDEVEC)
```

Arguments

N — Order of the matrices A and B. (Input)

A — Real symmetric matrix of order N. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

B — Positive definite symmetric matrix of order N. (Input)

LDB — Leading dimension of B exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Vector of length N containing the eigenvalues in decreasing order of magnitude. (Output)

EVEC — Matrix of order N. (Output)
The J-th eigenvector, corresponding to EVAL(J), is stored in the J-th column. Each vector is normalized to have Euclidean length equal to the value one.

LDEVEC — Leading dimension of EVEC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   GVCSP $N^2 + 5N$ units, or
   DGVCSP $2N^2 + 9N$ units.

   Workspace may be explicitly provided, if desired, by use of G3CSP/DG3CSP. The reference is

   ```
   CALL G3CSP (N, A, LDA, B, LDB, EVAL, EVEC, LDEVEC,
               IWK, WK1, WK2)
   ```

   The additional arguments are as follows:

   IWK — Integer work array of length N.

   WK1 — Work array of length 3N.

   WK2 — Work array of length $N^2 + N$.

2. Informational errors

Type	Code	
4	1	The iteration for an eigenvalue failed to converge.
4	2	Matrix B is not positive definite.

3. The success of this routine can be checked using GPISP (page 409).

Algorithm

Routine GVLSP (page 405) computes the eigenvalues and eigenvectors of $Az = \lambda Bz$, with A symmetric and B symmetric positive definite. The Cholesky factorization $B = R^T R$, with R a triangular matrix, is used to transform the equation $Az = \lambda Bz$, to

$$(R^{-T} AR^{-1})(Rz) = \lambda\ (Rz)$$

The eigenvalues and eigenvectors of $C = R^{-T} AR^{-1}$ are then computed. The generalized eigenvectors of A are given by $z = R^{-1}\ x$, where x is an eigenvector of C. This development is found in Martin and Wilkinson (1968). The Cholesky factorization is computed based on IMSL routine LFTDS, page 64. The eigenvalues and eigenvectors of C are computed based on routine EVCSF, page 339. Further discussion and some timing results are given Hanson et al. (1990).

Example

In this example, a DATA statement is used to set the matrices A and B. The eigenvalues, eigenvectors and performance index are computed and printed. For details on the performance index, see IMSL routine GPISP on page 409.

```
C                                 Declare variables
      INTEGER    LDA, LDB, LDEVEC, N
      PARAMETER  (N=3, LDA=N, LDB=N, LDEVEC=N)
C
      INTEGER    NOUT
      REAL       A(LDA,N), B(LDB,N), EVAL(N), EVEC(LDEVEC,N), GPISP, PI
      EXTERNAL   GPISP, GVCSP, UMACH, WRRRN
C                                 Define values of A:
C                                 A = (  1.1    1.2    1.4  )
C                                     (  1.2    1.3    1.5  )
C                                     (  1.4    1.5    1.6  )
      DATA A/1.1, 1.2, 1.4, 1.2, 1.3, 1.5, 1.4, 1.5, 1.6/
C
C                                 Define values of B:
C                                 B = (  2.0    1.0    0.0  )
C                                     (  1.0    2.0    1.0  )
C                                     (  0.0    1.0    2.0  )
      DATA B/2.0, 1.0, 0.0, 1.0, 2.0, 1.0, 0.0, 1.0, 2.0/
C
C                                 Find eigenvalues and vectors
      CALL GVCSP (N, A, LDA, B, LDB, EVAL, EVEC, LDEVEC)
C                                 Compute performance index
      PI = GPISP(N,N,A,LDA,B,LDB,EVAL,EVEC,LDEVEC)
C                                 Print results
      CALL UMACH (2, NOUT)
      CALL WRRRN ('EVAL', 1, N, EVAL, 1, 0)
      CALL WRRRN ('EVEC', N, N, EVEC, LDEVEC, 0)
      WRITE (NOUT,'(/,A,F6.3)') ' Performance index = ', PI
      END
```

Output

```
          EVAL
    1        2        3
1.386  -0.058  -0.003

             EVEC
          1         2         3
1   0.6431  -0.1147  -0.6817
2  -0.0224  -0.6872   0.7266
3   0.7655   0.7174  -0.0858

Performance index =  0.620
```

GPISP/DGPISP (Single/Double precision)

Compute the performance index for a generalized real symmetric eigensystem problem.

Usage

`GPISP(N, NEVAL, A, LDA, B, LDB, EVAL, EVEC, LDEVEC)`

Arguments

N — Order of the matrices `A` and `B`. (Input)

NEVAL — Number of eigenvalue/eigenvector pairs that the performance index computation is based on. (Input)

A — Symmetric matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement in the calling program. (Input)

B — Symmetric matrix of order `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement in the calling program. (Input)

EVAL — Vector of length `NEVAL` containing eigenvalues. (Input)

EVEC — `N` by `NEVAL` array containing the eigenvectors. (Input)

LDEVEC — Leading dimension of `EVEC` exactly as specified in the dimension statement in the calling program. (Input)

GPISP — Performance index. (Output)

Comments

1. Automatic workspace usage is

   GPISP 2 * N units, or
   DGPISP 4 * N units.

   Workspace may be explicitly provided, if desired, by use of G2ISP/DG2ISP. The reference is

   ```
   G2ISP(N, NEVAL, A, LDA, B, LDB, EVAL, EVEC, LDEVEC,
         WORK)
   ```

   The additional argument is

   WORK — Work array of length 2 * N.

2. Informational errors

Type	Code	
3	1	Performance index is greater than 100.
3	2	An eigenvector is zero.
3	3	The matrix A is zero.
3	4	The matrix B is zero.

3. The J-th eigenvalue should be ALPHA(J)/BETA(J), its eigenvector should be in the J-th column of EVEC.

Algorithm

Let M = NEVAL, λ = EVAL, x_j = EVEC(*, J) , the j-th column of EVEC. Also, let ε be the machine precision given by AMACH(4). The performance index, τ, is defined to be

$$\tau = \max_{1 \le j \le M} \frac{\left\| Ax_j - \lambda_j Bx_j \right\|_1}{\varepsilon \left(\left\| A \right\|_1 + \left| \lambda_j \right| \left\| B \right\|_1 \right) \left\| x_j \right\|_1}$$

The norms used are a modified form of the 1-norm. The norm of the complex vector v is

$$\left\| v \right\|_1 = \sum_{i=1}^{N} \left\{ \left| \Re v_i \right| + \left| \Im v_i \right| \right\}$$

While the exact value of τ is highly machine dependent, the performance of EVCSF (page 339) is considered excellent if $\tau < 1$, good if $1 \le \tau \le 100$, and poor if $\tau > 100$. The performance index was first developed by the EISPACK project at Argonne National Laboratory; see Garbow et al. (1977, pages 77−79).

Example

For an example of GPISP, see routine GVCSP on page 407.

Chapter 3: Interpolation and Approximation

Routines

3.1. Cubic Spline Interpolation

3.2. Cubic Spline Evaluation and Integration

3.3. B-spline Interpolation

3.4. Spline Evaluation, Integration, and Conversion to Piecewise Polynomial Given the B-spline Representation

Usage Notes

The majority of the routines in this chapter produce piecewise polynomial or spline functions that either interpolate or approximate given data, or are support routines for the evaluation, integration, and conversion from one representation to another. Two major subdivisions of routines are provided. The cubic spline routines begin with the letters “CS” and utilize the piecewise polynomial representation described below. The B-spline routines begin with the letters

“BS” and utilize the B-spline representation described below. Most of the spline routines are based on routines in the book by de Boor (1978).

Piecewise Polynomials

A univariate piecewise polynomial (function) p is specified by giving its breakpoint sequence $\xi \in \mathbf{R}^n$, the order k (degree $k - 1$) of its polynomial pieces, and the $k \times (n - 1)$ matrix c of its local polynomial coefficients. In terms of this information, the piecewise polynomial (pp) function is given by

$$p(x) = \sum_{j=1}^{k} c_{ji} \frac{(x - \xi_i)^{j-1}}{(j-1)!} \qquad \text{for } \xi_i \le x < \xi_{i+1}$$

The breakpoint sequence ξ is assumed to be strictly increasing, and we extend the pp function to the entire real axis by extrapolation from the first and last intervals. The subroutines in this chapter will consistently make the following identifications for FORTRAN variables:

$$c = \text{PPCOEF}$$

$$\xi = \text{BREAK}$$

$$k = \text{KORDER}$$

$$N = \text{NBREAK}$$

This representation is redundant when the pp function is known to be smooth. For example, if p is known to be continuous, then we can compute $c_{1,i+1}$ from the c_{ji} as follows

$$c_{1,i+1} = p(\xi_{i+1}) = c_{1i} + c_{2i}\Delta\xi_i + \ldots + c_{ki} \frac{(\Delta\xi_i)^{k-1}}{(k-1)!}$$

where $\Delta\xi_i := \xi_{i+1} - \xi_i$. For smooth pp, we prefer to use the irredundant representation in terms of the B-(for ‘basis’)-splines, at least when such a function is first to be determined. The above pp representation is employed for evaluation of the pp function at many points since it is more efficient.

Splines and B-splines

B-splines provide a particularly convenient and suitable basis for a given class of smooth pp functions. Such a class is specified by giving its breakpoint sequence, its order, and the required smoothness across each of the interior breakpoints. The corresponding B-spline basis is specified by giving its knot sequence $\mathbf{t} \in \mathbf{R}^M$. The specification rule is the following: If the class is to have all derivatives up to and including the j-th derivative continuous across the interior breakpoint ξ_i, then the number ξ_i should occur $k - j - 1$ times in the knot sequence. Assuming that ξ_1, and ξ_n are the endpoints of the interval of interest, one chooses the first k knots equal to ξ_1 and the last k knots equal to ξ_n. This can be done since the B-splines are defined to be right continuous near ξ_1 and left continuous near ξ_n.

When the above construction is completed, we will have generated a knot sequence $\boldsymbol{t}$ of length M; and there will be $m := M - k$ B-splines of order k, say B_1 , …, B_m that span the pp functions on the interval with the indicated smoothness. That is, each pp function in this class has a unique representation

$$p = a_1 B_1 + a_2 B_2 + \ldots + a_m B_m$$

as a linear combination of B-splines. The B-spline routines will consistently make use of the following identifiers for FORTRAN variables:

$$a = \text{BSCOEF}$$

$$\mathbf{t} = \text{XKNOT}$$

$$m = \text{NCOEF}$$

$$M = \text{NKNOT}$$

A B-spline is a particularly compact pp function. B_i is a nonnegative function that is nonzero only on the interval $[t_i, t_{i+k}]$. More precisely, the support of the i-th B-spline is $[t_i, t_{i+k}]$. No pp function in the same class (other than the zero function) has smaller support (i.e., vanishes on more intervals) than a B-spline. This makes B-splines particularly attractive basis functions since the influence of any particular B-spline coefficient extends only over a few intervals. When it is necessary to emphasize the dependence of the B-spline on its parameters, we will use the notation

$$B_{i,k,t}$$

to denote the i-th B-spline of order k for the knot sequence $\mathbf{t}$.

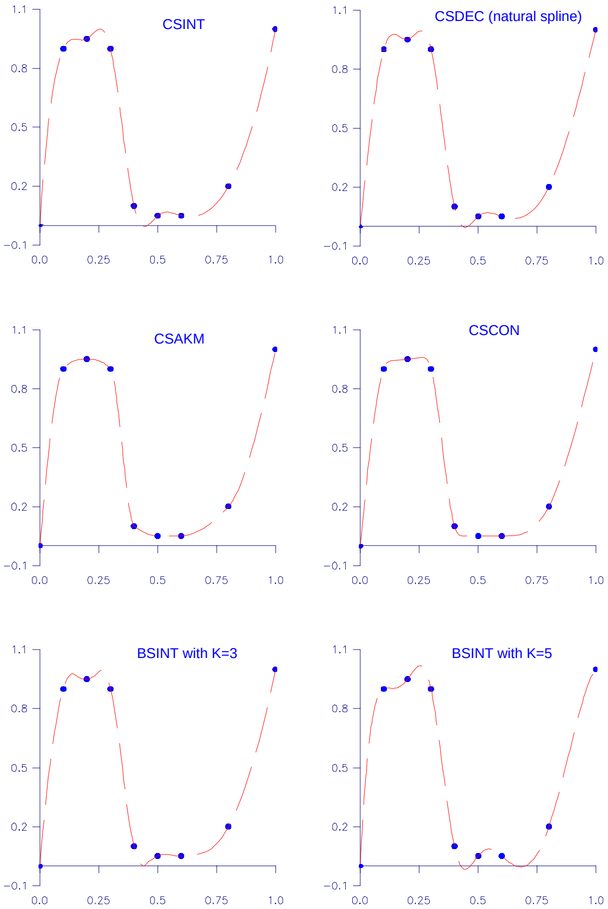


Figure 3-1 Spline Interpolants of the Same Data

Cubic Splines

Cubic splines are smooth (i.e., C^1 or C^2) fourth-order pp functions. For historical and other reasons, cubic splines are the most heavily used pp functions. Therefore, we provide special routines for their construction and evaluation. The routines for their determination use yet another representation (in terms of value and slope at all the breakpoints) but output the pp representation as described above for general pp functions.

We provide seven cubic spline interpolation routines: `CSIEZ` (page 420), `CSINT` (page 423), `CSDEC` (page 425), `CSHER` (page 429), `CSAKM` (page 432), `CSCON` (page 434), and `CSPER` (page 438). The first routine, `CSIEZ`, is an easy-to-use version of `CSINT` coupled with `CSVAL`. The routine `CSIEZ` will compute the value of the cubic spline interpolant (to given data using the ‘not-a-knot’ criterion) on a grid. The routine CSDEC allows the user to specify various endpoint conditions (such as the value of the first or second derivative at the right and left points). This means that the natural cubic spline can be obtained using this routine by setting the second derivative to zero at both endpoints. If function values and derivatives are available, then the Hermite cubic interpolant can be computed using CSHER. The two routines CSAKM and CSCON are designed so that the shape of the curve matches the shape of the data. In particular, CSCON preserves the convexity of the data while CSAKM attempts to minimize oscillations. If the data is periodic, then CSPER will produce a periodic interpolant. The routine CONFT (page 551) allows the user wide latitude in enforcing shapes. This routine returns the B-spline representation.

It is possible that the cubic spline interpolation routines will produce unsatisfactory results. The adventurous user should consider using the B-spline interpolation routine BSINT that allows one to choose the knots and order of the spline interpolant.

In Figure 3-1, we display six spline interpolants to the same data. This data can be found in Example 1 of the IMSL routine CSCON (page 434) Notice the different characteristics of the interpolants. The interpolation routines CSAKM (page 432) and CSCON are the only two that attempt to preserve the shape of the data. The other routines tend to have extraneous inflection points, with the piecewise quartic (k = 5) exhibiting the most oscillation.

Tensor Product Splines

The simplest method of obtaining multivariate interpolation and approximation routines is to take univariate methods and form a multivariate method via tensor products. In the case of two-dimensional spline interpolation, the development proceeds as follows: Let $\mathbf{t}_x$ be a knot sequence for splines of order k_x, and $\mathbf{t}_y$ be a knot sequence for splines of order k_y. Let $N_x + k_x$ be the length of $\mathbf{t}_x$, and $N_y + k_y$ be the length of $\mathbf{t}_y$. Then, the tensor product spline has the form

$$\sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}(x) B_{m,k_y,\mathbf{t}_y}(y)$$

Given two sets of points

$$\{x_i\}_{i=1}^{N_x} \text{ and } \{y_i\}_{i=1}^{N_y}$$

for which the corresponding univariate interpolation problem could be solved, the tensor product interpolation problem becomes: Find the coefficients c_{nm} so that

$$\sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}(x_i) B_{m,k_y,\mathbf{t}_y}(y_i) = f_{ij}$$

This problem can be solved efficiently by repeatedly solving univariate interpolation problems as described in de Boor (1978, page 347). Three-dimensional interpolation has analogous behavior. In this chapter, we provide routines that compute the two-dimensional tensorproduct spline coefficients given two-dimensional interpolation data (BS2IN, page 459), compute the three-dimensional tensor-product spline coefficients given three-dimensional interpolation data (BS3IN, page 464) compute the two-dimensional tensor-product spline coefficients for a tensor-product least squares problem (BSLS2, page 561), and compute the three-dimensional tensor-product spline coefficients for a tensor-product least squares problem (BSLS3, page 566). In addition, we provide evaluation, differentiation, and integration routines for the twoand three-dimensional tensor-product spline functions. The relevant routines are BS2VL (page 479), BS3VL (page 490), BS2DR (page 480), BS3DR (page 491), BS2GD (page 483), BS3GD (page 495), BS2IG (page 487), and BS3IG (page 500).

Quadratic Interpolation

The routines that begin with the letters “QD” in this chapter are designed to interpolate a one-, two-, or three-dimensional (tensor product) table of values and return an approximation to the value of the underlying function or one of its derivatives at a given point. These routines are all based on quadratic polynomial interpolation.

Scattered Data Interpolation

We have one routine, SURF, that will return values of an interpolant to scattered data in the plane. This routine is based on work by Akima (1978), which utilizes C^1 piecewise quintics on a triangular mesh.

Least Squares

Routines are provided to smooth noisy data: regression using linear polynomials (RLINE), regression using arbitrary polynomials (RCURV, page 535), and regression using user-supplied functions (FNLSQ, page 539). Additional routines compute the least-squares fit using splines with fixed knots (BSLSQ, page 543) or free knots (BSVLS, page 547). These routines can produce cubic-spline least-squares fit simply by setting the order to 4. The routine CONFT (page 551) computes a fixed-knot spline weighted least-squares fit subject to linear constraints. This routine is very general and is recommended if issues of shape are important. The two- and three-dimensional tensor-product spline regression routines are (BSLS2, page 561) and (BSLS3, page 566).

Smoothing by Cubic Splines

Two “smoothing spline” routines are provided. The routine CSSMH (page 575) returns the cubic spline that smooths the data, given a smoothing parameter chosen by the user. Whereas, CSSCV (page 578) estimates the smoothing parameter by cross-validation and then returns the cubic spline that smooths the data. In this sense, CSSCV is the easier of the two routines to use. The routine CSSED (page 572) returns a smoothed data vector approximating the values of the underlying function when the data are contaminated by a few random spikes.

Rational Chebyshev Approximation

The routine RATCH (page 581) computes a rational Chebyshev approximation to a user-supplied function. Since polynomials are rational functions, this routine can be used to compute best polynomial approximations.

Using the Univariate Spline Routines

An easy to use spline interpolation routine CSIEZ (page 420) is provided . This routine computes an interpolant and returns the values of the interpolant on a user-supplied grid. A slightly more advanced routine SPLEZ (page 447) computes (at the users discretion) one of several interpolants or least-squares fits and returns function values or derivatives on a user-supplied grid.

For more advanced uses of the interpolation (or least squares) spline routines, one first forms an interpolant from interpolation (or least-squares) data. Then it must be evaluated, differentiated, or integrated once the interpolant has been formed. One way to perform these tasks, using cubic splines with the ‘not-a-knot’ end condition, is to call CSINT to obtain the local coefficients of the piecewise cubic interpolant and then call CSVAL to evaluate the interpolant. A more complicated situation arises if one wants to compute a quadratic spline interpolant and then evaluate it (efficiently) many times. Typically, the sequence of routines called might be BSNAK (get the knots), BSINT (returns the B-spline coefficients of the interpolant), BSCPP (convert to pp form), and PPVAL (evaluate). The last two calls could be replaced by a call to the B-spline grid evaluator BS1GD, or the last call could be replaced with pp grid evaluator PP1GD. The interconnection of the spline routines is summarized in Figure 3-2.

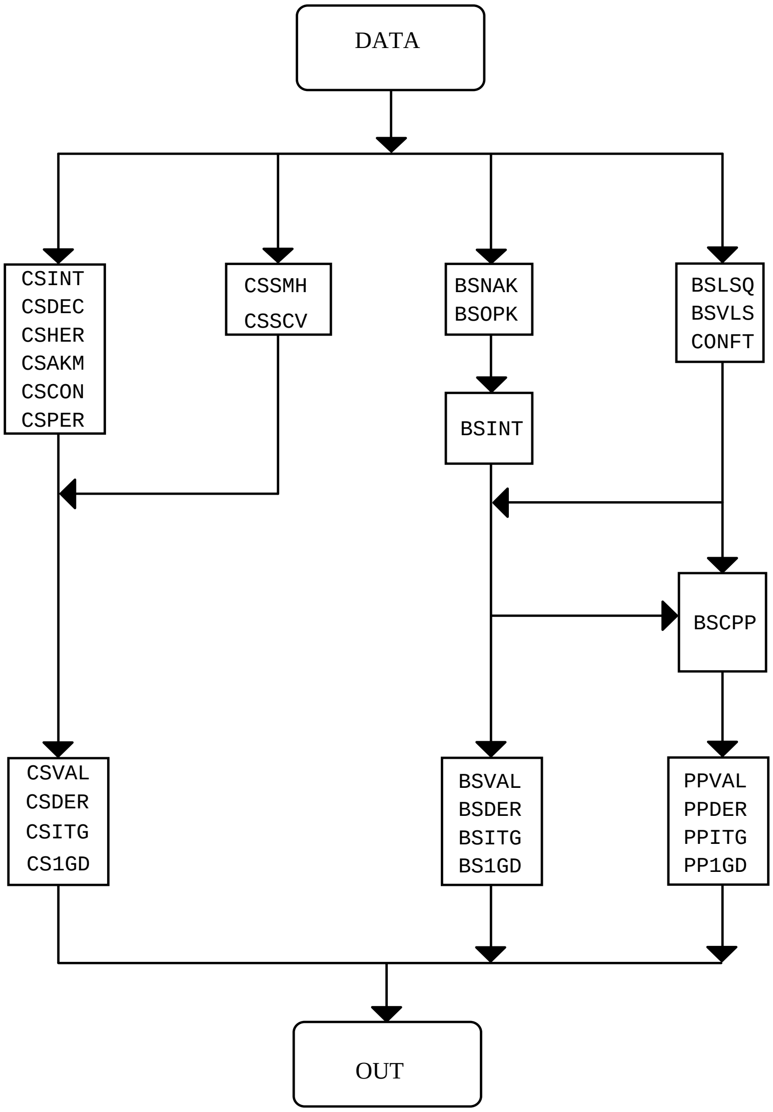


Figure 3-2   Interrelation of the Spline Routines

Choosing an Interpolation Routine

The choice of an interpolation routine depends both on the type of data and on the use of the interpolant. We provide 18 interpolation routines. These routines are depicted in a decision tree in Figure 3-3. This figure provides a guide for selecting an appropriate interpolation routine. For example, if periodic one-dimensional (univariate) data is available, then the path through *univariate* to *periodic* leads to the IMSL routine CSPER, which is the proper routine for this setting. The general-purpose univariate interpolation routines can be found in the box beginning with CSINT. Two- and three-dimensional tensor-product interpolation routines are also provided. For two-dimensional scattered data, the appropriate routine is SURF .

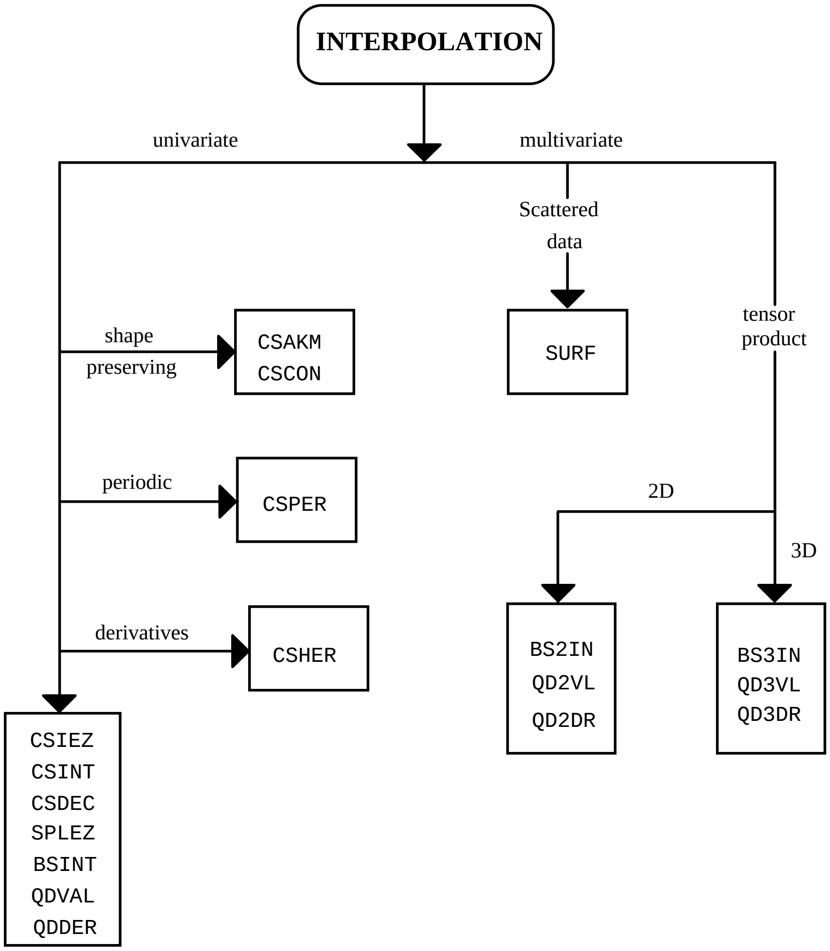


Figure 3-3 Choosing an Interplation Routine

CSIEZ/DCSIEZ (Single/Double precision)

Compute the cubic spline interpolant with the 'not-a-knot' condition and return values of the interpolant at specified points.

Usage

```
CALL CSIEZ (NDATA, XDATA, FDATA, N, XVEC, VALUE)
```

Arguments

NDATA — Number of data points. (Input)
NDATA must be at least 2.

XDATA — Array of length NDATA containing the data point abscissas. (Input)
The data point abscissas must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

N — Length of vector XVEC. (Input)

XVEC — Array of length N containing the points at which the spline is to be evaluated. (Input)

VALUE — Array of length N containing the values of the spline at the points in XVEC. (Output)

Comments

Automatic workspace usage is

CSIEZ MAX0(N, NDATA) + 3 * N + 5 * NDATA units, or
DCSIEZ MAX0(N, NDATA) + 5 * N + 10 * NDATA units.

Workspace may be explicitly provided, if desired, by use of C2IEZ/DC2IEZ. The reference is

```
CALL C2IEZ (NDATA, XDATA, FDATA, N, XVEC, VALUE, IWK, WK1,
           WK2)
```

The additional arguments are as follows:

IWK — Integer work array of length MAX0(N, NDATA) + N.

WK1 — Real work array of length 5 * NDATA.

WK2 — Real work array of length 2 * N.

Algorithm

This routine is designed to let the user easily compute the values of a cubic spline interpolant. The routine CSIEZ computes a spline interpolant to a set of data points (x_i, f_i) for i = 1, …, NDATA. The output for this routine consists of a vector of values of the computed cubic spline. Specifically, let $n = N$, v = XVEC, and y = VALUE, then if s is the computed spline we set

$$y_j = s(v_j)\, j = 1, \ldots, n$$

Additional documentation can be found by referring to the IMSL routines CSINT (page 423) or SPLEZ (page 447).

Example

In this example, a cubic spline interpolant to a function F is computed. The values of this spline are then compared with the exact function values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=11)
C
      INTEGER    I, NOUT
      REAL       F, FDATA(NDATA), FLOAT, SIN, VALUE(2*NDATA-1), X,
     &           XDATA(NDATA), XVEC(2*NDATA-1)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSIEZ, UMACH
C                                 Define function
      F(X) = SIN(15.0*X)
C                                 Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
      DO 20  I=1, 2*NDATA - 1
         XVEC(I) = FLOAT(I-1)/FLOAT(2*NDATA-2)
   20 CONTINUE
C                                 Compute cubic spline interpolant
      CALL CSIEZ (NDATA, XDATA, FDATA, 2*NDATA-1, XVEC, VALUE)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99998)
99998 FORMAT (13X, ’X’, 9X, ’INTERPOLANT’, 5X, ’ERROR’)
C                                 Print the interpolant and the error
C                                 on a finer grid
      DO 30  I=1, 2*NDATA - 1
         WRITE (NOUT,99999) XVEC(I), VALUE(I), F(XVEC(I)) - VALUE(I)
   30 CONTINUE
99999 FORMAT(’ ’, 2F15.3, F15.6)
      END
```

Output

```
  X         INTERPOLANT     ERROR
0.000          0.000       0.000000
0.050          0.809      -0.127025
0.100          0.997       0.000000
0.150          0.723       0.055214
0.200          0.141       0.000000
0.250         -0.549      -0.022789
0.300         -0.978       0.000000
0.350         -0.843      -0.016246
0.400         -0.279       0.000000
0.450          0.441       0.009348
0.500          0.938       0.000000
0.550          0.903       0.019947
0.600          0.412       0.000000
0.650         -0.315      -0.004895
0.700         -0.880       0.000000
0.750         -0.938      -0.029541
0.800         -0.537       0.000000
0.850          0.148       0.034693
0.900          0.804       0.000000
0.950          1.086      -0.092559
1.000          0.650       0.000000
```

CSINT/DCSINT (Single/Double precision)

Compute the cubic spline interpolant with the ‘not-a-knot’ condition.

Usage

```
CALL CSINT (NDATA, XDATA, FDATA, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)
`NDATA` must be at least 2.

XDATA — Array of length `NDATA` containing the data point abscissas. (Input)
The data point abscissas must be distinct.

FDATA — Array of length `NDATA` containing the data point ordinates. (Input)

BREAK — Array of length `NDATA` containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by `NDATA` containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   `CSINT  NDATA` units, or
   `DCSINT NDATA` units.

   Workspace may be explicitly provided, if desired, by use of `C2INT`/`DC2INT`. The reference is

   ```
   CALL C2INT (NDATA, XDATA, FDATA, BREAK, CSCOEF, IWK)
   ```

   The additional argument is

   IWK — Work array of length `NDATA`.

2. The cubic spline can be evaluated using `CSVAL` (page 440); its derivative can be evaluated using `CSDER` (page 441).

3. Note that column `NDATA` of `CSCOEF` is used as workspace.

Algorithm

The routine `CSINT` computes a C^2 cubic spline interpolant to a set of data points (x_i, f_i) for $i = 1, \ldots,$ `NDATA` $= N$. The breakpoints of the spline are the abscissas. Endpoint conditions are automatically determined by the program. These conditions correspond to the “not-a-knot” condition (see de Boor 1978), which requires that the third derivative of the spline be continuous at the second and next-to-last breakpoint. If N is 2 or 3, then the linear or quadratic interpolating polynomial is computed, respectively.

If the data points arise from the values of a smooth (say C^4) function f, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\| f - s \|_{[\xi_1,\xi_N]} \le C \left\| f^{(4)} \right\|_{[\xi_1,\xi_N]} |\xi|^4$$

where

$$|\xi| := \max_{i=2,\ldots,N} |\xi_i - \xi_{i-1}|$$

For more details, see de Boor (1978, pages 55−56).

Example

In this example, a cubic spline interpolant to a function F is computed. The values of this spline are then compared with the exact function values.

```
C                                  Specifications
      INTEGER    NDATA
      PARAMETER  (NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, F,
     &           FDATA(NDATA), FLOAT, SIN, X, XDATA(NDATA)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSINT, CSVAL, UMACH
C                                  Define function
      F(X) = SIN(15.0*X)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Compute cubic spline interpolant
      CALL CSINT (NDATA, XDATA, FDATA, BREAK, CSCOEF)
C                                  Get output unit number.
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
      NINTV = NDATA - 1
C                                  Print the interpolant and the error
C                                  on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = FLOAT(I-1)/FLOAT(2*NDATA-2)
         WRITE (NOUT,’(2F15.3,F15.6)’) X, CSVAL(X,NINTV,BREAK,CSCOEF),
     &                                  F(X) - CSVAL(X,NINTV,BREAK,
     &                                  CSCOEF)
   20 CONTINUE
      END
```

Output

```
  X         Interpolant     Error
0.000          0.000       0.000000
0.050          0.809      -0.127025
0.100          0.997       0.000000
0.150          0.723       0.055214
0.200          0.141       0.000000
0.250         -0.549      -0.022789
0.300         -0.978       0.000000
0.350         -0.843      -0.016246
0.400         -0.279       0.000000
0.450          0.441       0.009348
0.500          0.938       0.000000
0.550          0.903       0.019947
0.600          0.412       0.000000
0.650         -0.315      -0.004895
0.700         -0.880       0.000000
0.750         -0.938      -0.029541
0.800         -0.537       0.000000
0.850          0.148       0.034693
0.900          0.804       0.000000
0.950          1.086      -0.092559
1.000          0.650       0.000000
```

CSDEC/DCSDEC (Single/Double precision)

Compute the cubic spline interpolant with specified derivative endpoint conditions.

Usage

```
CALL CSDEC (NDATA, XDATA, FDATA, ILEFT, DLEFT, IRIGHT,
            DRIGHT, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)

XDATA — Array of length NDATA containing the data point abscissas. (Input) The data point abscissas must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

ILEFT — Type of end condition at the left endpoint. (Input)

ILEFT	**Condition**
0	“Not-a-knot” condition
1	First derivative specified by DLEFT
2	Second derivative specified by DLEFT

DLEFT — Derivative at left endpoint if ILEFT is equal to 1 or 2. (Input) If ILEFT = 0, then DLEFT is ignored.

IRIGHT — Type of end condition at the right endpoint. (Input)

`IRIGHT`	**Condition**
0	“Not-a-knot” condition
1	First derivative specified by `DRIGHT`
2	Second derivative specified by `DRIGHT`

DRIGHT — Derivative at right endpoint if `IRIGHT` is equal to 1 or 2. (Input) If `IRIGHT` = 0 then `DRIGHT` is ignored.

BREAK — Array of length `NDATA` containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by `NDATA` containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   `CSDEC  NDATA` units, or
   `DCSDEC NDATA` units.

   Workspace may be explicitly provided, if desired, by use of `C2DEC`/`DC2DEC`. The reference is

   ```
   CALL C2DEC (NDATA, XDATA, FDATA, ILEFT, DLEFT,
               IRIGHT, DRIGHT, BREAK, CSCOEF, IWK)
   ```

   The additional argument is

   IWK — Work array of length `NDATA`.

2. The cubic spline can be evaluated using `CSVAL` (page 440); its derivative can be evaluated using `CSDER` (page 441).

3. Note that column `NDATA` of `CSCOEF` is used as workspace.

Algorithm

The routine `CSDEC` computes a C^2 cubic spline interpolant to a set of data points (x_i, f_i) for $i = 1, \ldots,$ `NDATA` $= N$. The breakpoints of the spline are the abscissas. Endpoint conditions are to be selected by the user. The user may specify not-a-knot, first derivative, or second derivative at each endpoint (see de Boor 1978, Chapter 4).

If the data (including the endpoint conditions) arise from the values of a smooth (say C^4) function f, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\left\| f - s \right\|_{[\xi_1,\xi_N]} \le C \left\| f^{(4)} \right\|_{[\xi_1,\xi_N]} \left| \xi \right|^4$$

where

$$|\xi| := \mathop{}_{i=2,\ldots,N}|\xi_i - \xi_{i-1}|$$

For more details, see de Boor (1978, Chapter 4 and 5).

Example 1

In Example 1, a cubic spline interpolant to a function *f* is computed. The value of the derivative at the left endpoint and the value of the second derivative at the right endpoint are specified. The values of this spline are then compared with the exact function values.

```
      INTEGER    ILEFT, IRIGHT, NDATA
      PARAMETER  (ILEFT=1, IRIGHT=2, NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), COS, CSCOEF(4,NDATA), CSVAL, DLEFT,
     &           DRIGHT, F, FDATA(NDATA), FLOAT, SIN, X, XDATA(NDATA)
      INTRINSIC  COS, FLOAT, SIN
      EXTERNAL   CSDEC, CSVAL, UMACH
C                                  Define function
      F(X) = SIN(15.0*X)
C                                  Initialize DLEFT and DRIGHT
      DLEFT  = 15.0*COS(15.0*0.0)
      DRIGHT = -15.0*15.0*SIN(15.0*1.0)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Compute cubic spline interpolant
      CALL CSDEC (NDATA, XDATA, FDATA, ILEFT, DLEFT, IRIGHT, DRIGHT,
     &            BREAK, CSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
      NINTV = NDATA - 1
C                                  Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = FLOAT(I-1)/FLOAT(2*NDATA-2)
         WRITE (NOUT,’(2F15.3,F15.6)’) X, CSVAL(X,NINTV,BREAK,CSCOEF),
     &                                 F(X) - CSVAL(X,NINTV,BREAK,
     &                                 CSCOEF)
   20 CONTINUE
      END
```

Output

```
   X         Interpolant     Error
0.000          0.000       0.000000
0.050          0.675       0.006332
0.100          0.997       0.000000
0.150          0.759       0.019485
0.200          0.141       0.000000
0.250         -0.558      -0.013227
0.300         -0.978       0.000000
```

```
0.350         -0.840      -0.018765
0.400         -0.279       0.000000
0.450          0.440       0.009859
0.500          0.938       0.000000
0.550          0.902       0.020420
0.600          0.412       0.000000
0.650         -0.312      -0.007301
0.700         -0.880       0.000000
0.750         -0.947      -0.020391
0.800         -0.537       0.000000
0.850          0.182       0.000497
0.900          0.804       0.000000
0.950          0.959       0.035074
1.000          0.650       0.000000
```

Example 2

In Example 2, we compute the *natural* cubic spline interpolant to a function *f* by forcing the second derivative of the interpolant to be zero at both endpoints. As in the previous example, we compare the exact function values with the values of the spline.

```
      INTEGER    ILEFT, IRIGHT, NDATA
      PARAMETER  (ILEFT=2, IRIGHT=2, NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, DLEFT, DRIGHT,
     &           F, FDATA(NDATA), FLOAT, SIN, X, XDATA(NDATA)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSDEC, CSVAL, UMACH
C                                 Initialize DLEFT and DRIGHT
      DATA DLEFT/0./, DRIGHT/0./
C                                 Define function
      F(X) = SIN(15.0*X)
C                                 Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Compute cubic spline interpolant
      CALL CSDEC (NDATA, XDATA, FDATA, ILEFT, DLEFT, IRIGHT, DRIGHT,
     &            BREAK, CSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
      NINTV = NDATA - 1
C                                 Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = FLOAT(I-1)/FLOAT(2*NDATA-2)
         WRITE (NOUT,’(2F15.3,F15.6)’) X, CSVAL(X,NINTV,BREAK,CSCOEF),
     &                                 F(X) - CSVAL(X,NINTV,BREAK,
     &                                 CSCOEF)
   20 CONTINUE
      END
```

Output

```
    X          Interpolant     Error
0.000            0.000       0.000000
0.050            0.667       0.015027
0.100            0.997       0.000000
0.150            0.761       0.017156
0.200            0.141       0.000000
0.250           -0.559      -0.012609
0.300           -0.978       0.000000
0.350           -0.840      -0.018907
0.400           -0.279       0.000000
0.450            0.440       0.009812
0.500            0.938       0.000000
0.550            0.902       0.020753
0.600            0.412       0.000000
0.650           -0.311      -0.008586
0.700           -0.880       0.000000
0.750           -0.952      -0.015585
0.800           -0.537       0.000000
```

CSHER/DCSHER (Single/Double precision)

Compute the Hermite cubic spline interpolant.

Usage

```
CALL CSHER (NDATA, XDATA, FDATA, DFDATA, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)

XDATA — Array of length `NDATA` containing the data point abscissas. (Input) The data point abscissas must be distinct.

FDATA — Array of length `NDATA` containing the data point ordinates. (Input)

DFDATA — Array of length `NDATA` containing the values of the derivative. (Input)

BREAK — Array of length `NDATA` containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by `NDATA` containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   `CSHER  NDATA` units, or
   `DCSHER NDATA` units.

   Workspace may be explicitly provided, if desired, by use of `C2HER`/`DC2HER`. The reference is

```
CALL C2HER (NDATA, XDATA, FDATA, DFDATA, BREAK,
            CSCOEF, IWK)
```

The additional argument is

IWK — Work array of length `NDATA`.

2. Informational error

Type	Code	
4	2	The `XDATA` values must be distinct.

3. The cubic spline can be evaluated using `CSVAL` (page 440); its derivative can be evaluated using `CSDER` (page 441).

4. Note that column `NDATA` of `CSCOEF` is used as workspace.

Algorithm

The routine `CSHER` computes a C^1 cubic spline interpolant to the set of data points

$$\left(x_i, f_i\right) \text{and} \left(x_i, f_i'\right)$$

for $i = 1, \ldots,$ `NDATA` = `N`. The breakpoints of the spline are the abscissas.

If the data points arise from the values of a smooth (say C^4) function f, i.e.,

$$f_i = f\left(x_i\right) \text{and } f_i' = f'\left(x_i\right)$$

then the error will behave in a predictable fashion. Let ξ be the

breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\left\| f - s \right\|_{\left[\xi_1, \xi_N\right]} \le C \left\| f^{(4)} \right\|_{\left[\xi_1, \xi_N\right]} \left|\xi\right|^4$$

where

$$\left|\xi\right| := \underset{i=2,\ldots,N}{} \left|\xi_i - \xi_{i-1}\right|$$

For more details, see de Boor (1978, page 51).

Example

In this example, a cubic spline interpolant to a function f is computed. The value of the function f and its derivative f' are computed on the interpolation nodes and passed to `CSHER`. The values of this spline are then compared with the exact function values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), COS, CSCOEF(4,NDATA), CSVAL, DF,
     &           DFDATA(NDATA), F, FDATA(NDATA), FLOAT, SIN, X,
     &           XDATA(NDATA)
      INTRINSIC  COS, FLOAT, SIN
      EXTERNAL   CSHER, CSVAL, UMACH
C                                  Define function and derivative
      F(X)  = SIN(15.0*X)
      DF(X) = 15.0*COS(15.0*X)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I)  = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I)  = F(XDATA(I))
         DFDATA(I) = DF(XDATA(I))
   10 CONTINUE
C                                  Compute cubic spline interpolant
      CALL CSHER (NDATA, XDATA, FDATA, DFDATA, BREAK, CSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, 'X', 9X, 'Interpolant', 5X, 'Error')
      NINTV = NDATA - 1
C                                  Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = FLOAT(I-1)/FLOAT(2*NDATA-2)
         WRITE (NOUT,'(2F15.3, F15.6)') X, CSVAL(X,NINTV,BREAK,CSCOEF)
     &                                      , F(X) - CSVAL(X,NINTV,BREAK,
     &                                      CSCOEF)
   20 CONTINUE
      END
```

Output

```
   X         Interpolant     Error
0.000          0.000       0.000000
0.050          0.673       0.008654
0.100          0.997       0.000000
0.150          0.768       0.009879
0.200          0.141       0.000000
0.250         -0.564      -0.007257
0.300         -0.978       0.000000
0.350         -0.848      -0.010906
0.400         -0.279       0.000000
0.450          0.444       0.005714
0.500          0.938       0.000000
0.550          0.911       0.011714
0.600          0.412       0.000000
0.650         -0.315      -0.004057
0.700         -0.880       0.000000
0.750         -0.956      -0.012288
0.800         -0.537       0.000000
0.850          0.180       0.002318
0.900          0.804       0.000000
0.950          0.981       0.012616
1.000          0.650       0.000000
```

CSAKM/DCSAKM (Single/Double precision)

Compute the Akima cubic spline interpolant.

Usage

```
CALL CSAKM (NDATA, XDATA, FDATA, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points.   (Input)

XDATA — Array of length `NDATA` containing the data point abscissas.   (Input)
The data point abscissas must be distinct.

FDATA — Array of length `NDATA` containing the data point ordinates.   (Input)

BREAK — Array of length `NDATA` containing the breakpoints for the piecewise cubic representation.   (Output)

CSCOEF — Matrix of size 4 by `NDATA` containing the local coefficients of the cubic pieces.   (Output)

Comments

1. Automatic workspace usage is

   `CSAKM  NDATA` units, or
   `DCSAKM NDATA` units.

   Workspace may be explicitly provided, if desired, by use of `C2AKMD/C2AKM`. The reference is

   ```
   CALL C2AKM (NDATA, XDATA, FDATA, BREAK, CSCOEF, IWK)
   ```

   The additional argument is

   IWK — Work array of length `NDATA`.

2. The cubic spline can be evaluated using `CSVAL` (page 440); its derivative can be evaluated using `CSDER` (page 441).

3. Note that column `NDATA` of `CSCOEF` is used as workspace.

Algorithm

The routine `CSAKM` computes a C^1 cubic spline interpolant to a set of data points (x_i, f_i) for $i = 1, \ldots,$ `NDATA` $= N$. The breakpoints of the spline are the abscissas. Endpoint conditions are automatically determined by the program; see Akima (1970) or de Boor (1978).

If the data points arise from the values of a smooth (say C^4) function f, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\left\| f - s \right\|_{[\xi_1,\xi_N]} \le C \left\| f^{(2)} \right\|_{[\xi_1,\xi_N]} \left| \xi \right|^2$$

where

$$\left| \xi \right| := \max_{i=2,\ldots,N} \left| \xi_i - \xi_{i-1} \right|$$

The routine CSAKM is based on a method by Akima (1970) to combat wiggles in the interpolant. The method is nonlinear; and although the interpolant is a piecewise cubic, cubic polynomials are not reproduced. (However, linear polynomials are reproduced.)

Example

In this example, a cubic spline interpolant to a function *f* is computed. The values of this spline are then compared with the exact function values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, F,
     &           FDATA(NDATA), FLOAT, SIN, X, XDATA(NDATA)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSAKM, CSVAL, UMACH
C                                  Define function
      F(X) = SIN(15.0*X)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Compute cubic spline interpolant
      CALL CSAKM (NDATA, XDATA, FDATA, BREAK, CSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
      NINTV = NDATA - 1
C                                  Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = FLOAT(I-1)/FLOAT(2*NDATA-2)
         WRITE (NOUT,’(2F15.3,F15.6)’) X, CSVAL(X,NINTV,BREAK,CSCOEF),
     &                                   F(X) - CSVAL(X,NINTV,BREAK,
     &                                   CSCOEF)
   20 CONTINUE
      END
```

Output

```
   X          Interpolant     Error
0.000           0.000       0.000000
0.050           0.818      -0.135988
0.100           0.997       0.000000
0.150           0.615       0.163487
0.200           0.141       0.000000
0.250          -0.478      -0.093376
0.300          -0.978       0.000000
0.350          -0.812      -0.046447
0.400          -0.279       0.000000
0.450           0.386       0.064491
0.500           0.938       0.000000
0.550           0.854       0.068274
0.600           0.412       0.000000
0.650          -0.276      -0.043288
0.700          -0.880       0.000000
0.750          -0.889      -0.078947
0.800          -0.537       0.000000
0.850           0.149       0.033757
0.900           0.804       0.000000
0.950           0.932       0.061260
1.000           0.650       0.000000
```

CSCON/DCSCON (Single/Double precision)

Compute a cubic spline interpolant that is consistent with the concavity of the data.

Usage

```
CALL CSCON (NDATA, XDATA, FDATA, IBREAK, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)
NDATA must be at least 3.

XDATA — Array of length NDATA containing the data point abscissas. (Input)
The data point abscissas must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

IBREAK — The number of breakpoints. (Output)
It will be less than 2 * NDATA.

BREAK — Array of length IBREAK containing the breakpoints for the piecewise cubic representation in its first IBREAK positions. (Output)
The dimension of BREAK must be at least 2 * NDATA.

CSCOEF — Matrix of size 4 by N where N is the dimension of BREAK. (Output)
The first IBREAK – 1 columns of CSCOEF contain the local coefficients of the cubic pieces.

Comments

1. Automatic workspace usage is

   `CSCON` 11 `*` `NDATA` – 14 units, or
   `DCSCON` 21 `*` `NDATA` – 28 units.

   Workspace may be explicitly provided, if desired, by use of `C2CON`/`DC2CON`. The reference is

   ```
   CALL C2CON (NDATA, XDATA, FDATA, IBREAK, BREAK,
               CSCOEF, ITMAX, XSRT, FSRT, A, Y, DIVD,
               ID, WK)
   ```

   The additional arguments are as follows:

   ITMAX — Maximum number of iterations of Newton’s method. (Input)

   XSRT — Work array of length `NDATA` to hold the sorted `XDATA` values.

   FSRT — Work array of length `NDATA` to hold the sorted `FDATA` values.

   A — Work array of length `NDATA`.

   Y — Work array of length `NDATA` – 2.

   DIVD — Work array of length `NDATA` – 2.

   ID — Integer work array of length `NDATA`.

   WK — Work array of length 5 `*` (`NDATA` – 2).

2 Informational errors

Type	Code	
3	16	Maximum number of iterations exceeded, call `C2CON`/`DC2CON` to set a larger number for `ITMAX`.
4	3	The `XDATA` values must be distinct.

3. The cubic spline can be evaluated using `CSVAL` (page 440); its derivative can be evaluated using `CSDER` (page 441).

4. The default value for `ITMAX` is 25. This can be reset by calling `C2CON`/`DC2CON` directly.

Algorithm

The routine `CSCON` computes a cubic spline interpolant to n = `NDATA` data points $\{x_i, f_i\}$ for $i = 1, \ldots, n$. For ease of explanation, we will assume that $x_i < x_{i+1}$, although it is not necessary for the user to sort these data values. If the data are strictly convex, then the computed spline is convex, C^2, and minimizes the expression

$$\int_{x_1}^{x_n} (g'')^2$$

over all convex C^1 functions that interpolate the data. In the general case when the data have both convex and concave regions, the convexity of the spline is consistent with the data and the above integral is minimized under the appropriate constraints. For more information on this interpolation scheme, we refer the reader to Micchelli et al. (1985) and Irvine et al. (1986).

One important feature of the splines produced by this subroutine is that it is not possible, a priori, to predict the number of breakpoints of the resulting interpolant. In most cases, there will be breakpoints at places other than data locations. The method is nonlinear; and although the interpolant is a piecewise cubic, cubic polynomials are not reproduced. (However, linear polynomials are reproduced.) This routine should be used when it is important to preserve the convex and concave regions implied by the data.

Example

We first compute the shape-preserving interpolant using `CSCON`, and display the coefficients and breakpoints. Second, we interpolate the same data using `CSINT` (page 423) in a program not shown and overlay the two results. The graph of the result from `CSINT` is represented by the dashed line. Notice the extra inflection points in the curve produced by `CSINT`.

```
C                                 Specifications
      INTEGER    NDATA
      PARAMETER  (NDATA=9)
C
      INTEGER    IBREAK, NOUT
      REAL       BREAK(2*NDATA), CSCOEF(4,2*NDATA), FDATA(NDATA),
     &           XDATA(NDATA)
      CHARACTER  CLABEL(14)*2, RLABEL(4)*2
      EXTERNAL   CSCON, UMACH, WRRRL
C
      DATA XDATA/0.0, .1, .2, .3, .4, .5, .6, .8, 1./
      DATA FDATA/0.0, .9, .95, .9, .1, .05, .05, .2, 1./
      DATA RLABEL/’ 1’, ’ 2’, ’ 3’, ’ 4’/
      DATA CLABEL/’  ’, ’ 1’, ’ 2’, ’ 3’, ’ 4’, ’ 5’, ’ 6’,
     &     ’ 7’, ’ 8’, ’ 9’, ’10’, ’11’, ’12’, ’13’/
C                                 Compute cubic spline interpolant
      CALL CSCON (NDATA, XDATA, FDATA, IBREAK, BREAK, CSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print the BREAK points and the
C                                 coefficients (CSCOEF) for
C                                 checking purposes.
      WRITE (NOUT,’(1X,A,I2)’) ’IBREAK = ’, IBREAK
      CALL WRRRL (’BREAK’, 1, IBREAK, BREAK, 1, 0, ’(F9.3)’, RLABEL,
     &            CLABEL)
      CALL WRRRL (’CSCOEF’, 4, IBREAK, CSCOEF, 4, 0, ’(F9.3)’, RLABEL,
     &            CLABEL)
      END
```

Output

```
IBREAK = 13
                               BREAK
           1          2          3          4          5          6
1      0.000      0.100      0.136      0.200      0.259      0.300

           7          8          9         10         11         12
1      0.400      0.436      0.500      0.600      0.609      0.800

          13
1      1.000

                              CSCOEF
           1          2          3          4          5          6
1      0.000      0.900      0.942      0.950      0.958      0.900
2     11.886      3.228      0.131      0.131      0.131    -4.434
3      0.000   -173.170      0.000      0.000      0.000   220.218
4  -1731.699   4841.604      0.000      0.000  -5312.082  4466.875

           7          8          9         10         11         12
1      0.100      0.050      0.050      0.050      0.050      0.200
2     -4.121      0.000      0.000      0.000      0.000      2.356
3    226.470      0.000      0.000      0.000      0.000    24.664
4  -6222.348      0.000      0.000      0.000    129.115   123.321

          13
1      1.000
2      0.000
3      0.000
4      0.000
```

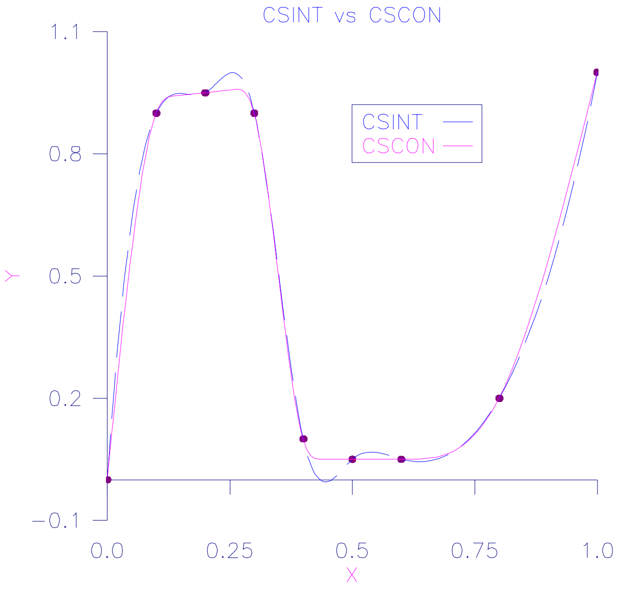


Figure 3-4   CSCON vs. CSINT

CSPER/DCSPER (Single/Double precision)

Compute the cubic spline interpolant with periodic boundary conditions.

Usage

```
CALL CSPER (NDATA, XDATA, FDATA, BREAK, CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)
NDATA must be at least 4.

XDATA — Array of length NDATA containing the data point abscissas. (Input)
The data point abscissas must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

BREAK — Array of length NDATA containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by NDATA containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   CSPER 7 * NDATA units, or
   DCSPER 13 * NDATA units.

   Workspace may be explicitly provided, if desired, by use of C2PER/DC2PER. The reference is

   ```
   CALL C2PER (NDATA, XDATA, FDATA, BREAK, CSCOEF, WK,
               IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 6 * NDATA.

   IWK — Work array of length NDATA.

2. Informational error

Type	Code	
3	1	The data set is not periodic, i.e., the function values at the smallest and largest XDATA points are not equal. The value at the smallest XDATA point is used.

3. The cubic spline can be evaluated using CSVAL (page 440)and its derivative can be evaluated using CSDER (page 441).

Algorithm

The routine CSPER computes a C^2 cubic spline interpolant to a set of data points (x_i, f_i) for $i = 1, \ldots,$ NDATA $= N$. The breakpoints of the spline are the abscissas. The program enforces periodic endpoint conditions. This means that the spline s satisfies $s(a) = s(b)$, $s'(a) = s'(b)$, and $s''(a) = s''(b)$, where a is the leftmost abscissa and b is the rightmost abscissa. If the ordinate values corresponding to a and b are not equal, then a warning message is issued. The ordinate value at b is set equal to the ordinate value at a and the interpolant is computed.

If the data points arise from the values of a smooth (say C^4) periodic function f, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. Let ξ be the breakpoint vector for the above spline interpolant. Then, the maximum absolute error satisfies

$$\|f - s\|_{[\xi_1,\xi_N]} \le C\left\|f^{(4)}\right\|_{[\xi_1,\xi_N]}|\xi|^4$$

where

$$|\xi| := \max_{i=2,\ldots,N}|\xi_i - \xi_{i-1}|$$

For more details, see de Boor (1978, pages 320−322).

Example

In this example, a cubic spline interpolant to a function f is computed. The values of this spline are then compared with the exact function values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=11)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, F,
     &           FDATA(NDATA), FLOAT, H, PI, SIN, X, XDATA(NDATA)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSPER, CSVAL, UMACH
C
C                                 Define function
      F(X) = SIN(15.0*X)
C                                 Set up a grid
      PI = CONST('PI')
      H = 2.0*PI/15.0/10.0
      DO 10  I=1, NDATA
         XDATA(I) = H*FLOAT(I-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Round off will cause FDATA(11) to
C                                 be nonzero; this would produce a
C                                 warning error since FDATA(1) is zero.
C                                 Therefore, the value of FDATA(1) is
C                                 used rather than the value of
C                                 FDATA(11).
      FDATA(NDATA) = FDATA(1)
```

```
C
C                                  Compute cubic spline interpolant
      CALL CSPER (NDATA, XDATA, FDATA, BREAK, CSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
      NINTV = NDATA - 1
      H     = H/2.0
C                                  Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA - 1
         X = H*FLOAT(I-1)
         WRITE (NOUT,’(2F15.3,F15.6)’) X, CSVAL(X,NINTV,BREAK,CSCOEF),
     &                                 F(X) - CSVAL(X,NINTV,BREAK,
     &                                 CSCOEF)
   20 CONTINUE
      END
```

Output

```
    X         Interpolant     Error
0.000          0.000       0.000000
0.021          0.309       0.000138
0.042          0.588       0.000000
0.063          0.809       0.000362
0.084          0.951       0.000000
0.105          1.000       0.000447
0.126          0.951       0.000000
0.147          0.809       0.000362
0.168          0.588       0.000000
0.188          0.309       0.000138
0.209          0.000       0.000000
0.230         -0.309      -0.000138
0.251         -0.588       0.000000
0.272         -0.809      -0.000362
0.293         -0.951       0.000000
0.314         -1.000      -0.000447
0.335         -0.951       0.000000
0.356         -0.809      -0.000362
0.377         -0.588       0.000000
0.398         -0.309      -0.000138
0.419          0.000       0.000000
```

CSVAL/DCSVAL (Single/Double precision)

Evaluate a cubic spline.

Usage

`CSVAL(X, NINTV, BREAK, CSCOEF)`

Arguments

X — Point at which the spline is to be evaluated.   (Input)

NINTV — Number of polynomial pieces.   (Input)

BREAK — Array of length `NINTV` + 1 containing the breakpoints for the piecewise cubic representation. (Input)
`BREAK` must be strictly increasing.

CSCOEF — Matrix of size 4 by `NINTV` + 1 containing the local coefficients of the cubic pieces. (Input)

CSVAL — Value of the polynomial at `X`. (Output)

Algorithm

The routine `CSVAL` evaluates a cubic spline at a given point. It is a special case of the routine `PPDER` (page 507), which evaluates the derivative of a piecewise polynomial. (The value of a piecewise polynomial is its zero-th derivative and a cubic spline is a piecewise polynomial of order 4.) The routine `PPDER` is based on the routine `PPVALU` in de Boor (1978, page 89).

Example

For an example of the use of `CSVAL`, see IMSL routine `CSINT` (page 423).

CSDER/DCSDER (Single/Double precision)

Evaluate the derivative of a cubic spline.

Usage

```
CSDER(IDERIV, X, NINTV, BREAK, CSCOEF)
```

Arguments

IDERIV — Order of the derivative to be evaluated. (Input)
In particular, `IDERIV` = 0 returns the value of the polynomial.

X — Point at which the polynomial is to be evaluated. (Input)

NINTV — Number of polynomial pieces. (Input)

BREAK — Array of length `NINTV` + 1 containing the breakpoints for the piecewise cubic representation. (Input)
`BREAK` must be strictly increasing.

CSCOEF — Matrix of size 4 by `NINTV` + 1 containing the local coefficients of the cubic pieces. (Input)

CSDER — Value of the `IDERIV`-th derivative of the polynomial at `X`. (Output)

Algorithm

The function `CSDER` evaluates the derivative of a cubic spline at a given point. It is a special case of the routine `PPDER` (page 507), which evaluates the derivative of a piecewise polynomial. (A cubic spline is a piecewise polynomial of order 4.) The routine PPDER is based on the routine PPVALU in de Boor (1978, page 89).

Example

In this example, we compute a cubic spline interpolant to a function *f* using IMSL routine CSINT (page 423). The values of the spline and its first and second derivatives are computed using CSDER. These values can then be compared with the corresponding values of the interpolated function.

```
       INTEGER    NDATA
      PARAMETER  (NDATA=10)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CDDF, CDF, CF, COS, CSCOEF(4,NDATA),
     &           CSDER, DDF, DF, F, FDATA(NDATA), FLOAT, SIN, X,
     &           XDATA(NDATA)
      INTRINSIC  COS, FLOAT, SIN
      EXTERNAL   CSDER, CSINT, UMACH
C                                 Define function and derivatives
      F(X)   = SIN(15.0*X)
      DF(X)  = 15.0*COS(15.0*X)
      DDF(X) = -225.0*SIN(15.0*X)
C                                 Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Compute cubic spline interpolant
      CALL CSINT (NDATA, XDATA, FDATA, BREAK, CSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
99999 FORMAT (9X, ’X’, 8X, ’S(X)’, 5X, ’Error’, 6X, ’S’’(X)’, 5X,
     &       ’Error’, 6X, ’S’’’’(X)’, 4X, ’Error’, /)
      NINTV = NDATA - 1
C                                 Print the interpolant on a finer grid
      DO 20  I=1, 2*NDATA
         X    = FLOAT(I-1)/FLOAT(2*NDATA-1)
         CF   = CSDER(0,X,NINTV,BREAK,CSCOEF)
         CDF  = CSDER(1,X,NINTV,BREAK,CSCOEF)
         CDDF = CSDER(2,X,NINTV,BREAK,CSCOEF)
         WRITE (NOUT,’(F11.3, 3(F11.3, F11.6))’) X, CF, F(X) - CF,
     &                                             CDF, DF(X) - CDF,
     &                                             CDDF, DDF(X) - CDDF
   20 CONTINUE
      END
```

Output

```
   X        S(X)     Error      S’(X)     Error      S’’(X)    Error

0.000      0.000   0.000000     26.285 -11.284739   -379.458 379.457794
0.053      0.902  -0.192203      8.841   1.722460   -283.411 123.664734
0.105      1.019  -0.019333     -3.548   3.425718   -187.364 -37.628586
0.158      0.617   0.081009    -10.882   0.146207    -91.317 -65.824875
0.211     -0.037   0.021155    -13.160  -1.837700      4.730  -1.062027
```

```
0.263     -0.674  -0.046945    -10.033  -0.355268    117.916  44.391640
0.316     -0.985  -0.015060     -0.719   1.086203    235.999 -11.066727
0.368     -0.682  -0.004651     11.314  -0.409097    154.861  -0.365387
0.421      0.045  -0.011915     14.708   0.284042    -25.887  18.552732
0.474      0.708   0.024292      9.508   0.702690   -143.785 -21.041260
0.526      0.978   0.020854      0.161  -0.771948   -211.402 -13.411087
0.579      0.673   0.001410    -11.394   0.322443   -163.483  11.674103
0.632     -0.064   0.015118    -14.937  -0.045511     28.856 -17.856323
0.684     -0.724  -0.019246     -8.859  -1.170871    163.866   3.435547
0.737     -0.954  -0.044143      0.301   0.554493    184.217  40.417282
0.789     -0.675   0.012143     10.307   0.928152    166.021 -16.939514
0.842      0.027   0.038176     15.015  -0.047344     12.914 -27.575521
0.895      0.764  -0.010112     11.666  -1.819128   -140.193 -29.538193
0.947      1.114  -0.116304      0.258  -1.357680   -293.301  68.905701
1.000      0.650   0.000000    -19.208   7.812407   -446.408 300.092896
```

CS1GD/DCS1GD (Single/Double precision)

Evaluate the derivative of a cubic spline on a grid.

Usage

```
CALL CS1GD (IDERIV, N, XVEC, NINTV, BREAK, CSCOEF, VALUE)
```

Arguments

IDERIV — Order of the derivative to be evaluated.   (Input)
In particular, `IDERIV` = 0 returns the values of the cubic spline.

N — Length of vector `XVEC`.   (Input)

XVEC — Array of length `N` containing the points at which the cubic spline is to be evaluated.   (Input)
The points in `XVEC` should be strictly increasing.

NINTV — Number of polynomial pieces.   (Input)

BREAK — Array of length `NINTV` + 1 containing the breakpoints for the piecewise cubic representation.   (Input)
`BREAK` must be strictly increasing.

CSCOEF — Matrix of size 4 by `NINTV` + 1 containing the local coefficients of the cubic pieces.   (Input)

VALUE — Array of length `N` containing the values of the `IDERIV`-th derivative of the cubic spline at the points in `XVEC`.   (Output)

Comments

1. Automatic workspace usage is

   `CS1GD` 3*N* units, or
   `DCS1GD` 5*N* units.

Workspace may be explicitly provided, if desired, by use of C21GD/DC21GD. The reference is

```
CALL C21GD (IDERIV, N, XVEC, NINTV, BREAK, CSCOEF,
            VALUE, IWK, WORK1, WORK2)
```

The additional arguments are as follows:

IWK — Array of length N.

WORK1 — Array of length N.

WORK2 — Array of length N.

2. Informational error

Type	Code	
4	4	The points in XVEC must be strictly increasing.

Algorithm

The routine CS1GD evaluates a cubic spline (or its derivative) at a vector of points. That is, given a vector x of length n satisfying $x_i < x_{i+1}$ for $i = 1, \ldots, n - 1$, a derivative value j, and a cubic spline s that is represented by a breakpoint sequence and coefficient matrix this routine returns the values

$$s^{(j)}(x_i) \quad i = 1, \ldots, n$$

in the array VALUE. The functionality of this routine is the same as that of CSDER (page 441) called in a loop, however CS1GD should be much more efficient.

Example

To illustrate the use of CS1GD, we modify the example program for CSINT (page 423). In this example, a cubic spline interpolant to F is computed. The values of this spline are then compared with the exact function values. The routine CS1GD is based on the routine PPVALU in de Boor (1978, page 89).

```
C                                  Specifications
      INTEGER    NDATA, N
      PARAMETER  (NDATA=11, N=2*NDATA-1)
C
      INTEGER    I, NINTV, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, F,
     &           FDATA(NDATA), FLOAT, SIN, X, XDATA(NDATA),
     &           FVALUE(N), VALUE(N), XVEC(N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CSINT, CSVAL, UMACH
C                                  Define function
      F(X) = SIN(15.0*X)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Compute cubic spline interpolant
```

```
      CALL CSINT (NDATA, XDATA, FDATA, BREAK, CSCOEF)
      DO 20  I=1, N
         XVEC(I) = FLOAT(I-1)/FLOAT(2*NDATA-2)
         FVALUE(I) = F(XVEC(I))
   20  CONTINUE
      IDERIV = 0
      NINTV = NDATA - 1
      CALL CS1GD (IDERIV, N, XVEC, NINTV, BREAK, CSCOEF, VALUE)
C                                 Get output unit number.
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
99999 FORMAT (13X, ’X’, 9X, ’Interpolant’, 5X, ’Error’)
C                                 Print the interpolant and the error
C                                 on a finer grid
      DO 30 J=1, N
         WRITE (NOUT,’(2F15.3,F15.6)’) XVEC(J), VALUE(J),
     &                                    FVALUE(J)-VALUE(J)
   30 CONTINUE
      END
```

Output

```
   X         Interpolant     Error
0.000          0.000       0.000000
0.050          0.809      -0.127025
0.100          0.997       0.000000
0.150          0.723       0.055214
0.200          0.141       0.000000
0.250         -0.549      -0.022789
0.300         -0.978       0.000000
0.350         -0.843      -0.016246
0.400         -0.279       0.000000
0.450          0.441       0.009348
0.500          0.938       0.000000
0.550          0.903       0.019947
0.600          0.412       0.000000
0.650         -0.315      -0.004895
0.700         -0.880       0.000000
0.750         -0.938      -0.029541
0.800         -0.537       0.000000
0.850          0.148       0.034693
0.900          0.804       0.000000
0.950          1.086      -0.092559
1.000          0.650       0.000000
```

CSITG/DCSITG (Single/Double precision)

Evaluate the integral of a cubic spline.

Usage

```
CSITG(A, B, NINTV, BREAK, CSCOEF)
```

Arguments

A — Lower limit of integration.   (Input)

B — Upper limit of integration.   (Input)

NINTV — Number of polynomial pieces.   (Input)

BREAK — Array of length `NINTV` + 1 containing the breakpoints for the piecewise cubic representation.   (Input)
`BREAK` must be strictly increasing.

CSCOEF — Matrix of size 4 by `NINTV` + 1 containing the local coefficients of the cubic pieces.   (Input)

CSITG — Value of the integral of the spline from `A` to `B`.   (Output)

Algorithm

The function `CSITG` evaluates the integral of a cubic spline over an interval. It is a special case of the routine `PPITG` (page 512), which evaluates the integral of a piecewise polynomial. (A cubic spline is a piecewise polynomial of order 4.)

Example

This example computes a cubic spline interpolant to the function x^2 using `CSINT` (page 423) and evaluates its integral over the intervals [0., .5] and [0., 2.]. Since `CSINT` uses the not-a knot condition, the interpolant reproduces x^2, hence the integral values are 1/24 and 8/3, respectively.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=10)
C
      INTEGER    I, NINTV, NOUT
      REAL       A, B, BREAK(NDATA), CSCOEF(4,NDATA), CSITG, ERROR,
     &           EXACT, F, FDATA(NDATA), FI, FLOAT, VALUE, X,
     &           XDATA(NDATA)
      INTRINSIC  FLOAT
      EXTERNAL   CSINT, CSITG, UMACH
C                                 Define function and integral
      F(X)  = X*X
      FI(X) = X*X*X/3.0
C                                 Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Compute cubic spline interpolant
      CALL CSINT (NDATA, XDATA, FDATA, BREAK, CSCOEF)
C                                 Compute the integral of F over
C                                 [0.0,0.5]
      A     = 0.0
      B     = 0.5
      NINTV = NDATA - 1
      VALUE = CSITG(A,B,NINTV,BREAK,CSCOEF)
      EXACT = FI(B) - FI(A)
      ERROR = EXACT - VALUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print the result
```

```
      WRITE (NOUT,99999) A, B, VALUE, EXACT, ERROR
C                                 Compute the integral of F over
C                                 [0.0,2.0]
      A     = 0.0
      B     = 2.0
      VALUE = CSITG(A,B,NINTV,BREAK,CSCOEF)
      EXACT = FI(B) - FI(A)
      ERROR = EXACT - VALUE
C                                 Print the result
      WRITE (NOUT,99999) A, B, VALUE, EXACT, ERROR
99999 FORMAT (' On the closed interval (', F3.1, ',', F3.1,
     &       ') we have :', /, 1X, 'Computed Integral = ', F10.5, /,
     &       1X, 'Exact Integral    = ', F10.5, /, 1X, 'Error         '
     &       , '    = ', F10.6, /, /)
      END
```

Output

```
On the closed interval (0.0,0.5) we have :
Computed Integral =    0.04167
Exact Integral    =    0.04167
Error             =   0.000000

On the closed interval (0.0,2.0) we have :
Computed Integral =    2.66666
Exact Integral    =    2.66667
Error             =   0.000006
```

SPLEZ/DSPLEZ (Single/Double precision)

Compute the values of a spline that either interpolates or fits user-supplied data.

Usage

```
CALL SPLEZ (NDATA, XDATA, FDATA, ITYPE, IDER, N, XVEC,
            VALUE)
```

Arguments

NDATA — Number of data points. (Input)
All choices of ITYPE are valid if NDATA is larger than 6. More specifically,

NDATA > ITYPE	or ITYPE = 1.
NDATA > 3	for ITYPE = 2, 3.
NDATA > (ITYPE – 3)	for ITYPE = 4, 5, 6, 7, 8.
NDATA > 3	for ITYPE = 9, 10, 11, 12.
NDATA > KORDER	for ITYPE = 13, 14, 15.

XDATA — Array of length NDATA containing the data point abscissae. (Input)
The data point abscissas must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

ITYPE — Type of interpolant desired. (Input)

ITYPE

1	yields CSINT
2	yields CSAKM
3	yields CSCON
4	yields BSINT-BSNAK K = 2
5	yields BSINT-BSNAK K = 3
6	yields BSINT-BSNAK K = 4
7	yields BSINT-BSNAK K = 5
8	yields BSINT-BSNAK K = 6
9	yields CSSCV
10	yields BSLSQ K = 2
11	yields BSLSQ K = 3
12	yields BSLSQ K = 4
13	yields BSVLS K = 2
14	yields BSVLS K = 3
15	yields BSVLS K = 4

IDER — Order of the derivative desired. (Input)

N — Number of function values desired. (Input)

XVEC — Array of length N containing the points at which the spline function values are desired. (Input)
The entries of XVEC must be distinct.

VALUE — Array of length N containing the spline values. (Output)
VALUE (I) = S(XVEC (I)) if IDER = 0, VALUE(I) = S′(XVEC (I)) if IDER = 1, and so forth, where S is the computed spline.

Comments

1. Automatic workspace usage is

   SPLEZ 32 * NDATA + 5 * N + 22 + MAX0(NDATA, N) units, or
   DSPLEZ 64 * NDATA + 9 * N + 44 + MAX0(NDATA, N) units.

   Workspace may be explicitly provided, if desired, by use of S2LEZ/DS2LEZ. The reference is

   ```
   CALL S2LEZ (NDATA, XDATA, FDATA, ITYPE, IDER, N,
               XVEC, VALUE, WRK, IWK)
   ```

   The additional arguments are as follows:

   WRK — Work array of length 32 * NDATA + 4 * N + 22.

   IWK — Work array of length MAX0(NDATA, N) + N.

2. Informational errors

Type	Code	
4	1	XDATA entries are not unique.
4	2	XVEC entries are not unique.

3. The workspace listed above is the maximum that is needed. Depending on the choice of ITYPE, the actual amount used may be less. If workspace is a critical resource, consult the explicit routines listed under ITYPE to see if less workspace can be used.

Algorithm

This routine is designed to let the user experiment with various interpolation and smoothing routines in the library.

The routine SPLEZ computes a spline interpolant to a set of data points (x_i, f_i) for i = 1, …, NDATA if ITYPE = 1, …, 8. If ITYPE ≥ 9, various smoothing or least squares splines are computed. The output for this routine consists of a vector of values of the computed spline or its derivatives. Specifically, let i = IDER, n = N, v = XVEC, and y = VALUE, then if s is the computed spline we set

$$y_j = s^{(i)}(v_j) \qquad j = 1, \ldots, n$$

The routines called are listed above under the ITYPE heading. Additional documentation can be found by referring to these routines.

Example

In this example, all the ITYPE parameters are exercised. The values of the spline are then compared with the exact function values and derivatives.

```
      INTEGER    NDATA, N
      PARAMETER  (NDATA=21, N=2*NDATA-1)
C                                  Specifications for local variables
      INTEGER    I, IDER, ITYPE, NOUT
      REAL       FDATA(NDATA), FPVAL(N), FVALUE(N),
     &           VALUE(N), XDATA(NDATA), XVEC(N), EMAX1(15),
     &           EMAX2(15)
C                                  Specifications for intrinsics
      INTRINSIC  FLOAT, SIN, COS
      REAL       FLOAT, SIN, COS
C                                  Specifications for subroutines
      EXTERNAL   UMACH, SPLEZ, SAXPY
C
      REAL       F, FP
C
C                                  Define a function
      F(X)  = SIN(X*X)
      FP(X) = 2*X*COS(X*X)
C
      CALL UMACH (2, NOUT)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = 3.0*(FLOAT(I-1)/FLOAT(NDATA-1))
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
      DO 20  I=1, N
         XVEC(I)    = 3.0*(FLOAT(I-1)/FLOAT(2*NDATA-2))
         FVALUE(I)  = F(XVEC(I))
         FPVAL(I) = FP(XVEC(I))
   20 CONTINUE
C
      WRITE (NOUT,99999)
C                                 Loop to call SPLEZ for each ITYPE
      DO 40  ITYPE=1, 15
         DO 30  IDER=0, 1
            CALL SPLEZ (NDATA, XDATA, FDATA, ITYPE, IDER, N,
     &                      XVEC, VALUE)
C                                Compute the maximum error
            IF (IDER .EQ. 0) THEN
               CALL SAXPY (N, -1.0, FVALUE, 1, VALUE, 1)
               EMAX1(ITYPE) = ABS(VALUE(ISAMAX(N,VALUE,1)))
            ELSE
               CALL SAXPY (N, -1.0, FPVAL, 1, VALUE, 1)
               EMAX2(ITYPE) = ABS(VALUE(ISAMAX(N,VALUE,1)))
            END IF
   30  CONTINUE
         WRITE (NOUT,'(I7,2F20.6)') ITYPE, EMAX1(ITYPE), EMAX2(ITYPE)
   40 CONTINUE
C
99999 FORMAT (4X, 'ITYPE', 6X, 'Max error for f', 5X,
     &       'Max error for f''', /)
      END
```

Output

```
ITYPE      Max error for f     Max error for f'

 1           0.014082            0.658018
 2           0.024682            0.897757
 3           0.020896            0.813228
 4           0.083615            2.168083
 5           0.010403            0.508043
 6           0.014082            0.658020
 7           0.004756            0.228858
 8           0.001070            0.077159
 9           0.020896            0.813228
10           0.392603            6.047916
11           0.162793            1.983959
12           0.045404            1.582624
13           0.588370            7.680381
14           0.752475            9.673786
15           0.049340            1.713031
```

BSINT/DBSINT (Single/Double precision)

Compute the spline interpolant, returning the B-spline coefficients.

Usage

`CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)`

Arguments

NDATA — Number of data points. (Input)

XDATA — Array of length NDATA containing the data point abscissas. (Input)

FDATA — Array of length NDATA containing the data point ordinates. (Input)

KORDER — Order of the spline. (Input)
KORDER must be less than or equal to NDATA.

XKNOT — Array of length NDATA + KORDER containing the knot sequence. (Input)
XKNOT must be nondecreasing.

BSCOEF — Array of length NDATA containing the B-spline coefficients. (Output)

Comments

1. Automatic workspace usage is

   BSINT (5 * KORDER + 1) * NDATA units, or
   DBSINT (10 * KORDER + 1) * NDATA units.

   Workspace may be explicitly provided, if desired, by use of B2INT/DB2INT. The reference is

   ```
   CALL B2INT (NDATA, XDATA, FDATA, KORDER, XKNOT,
               BSCOEF, WK1, WK2, WK3, IWK)
   ```

   The additional arguments are as follows:

   WK1 — Work array of length (5 * KORDER – 2) * NDATA.

   WK2 — Work array of length NDATA.

   WK3 — Work array of length NDATA.

   IWK — Work array of length NDATA.

2. Informational errors

Type	Code	
3	1	The interpolation matrix is ill-conditioned.
4	3	The XDATA values must be distinct.
4	4	Multiplicity of the knots cannot exceed the order of the spline.
4	5	The knots must be nondecreasing.
4	15	The I-th smallest element of the data point array must be greater than the Ith knot and less than the (I + KORDER)-th knot.
4	16	The largest element of the data point array must be greater than the (NDATA)-th knot and less than or equal to the (NDATA + KORDER)-th knot.
4	17	The smallest element of the data point array must be greater than or equal to the first knot and less than the (KORDER + 1)st knot.

3. The spline can be evaluated using BSVAL (page 469), and its derivative can be evaluated using BSDER (page 471).

Algorithm

Following the notation in de Boor (1978, page 108), let $B_j = B_{j,k,t}$ denote the j-th B-spline of order k with respect to the knot sequence **t**. Then, BSINT computes the vector a satisfying

$$\sum_{j=1}^{N} a_j B_j(x_i) = f_i$$

and returns the result in BSCOEF = a. This linear system is banded with at most $k - 1$ subdiagonals and $k - 1$ superdiagonals. The matrix

$$A = (B_j(x_i))$$

is totally positive and is invertible if and only if the diagonal entries are nonzero. The routine BSINT is based on the routine SPLINT by de Boor (1978, page 204).

The routine BSINT produces the coefficients of the B-spline interpolant of order KORDER with knot sequence XKNOT to the data (x_i, f_i) for $i = 1$ to NDATA, where x = XDATA and f = FDATA. Let $\mathbf{t}$ = XKNOT, k = KORDER, and N = NDATA. First, BSINT sorts the XDATA vector and stores the result in x. The elements of the FDATA vector are permuted appropriately and stored in f, yielding the equivalent data (x_i, f_i) for $i = 1$ to N. The following preliminary checks are performed on the data. We verify that

$$\begin{aligned} x_i &< x_{i+1} & i &= 1, \ldots, N-1 \\ \mathbf{t}_i &< \mathbf{t}_{i+1} & i &= 1, \ldots, N \\ \mathbf{t}_i &\le \mathbf{t}_{i+k} & i &= 1, \ldots, N+k-1 \end{aligned}$$

The first test checks to see that the abscissas are distinct. The second and third inequalities verify that a valid knot sequence has been specified.

In order for the interpolation matrix to be nonsingular, we also check $\mathbf{t}_k \le x_i \le \mathbf{t}_{N+1}$ for $i = 1$ to N. This first inequality in the last check is necessary since the method used to generate the entries of the interpolation matrix requires that the k possibly nonzero B-splines at x_i,

$$B_{j-k+1}, \ldots, B_j \qquad \text{where } j \text{ satisfies } \mathbf{t}_j \le x_i < \mathbf{t}_{j+1}$$

be well-defined (that is, $j - k + 1 \ge 1$).

General conditions are not known for the exact behavior of the error in spline interpolation, however, if **t** and x are selected properly and the data points arise from the values of a smooth (say C^k) function f, i.e. $f_i = f(x_i)$, then the error will behave in a predictable fashion. The maximum absolute error satisfies

$$\left\| f - s \right\|_{[\mathbf{t}_k, \mathbf{t}_{N+1}]} \le C \left\| f^{(k)} \right\|_{[\mathbf{t}_k, \mathbf{t}_{N+1}]} \left| \mathbf{t} \right|^k$$

where

$$\left| \mathbf{t} \right| := \max_{i=k,\ldots,N} \left| \mathbf{t}_{i+1} - \mathbf{t}_i \right|$$

For more information on this problem, see de Boor (1978, Chapter 13) and the references therein. This routine can be used in place of the IMSL routine CSINT (page 423) by calling BSNAK (page 454) to obtain the proper knots, then calling BSINT yielding the B-spline coefficients, and finally calling IMSL routine BSCPP (page 504) to convert to piecewise polynomial form.

Example

In this example, a spline interpolant *s*, to

$$f(x) = \sqrt{x}$$

is computed. The interpolated values are then compared with the exact function values using the IMSL routine BSVAL (page 469).

```
      INTEGER    KORDER, NDATA, NKNOT
      PARAMETER  (KORDER=3, NDATA=5, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NCOEF, NOUT
      REAL       BSCOEF(NDATA), BSVAL, BT, F, FDATA(NDATA), FLOAT,
     &           SQRT, X, XDATA(NDATA), XKNOT(NKNOT), XT
      INTRINSIC  FLOAT, SQRT
      EXTERNAL   BSINT, BSNAK, BSVAL, UMACH
C                                 Define function
      F(X) = SQRT(X)
C                                 Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Interpolate
      CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print on a finer grid
      NCOEF = NDATA
      XT    = XDATA(1)
C                                 Evaluate spline
      BT    = BSVAL(XT,KORDER,XKNOT,NCOEF,BSCOEF)
      WRITE (NOUT,99998) XT, BT, F(XT) - BT
      DO 20  I=2, NDATA
         XT = (XDATA(I-1)+XDATA(I))/2.0
C                                 Evaluate spline
         BT = BSVAL(XT,KORDER,XKNOT,NCOEF,BSCOEF)
         WRITE (NOUT,99998) XT, BT, F(XT) - BT
         XT = XDATA(I)
C                                   Evaluate spline
         BT = BSVAL(XT,KORDER,XKNOT,NCOEF,BSCOEF)
         WRITE (NOUT,99998) XT, BT, F(XT) - BT
   20 CONTINUE
99998 FORMAT (' ', F6.4, 15X, F8.4, 12X, F11.6)
99999 FORMAT (/, 6X, 'X', 19X, 'S(X)', 18X, 'Error', /)
      END
```

Output

```
     X                   S(X)                Error
0.0000                 0.0000             0.000000
0.1250                 0.2918             0.061781
0.2500                 0.5000             0.000000
0.3750                 0.6247            -0.012311
0.5000                 0.7071             0.000000
0.6250                 0.7886             0.002013
0.7500                 0.8660             0.000000
0.8750                 0.9365            -0.001092
1.0000                 1.0000             0.000000
```

BSNAK/DBSNAK (Single/Double precision)

Compute the “not-a-knot” spline knot sequence.

Usage

CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)

Arguments

NDATA — Number of data points.   (Input)

XDATA — Array of length NDATA containing the location of the data points. (Input)

KORDER — Order of the spline.   (Input)

XKNOT — Array of length NDATA + KORDER containing the knot sequence. (Output)

Comments

1. Automatic workspace usage is

   BSNAK   2 * NDATA units, or
   DBSNAK 3 * NDATA units.

   Workspace may be explicitly provided, if desired, by use of B2NAK/DB2NAK. The reference is

   CALL B2NAK (NDATA, XDATA, KORDER, XKNOT, XSRT, IWK)

   The additional arguments are as follows:

XSRT — Work array of length `NDATA` to hold the sorted `XDATA` values. If `XDATA` is not needed, `XSRT` may be the same as `XDATA`.

IWK — Work array of length `NDATA` to hold the permutation of `XDATA`.

2. Informational error

Type	Code	
4	4	The `XDATA` values must be distinct.

3. The first knot is at the left endpoint and the last knot is slightly beyond the last endpoint. Both endpoints have multiplicity `KORDER`.

4. Interior knots have multiplicity one.

Algorithm

Given the data points x = `XDATA` , the order of the spline k = `KORDER`, and the number N = `NDATA` of elements in `XDATA`, the subroutine `BSNAK` returns in $\mathbf{t}$ = `XKNOT` a knot sequence that is appropriate for interpolation of data on x by splines of order k. The vector $\mathbf{t}$ contains the knot sequence in its first $N + k$ positions. If k is even and we assume that the entries in the input vector x are increasing, then $\mathbf{t}$ is returned as

$$\mathbf{t}_i = x_1 \qquad \text{for } i = 1, \ldots, k$$

$$\mathbf{t}_i = x_{i\text{ - }k/2} \qquad \text{for } i = k + 1, \ldots, N$$

$$\mathbf{t}_i = x_N + \varepsilon \qquad \text{for } i = N + 1, \ldots, N + k$$

where ε is a small positive constant. There is some discussion concerning this selection of knots in de Boor (1978, page 211). If k is odd, then $\mathbf{t}$ is returned as

$$\mathbf{t}_i = x_1 \qquad \text{for } i = 1, \ldots, k$$

$$\mathbf{t}_i = \left(x_{i-\frac{k-1}{2}} + x_{i-1-\frac{k-1}{2}}\right)/2 \quad \text{for } i = k + 1, \ldots, N$$

$$\mathbf{t}_i = x_N + \varepsilon \qquad \text{for } i = N + 1, \ldots, N + k$$

It is not necessary to sort the values in x since this is done in the routine `BSNAK`.

Example

In this example, we compute (for $k = 3, \ldots, 8$) six spline interpolants s_k to $F(x) = \sin(10x^3)$ on the interval [0,1]. The routine `BSNAK` is used to generate the knot sequences for s_k and then `BSINT` (page 450) is called to obtain the interpolant. We evaluate the absolute error

$$|s_k - F|$$

at 100 equally spaced points and print the maximum error for each k.

```
      INTEGER    KMAX, KMIN, NDATA
      PARAMETER  (KMAX=8, KMIN=3, NDATA=20)
C
```

```
      INTEGER    I, K, KORDER, NOUT
      REAL       ABS, AMAX1, BSCOEF(NDATA), BSVAL, DIF, DIFMAX, F,
     &           FDATA(NDATA), FLOAT, FT, SIN, ST, T, X, XDATA(NDATA),
     &           XKNOT(KMAX+NDATA), XT
      INTRINSIC  ABS, AMAX1, FLOAT, SIN
      EXTERNAL   BSINT, BSNAK, BSVAL, UMACH
C                                 Define function and tau function
      F(X) = SIN(10.0*X*X*X)
      T(X) = 1.0 - X*X
C                                 Set up data
      DO 10  I=1, NDATA
         XT       = FLOAT(I-1)/FLOAT(NDATA-1)
         XDATA(I) = T(XT)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Loop over different orders
      DO 30  K=KMIN, KMAX
         KORDER = K
C                                 Generate knots
         CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Interpolate
         CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
         DIFMAX = 0.0
         DO 20  I=1, 100
            XT     = FLOAT(I-1)/99.0
C                                 Evaluate spline
            ST     = BSVAL(XT,KORDER,XKNOT,NDATA,BSCOEF)
            FT     = F(XT)
            DIF    = ABS(FT-ST)
C                                 Compute maximum difference
            DIFMAX = AMAX1(DIF,DIFMAX)
   20  CONTINUE
C                                 Print maximum difference
         WRITE (NOUT,99998) KORDER, DIFMAX
   30 CONTINUE
C
99998 FORMAT (’ ’, I3, 5X, F9.4)
99999 FORMAT (’ KORDER’, 5X, ’Maximum difference’, /)
      END
```

Output

```
KORDER     Maximum difference
   3        0.0080
   4        0.0026
   5        0.0004
   6        0.0008
   7        0.0010
   8        0.0004
```

BSOPK/DBSOPK (Single/Double precision)

Compute the “optimal” spline knot sequence.

Usage

```
CALL BSOPK (NDATA, XDATA, KORDER, XKNOT)
```

Arguments

NDATA — Number of data points.   (Input)

XDATA — Array of length NDATA containing the location of the data points. (Input)

KORDER — Order of the spline.   (Input)

XKNOT — Array of length NDATA + KORDER containing the knot sequence. (Output)

Comments

1. Automatic workspace usage is

   BSOPK (NDATA – KORDER) * (3 * KORDER – 2) + 7 * NDATA + 2 * KORDER + 5 units, or

   DBSOPK 2 * (NDATA – KORDER) * (3 * KORDER – 2) + 13 * NDATA + 4 * KORDER + 10 units.

   Workspace may be explicitly provided, if desired, by use of B2OPK/DB2OPK. The reference is

   ```
   CALL B2OPK (NDATA, XDATA, KORDER, XKNOT, MAXIT, WK,
                IWK)
   ```

   The additional arguments are as follows:

   MAXIT — Maximum number of iterations of Newton’s Method. (Input) A suggested value is 10.

   WK — Work array of length (NDATA – KORDER) * (3 * KORDER – 2) + 6 * NDATA + 2 * KORDER + 5.

   IWK — Work array of length NDATA.

2. Informational errors

Type	Code	
3	6	Newton’s method iteration did not converge.
4	3	The XDATA values must be distinct.
4	4	Interpolation matrix is singular. The XDATA values may be too close together.

3. The default value for MAXIT is 10, this can be overridden by calling B2OPK/DB2OPK directly with a larger value.

Algorithm

Given the abscissas x = XDATA for an interpolation problem and the order of the spline interpolant k = KORDER, BSOPK returns the knot sequence **t** = XKNOT that minimizes the constant in the error estimate

$$\| f - s \| \le c \, \| f^{(k)} \|$$

In the above formula, f is any function in C^k and s is the spline interpolant to f at the abscissas x with knot sequence **t**.

The algorithm is based on a routine described in de Boor (1978, page 204), which in turn is based on a theorem of Micchelli, Rivlin and Winograd (1976).

Example

In this example, we compute (for k = 3, …, 8) six spline interpolants s_k to $F(x) = \sin(10x^3)$ on the interval [0, 1]. The routine BSOPK is used to generate the knot sequences for s_k and then BSINT (page 450) is called to obtain the interpolant. We evaluate the absolute error

$$| s_k - F |$$

at 100 equally spaced points and print the maximum error for each k.

```
      INTEGER    KMAX, KMIN, NDATA
      PARAMETER  (KMAX=8, KMIN=3, NDATA=20)
C
      INTEGER    I, K, KORDER, NOUT
      REAL       ABS, AMAX1, BSCOEF(NDATA), BSVAL, DIF, DIFMAX, F,
     &           FDATA(NDATA), FLOAT, FT, SIN, ST, T, X, XDATA(NDATA),
     &           XKNOT(KMAX+NDATA), XT
      INTRINSIC  ABS, AMAX1, FLOAT, SIN
      EXTERNAL   BSINT, BSOPK, BSVAL, UMACH
C                                  Define function and tau function
      F(X) = SIN(10.0*X*X*X)
      T(X) = 1.0 - X*X
C                                  Set up data
      DO 10  I=1, NDATA
         XT       = FLOAT(I-1)/FLOAT(NDATA-1)
         XDATA(I) = T(XT)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
C                                  Loop over different orders
      DO 30  K=KMIN, KMAX
         KORDER = K
C                                  Generate knots
         CALL BSOPK (NDATA, XDATA, KORDER, XKNOT)
C                                  Interpolate
         CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
         DIFMAX = 0.0
         DO 20  I=1, 100
```

```
            XT     = FLOAT(I-1)/99.0
C                                  Evaluate spline
            ST     = BSVAL(XT,KORDER,XKNOT,NDATA,BSCOEF)
            FT     = F(XT)
            DIF    = ABS(FT-ST)
C                                  Compute maximum difference
            DIFMAX = AMAX1(DIF,DIFMAX)
   20  CONTINUE
C                                  Print maximum difference
         WRITE (NOUT,99998) KORDER, DIFMAX
   30 CONTINUE
C
99998 FORMAT (' ', I3, 5X, F9.4)
99999 FORMAT (' KORDER', 5X, 'Maximum difference', /)
      END
```

Output

```
KORDER   Maximum difference

 3        0.0096
 4        0.0018
 5        0.0005
 6        0.0004
 7        0.0007
 8        0.0035
```

BS2IN/DBS2IN (Single/Double precision)

Compute a two-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.

Usage

```
CALL BS2IN (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF,
            KXORD, KYORD, XKNOT, YKNOT, BSCOEF)
```

Arguments

NXDATA — Number of data points in the X-direction.   (Input)

XDATA — Array of length NXDATA containing the data points in the X-direction. (Input)
XDATA must be strictly increasing.

NYDATA — Number of data points in the Y-direction.   (Input)

YDATA — Array of length NYDATA containing the data points in the Y-direction. (Input)
YDATA must be strictly increasing.

FDATA — Array of size NXDATA by NYDATA containing the values to be interpolated.   (Input)
FDATA (I, J) is the value at (XDATA (I), YDATA(J)).

LDF — The leading dimension of FDATA exactly as specified in the dimension statement of the calling program. (Input)

KXORD — Order of the spline in the X-direction. (Input)
KXORD must be less than or equal to NXDATA.

KYORD — Order of the spline in the Y-direction. (Input)
KYORD must be less than or equal to NYDATA.

XKNOT — Array of length NXDATA + KXORD containing the knot sequence in the X-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYDATA + KYORD containing the knot sequence in the Y-direction. (Input)
YKNOT must be nondecreasing.

BSCOEF — Array of length NXDATA * NYDATA containing the tensor-product B-spline coefficients. (Output)
BSCOEF is treated internally as a matrix of size NXDATA by NYDATA.

Comments

1. Automatic workspace usage is

   BS2IN MAX((2 * KXORD – 1) * NXDATA, (2 * KYORD – 1) * NYDATA) + MAX((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA) + 3 * MAX(NXDATA, NYDATA) + NXDATA + NYDATA units, or

   DBS2IN 2 * MAX ((2 * KXORD – 1) * NXDATA, (2 * KYORD – 1) * NYDATA) + 2 * MAX ((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA) + 5 * MAX(NXDATA, NYDATA) + 2 * NXDATA + NYDATA units.

   Workspace may be explicitly provided, if desired, by use of B22IN/DB22IN. The reference is

   ```
   CALL B22IN (NXDATA, XDATA, NYDATA, YDATA, FDATA,
              LDF, KXORD, KYORD, XKNOT, YKNOT, BSCOEF,
              WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length NXDATA * NYDATA + MAX((2 * KXORD –1) NXDATA, (2 * KYORD – 1) * NYDATA) + MAX((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA) + 2 * MAX(NXDATA, NYDATA).

   IWK — Work array of length MAX(NXDATA, NYDATA).

2. Informational errors

Type	Code	
3	1	Interpolation matrix is nearly singular. LU factorization failed.

3	2	Interpolation matrix is nearly singular. Iterative refinement failed.
4	6	The `XDATA` values must be strictly increasing.
4	7	The `YDATA` values must be strictly increasing.
4	13	Multiplicity of the knots cannot exceed the order of the spline.
4	14	The knots must be nondecreasing.
4	15	The `I`-th smallest element of the data point array must be greater than the `I`-th knot and less than the (`I` + `K_ORD`)-th knot.
4	16	The largest element of the data point array must be greater than the (`N_DATA`)-th knot and less than or equal to the (`N_DATA` + `K_ORD`)-th knot.
4	17	The smallest element of the data point array must be greater than or equal to the first knot and less than the (`K_ORD` + 1)st knot.

Algorithm

The routine `BS2IN` computes a tensor product spline interpolant. The tensor product spline interpolant to data $\{(x_i, y_j, f_{ij})\}$, where $1 \le i \le N_x$ and $1 \le j \le N_y$, has the form

$$\sum_{m=1}^{N_y} B_{n,k_x,\mathbf{t}_x}(x) B_{m,k_y,\mathbf{t}_y}(y)$$

where k_x and k_y are the orders of the splines. (These numbers are passed to the subroutine in `KXORD` and `KYORD`, respectively.) Likewise, $\mathbf{t}_x$ and $\mathbf{t}_y$ are the corresponding knot sequences (`XKNOT` and `YKNOT`). The algorithm requires that

$$\mathbf{t}_x(k_x) \le x_i \le \mathbf{t}_x(N_x + 1) \qquad 1 \le i \le N_x$$

$$\mathbf{t}_y(k_y) \le y_j \le \mathbf{t}_y(N_y + 1) \qquad 1 \le j \le N_y$$

Tensor product spline interpolants in two dimensions can be computed quite efficiently by solving (repeatedly) two univariate interpolation problems. The computation is motivated by the following observations. It is necessary to solve the system of equations

$$\sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}(x_i) B_{m,k_y,\mathbf{t}_y}(y_j) = f_{ij}$$

Setting

$$h_{mi} = \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}(x_i)$$

we note that for each fixed i from 1 to N_x, we have N_y linear equations in the same number of unknowns as can be seen below:

$$\sum_{m=1}^{N_y} h_{mi} B_{m,k_y,\mathbf{t}_y}\left(y_j\right) = f_{ij}$$

The same matrix appears in all of the equations above:

$$\left[B_{m,k_y,\mathbf{t}_y}\left(y_j\right)\right] \quad 1 \le m, j \le N_y$$

Thus, we need only factor this matrix once and then apply this factorization to the N_x righthand sides. Once this is done and we have computed h_{mi}, then we must solve for the coefficients c_{nm} using the relation

$$\sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}\left(x_i\right) = h_{mi}$$

for m from 1 to N_y, which again involves one factorization and N_y solutions to the different right-hand sides. The routine `BS2IN` is based on the routine `SPLI2D` by de Boor (1978, page 347).

Example

In this example, a tensor product spline interpolant to a function f is computed. The values of the interpolant and the error on a 4 × 4 grid are displayed.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, LDF, NXDATA, NXKNOT, NXVEC, NYDATA,
     &           NYKNOT, NYVEC
      PARAMETER  (KXORD=5, KYORD=2, NXDATA=21, NXVEC=4, NYDATA=6,
     &           NYVEC=4, LDF=NXDATA, NXKNOT=NXDATA+KXORD,
     &           NYKNOT=NYDATA+KYORD)
C
      INTEGER    I, J, NOUT, NXCOEF, NYCOEF
      REAL       BSCOEF(NXDATA,NYDATA), F, FDATA(LDF,NYDATA), FLOAT,
     &           VALUE(NXVEC,NYVEC), X, XDATA(NXDATA), XKNOT(NXKNOT),
     &           XVEC(NXVEC), Y, YDATA(NYDATA), YKNOT(NYKNOT),
     &           YVEC(NYVEC)
      INTRINSIC  FLOAT
      EXTERNAL   BS2GD, BS2IN, BSNAK, UMACH
C                                 Define function
      F(X,Y) = X*X*X + X*Y
C                                 Set up interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
C                                 Set up interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/5.0
   20 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
C                                 Generate FDATA
      DO 40  I=1, NYDATA
         DO 30  J=1, NXDATA
```

```
            FDATA(J,I) = F(XDATA(J),YDATA(I))
   30  CONTINUE
   40 CONTINUE
C                                  Interpolate
      CALL BS2IN (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF, KXORD,
     &            KYORD, XKNOT, YKNOT, BSCOEF)
      NXCOEF = NXDATA
      NYCOEF = NYDATA
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
C                                  Print over a grid of
C                                  [0.0,1.0] x [0.0,1.0] at 16 points.
      DO 50  I=1, NXVEC
         XVEC(I) = FLOAT(I-1)/3.0
   50 CONTINUE
      DO 60  I=1, NYVEC
         YVEC(I) = FLOAT(I-1)/3.0
   60 CONTINUE
C                                  Evaluate spline
      CALL BS2GD (0, 0, NXVEC, XVEC, NYVEC, YVEC, KXORD, KYORD, XKNOT,
     &            YKNOT, NXCOEF, NYCOEF, BSCOEF, VALUE, NXVEC)
      DO 80  I=1, NXVEC
         DO 70  J=1, NYVEC
            WRITE (NOUT,'(3F15.4,F15.6)') XVEC(I), YVEC(J),
     &                                  VALUE(I,J),
     &                                  (F(XVEC(I),YVEC(J))-
     &                                  VALUE(I,J))
   70  CONTINUE
   80  CONTINUE
99999 FORMAT (13X, 'X', 14X, 'Y', 10X, 'S(X,Y)', 9X, 'Error')
      END
```

Output

```
    X              Y          S(X,Y)         Error
0.0000         0.0000         0.0000       0.000000
0.0000         0.3333         0.0000       0.000000
0.0000         0.6667         0.0000       0.000000
0.0000         1.0000         0.0000       0.000000
0.3333         0.0000         0.0370       0.000000
0.3333         0.3333         0.1481       0.000000
0.3333         0.6667         0.2593       0.000000
0.3333         1.0000         0.3704       0.000000
0.6667         0.0000         0.2963       0.000000
0.6667         0.3333         0.5185       0.000000
0.6667         0.6667         0.7407       0.000000
0.6667         1.0000         0.9630       0.000000
1.0000         0.0000         1.0000       0.000000
1.0000         0.3333         1.3333       0.000000
1.0000         0.6667         1.6667       0.000000
1.0000         1.0000         2.0000       0.000000
```

BS3IN/DBS3IN (Single/Double precision)

Compute a three-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.

Usage

```
CALL BS3IN (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA,
            FDATA, LDF, MDF, KXORD, KYORD, KZORD, XKNOT,
            YKNOT, ZKNOT, BSCOEF)
```

Arguments

NXDATA — Number of data points in the *x*-direction. (Input)

XDATA — Array of length `NXDATA` containing the data points in the *x*-direction. (Input)
`XDATA` must be increasing.

NYDATA — Number of data points in the *y*-direction. (Input)

YDATA — Array of length `NYDATA` containing the data points in the *y*-direction. (Input)
`YDATA` must be increasing.

NZDATA — Number of data points in the *z*-direction. (Input)

ZDATA — Array of length `NZDATA` containing the data points in the *z*-direction. (Input)
`ZDATA` must be increasing.

FDATA — Array of size `NXDATA` by `NYDATA` by `NZDATA` containing the values to be interpolated. (Input)
`FDATA` (`I`, `J`, `K`) contains the value at (`XDATA` (`I`), `YDATA`(`J`), `ZDATA`(`K`)).

LDF — Leading dimension of `FDATA` exactly as specified in the dimension statement of the calling program. (Input)

MDF — Middle dimension of `FDATA` exactly as specified in the dimension statement of the calling program. (Input)

KXORD — Order of the spline in the *x*-direction. (Input)
`KXORD` must be less than or equal to `NXDATA`.

KYORD — Order of the spline in the *y*-direction. (Input)
`KYORD` must be less than or equal to `NYDATA`.

KZORD — Order of the spline in the *z*-direction. (Input)
`KZORD` must be less than or equal to `NZDATA`.

XKNOT — Array of length `NXDATA` + `KXORD` containing the knot sequence in the *x*-direction. (Input)
`XKNOT` must be nondecreasing.

YKNOT — Array of length NYDATA + KYORD containing the knot sequence in the *y*-direction. (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length NZDATA + KZORD containing the knot sequence in the z-direction. (Input)
ZKNOT must be nondecreasing.

BSCOEF — Array of length NXDATA * NYDATA * NZDATA containing the tensor-product B-spline coefficients. (Output)
BSCOEF is treated internally as a matrix of size NXDATA by NYDATA by NZDATA.

Comments

1. Automatic workspace usage is

   BS3IN MAX((2 * KXORD – 1) * NXDATA, (2 * KYORD – 1) * NYDATA, (2 * KZORD – 1) * NZDATA) + MAX((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA, (3 * KZORD – 2) * NZDATA) + 3 * MAX(NXDATA, NYDATA, NZDATA) + NXDATA * NYDATA * NZDATA units, or

   DBS3IN 2 * MAX((2 * KXORD – 1) * NXDATA, (2 * KYORD – 1) * NYDATA, (2 * KZORD – 1) * NZDATA) + 2 * MAX ((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA, (3 * KZORD –2) * NZDATA) + 5 * MAX(NXDATA, NYDATA, NZDATA) + 2 * NXDATA * NYDATA * NZDATA units.

   Workspace may be explicitly provided, if desired, by use of B23IN/DB23IN. The reference is

   ```
   CALL B23IN (NXDATA, XDATA, NYDATA, YDATA, NZDAYA,
               ZDATA, FDATA, LDF, MDF, KXORD, KYORD,
               KZORD, XKNOT, YKNOT, ZKNOT, BSCOEF, WK,
               IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length MAX((2 * KXORD – 1) * NXDATA, (2 * KYORD – 1) * NYDATA, (2 * KZORD – 1) * NZDATA) + MAX((3 * KXORD – 2) * NXDATA, (3 * KYORD – 2) * NYDATA + (3 * KZORD – 2) * NZDATA) + NXDATA * NYDATA *NZDATA + 2 * MAX(NXDATA, NYDATA, NZDATA).

   IWK — Work array of length MAX(NXDATA, NYDATA, NZDATA).

2. Informational errors

Type	Code	
3	1	Interpolation matrix is nearly singular. LU factorization failed.
3	2	Interpolation matrix is nearly singular. Iterative refinement failed.

4	13	Multiplicity of the knots cannot exceed the order of the spline.
4	14	The knots must be nondecreasing.
4	15	The `I`-th smallest element of the data point array must be greater than the Ith knot and less than the (`I` + `K_ORD`)-th knot.
4	16	The largest element of the data point array must be greater than the (`N_DATA`)-th knot and less than or equal to the (`N_DATA` + `K_ORD`)-th knot.
4	17	The smallest element of the data point array must be greater than or equal to the first knot and less than the (`K_ORD` + 1)st knot.
4	18	The `XDATA` values must be strictly increasing.
4	19	The `YDATA` values must be strictly increasing.
4	20	The `ZDATA` values must be strictly increasing.

Algorithm

The routine `BS3IN` computes a tensor-product spline interpolant. The tensor-product spline interpolant to data $\{(x_i, y_j, z_k, f_{ijk})\}$, where $1 \le i \le N_x$, $1 \le j \le N_y$, and $1 \le k \le N_z$ has the form

$$\sum_{l=1}^{N_z} \sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}(x) B_{m,k_y,\mathbf{t}_y}(y) B_{l,k_z,\mathbf{t}_z}(z)$$

where k_x, k_y, and k_z are the orders of the splines (these numbers are passed to the subroutine in `KXORD`, `KYORD`, and `KZORD`, respectively). Likewise, $\mathbf{t}_x$, $\mathbf{t}_y$, and $\mathbf{t}_z$ are the corresponding knot sequences (`XKNOT`, `YKNOT`, and `ZKNOT`). The algorithm requires that

$$\begin{aligned} \mathbf{t}_x(k_x) &\le x_i \le \mathbf{t}_x(N_x + 1) \quad 1 \le i \le N_x \\ \mathbf{t}_y(k_y) &\le y_j \le \mathbf{t}_y(N_y + 1) \quad 1 \le j \le N_y \\ \mathbf{t}_z(k_z) &\le z_k \le \mathbf{t}_z(N_z + 1) \quad 1 \le k \le N_z \end{aligned}$$

Tensor-product spline interpolants can be computed quite efficiently by solving (repeatedly) three univariate interpolation problems. The computation is motivated by the following observations. It is necessary to solve the system of equations

$$\sum_{l=1}^{N_z} \sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}(x_i) B_{m,k_y,\mathbf{t}_y}(y_j) B_{l,k_z,\mathbf{t}_z}(z_k) = f_{ijk}$$

Setting

$$h_{lij} = \textstyle\sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}(x_i) B_{m,k_y,\mathbf{t}_y}(y_j)$$

we note that for each fixed pair *ij* we have N_z linear equations in the same number of unknowns as can be seen below:

$$\sum_{l=1}^{N_z} h_{lij} B_{l,k_z,\mathbf{t}_z}(z_k) = f_{ijk}$$

The same interpolation matrix appears in all of the equations above:

$$\left[B_{l,k_z,\mathbf{t}_z}(z_k)\right] \qquad 1 \le l, k \le N_z$$

Thus, we need only factor this matrix once and then apply it to the N_xN_y right-hand sides. Once this is done and we have computed h_{lij}, then we must solve for the coefficients c_{nml} using the relation

$$\sum_{m=1}^{N_y}\sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}(x_i) B_{m,k_y,\mathbf{t}_y}(y_j) = h_{lij}$$

that is the *bivariate* tensor-product problem addressed by the IMSL routine `BS2IN` (page 459). The interested reader should consult the algorithm description in the two-dimensional routine if more detail is desired. The routine `BS3IN` is based on the routine `SPLI2D` by de Boor (1978, page 347).

Example

In this example, a tensor-product spline interpolant to a function *f* is computed. The values of the interpolant and the error on a 4 × 4 × 2 grid are displayed.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, KZORD, LDF, MDF, NXDATA, NXKNOT, NXVEC,
     &           NYDATA, NYKNOT, NYVEC, NZDATA, NZKNOT, NZVEC
      PARAMETER  (KXORD=5, KYORD=2, KZORD=3, NXDATA=21, NXVEC=4,
     &           NYDATA=6, NYVEC=4, NZDATA=8, NZVEC=2, LDF=NXDATA,
     &           MDF=NYDATA, NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD,
     &           NZKNOT=NZDATA+KZORD)
C
      INTEGER    I, J, K, NOUT, NXCOEF, NYCOEF, NZCOEF
      REAL       BSCOEF(NXDATA,NYDATA,NZDATA), F,
     &           FDATA(LDF,MDF,NZDATA), FLOAT, VALUE(NXVEC,NYVEC,NZVEC)
     &           , X, XDATA(NXDATA), XKNOT(NXKNOT), XVEC(NXVEC), Y,
     &           YDATA(NYDATA), YKNOT(NYKNOT), YVEC(NYVEC), Z,
     &           ZDATA(NZDATA), ZKNOT(NZKNOT), ZVEC(NZVEC)
      INTRINSIC  FLOAT
      EXTERNAL   BS3GD, BS3IN, BSNAK, UMACH
C                                 Define function.
      F(X,Y,Z) = X*X*X + X*Y*Z
C                                 Set up X-interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Set up Y-interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/FLOAT(NYDATA-1)
   20 CONTINUE
```

```
C                                 Set up Z-interpolation points
      DO 30  I=1, NZDATA
         ZDATA(I) = FLOAT(I-1)/FLOAT(NZDATA-1)
   30 CONTINUE
C                                 Generate knots
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
      CALL BSNAK (NZDATA, ZDATA, KZORD, ZKNOT)
C                                 Generate FDATA
      DO 50  K=1, NZDATA
         DO 40  I=1, NYDATA
            DO 40  J=1, NXDATA
               FDATA(J,I,K) = F(XDATA(J),YDATA(I),ZDATA(K))
   40  CONTINUE
   50 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Interpolate
      CALL BS3IN (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA, FDATA,
     &            LDF, MDF, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
     &            BSCOEF)
C
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      NZCOEF = NZDATA
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print over a grid of
C                                 [-1.0,1.0] x [0.0,1.0] x [0.0,1.0]
C                                 at 32 points.
      DO 60  I=1, NXVEC
         XVEC(I) = 2.0*(FLOAT(I-1)/3.0) - 1.0
   60 CONTINUE
      DO 70  I=1, NYVEC
         YVEC(I) = FLOAT(I-1)/3.0
   70 CONTINUE
      DO 80  I=1, NZVEC
         ZVEC(I) = FLOAT(I-1)
   80 CONTINUE
C                                 Call the evaluation routine.
      CALL BS3GD (0, 0, 0, NXVEC, XVEC, NYVEC, YVEC, NZVEC, ZVEC,
     &            KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT, NXCOEF,
     &            NYCOEF, NZCOEF, BSCOEF, VALUE, NXVEC, NYVEC)
      DO 110  I=1, NXVEC
         DO 100  J=1, NYVEC
            DO 90  K=1, NZVEC
               WRITE (NOUT,’(4F13.4, F13.6)’) XVEC(I), YVEC(K),
     &                                      ZVEC(K), VALUE(I,J,K),
     &                                      F(XVEC(I),YVEC(J),ZVEC(K))
     &                                       - VALUE(I,J,K)
   90       CONTINUE
  100    CONTINUE
  110 CONTINUE
99999 FORMAT (10X, ’X’, 11X, ’Y’, 10X, ’Z’, 10X, ’S(X,Y,Z)’, 7X,
     &       ’Error’)
      END
```

Output

```
    X           Y           Z          S(X,Y,Z)       Error
-1.0000       0.0000       0.0000       -1.0000     0.000000
-1.0000       0.3333       1.0000       -1.0000     0.000000
-1.0000       0.0000       0.0000       -1.0000     0.000000
-1.0000       0.3333       1.0000       -1.3333     0.000000
-1.0000       0.0000       0.0000       -1.0000     0.000000
-1.0000       0.3333       1.0000       -1.6667     0.000000
-1.0000       0.0000       0.0000       -1.0000     0.000000
-1.0000       0.3333       1.0000       -2.0000     0.000000
-0.3333       0.0000       0.0000       -0.0370     0.000000
-0.3333       0.3333       1.0000       -0.0370     0.000000
-0.3333       0.0000       0.0000       -0.0370     0.000000
-0.3333       0.3333       1.0000       -0.1481     0.000000
-0.3333       0.0000       0.0000       -0.0370     0.000000
-0.3333       0.3333       1.0000       -0.2593     0.000000
-0.3333       0.0000       0.0000       -0.0370     0.000000
-0.3333       0.3333       1.0000       -0.3704     0.000000
 0.3333       0.0000       0.0000        0.0370     0.000000
 0.3333       0.3333       1.0000        0.0370     0.000000
 0.3333       0.0000       0.0000        0.0370     0.000000
 0.3333       0.3333       1.0000        0.1481     0.000000
 0.3333       0.0000       0.0000        0.0370     0.000000
 0.3333       0.3333       1.0000        0.2593     0.000000
 0.3333       0.0000       0.0000        0.0370     0.000000
 0.3333       0.3333       1.0000        0.3704     0.000000
 1.0000       0.0000       0.0000        1.0000     0.000000
 1.0000       0.3333       1.0000        1.0000     0.000000
 1.0000       0.0000       0.0000        1.0000     0.000000
 1.0000       0.3333       1.0000        1.3333     0.000000
 1.0000       0.0000       0.0000        1.0000     0.000000
 1.0000       0.3333       1.0000        1.6667     0.000000
 1.0000       0.0000       0.0000        1.0000     0.000000
 1.0000       0.3333       1.0000        2.0000     0.000000
```

BSVAL/DBSVAL (Single/Double precision)

Evaluate a spline, given its B-spline representation.

Usage

`BSVAL(X, KORDER, XKNOT, NCOEF, BSCOEF)`

Arguments

X — Point at which the spline is to be evaluated. (Input)

KORDER — Order of the spline. (Input)

XKNOT — Array of length `KORDER` + `NCOEF` containing the knot sequence. (Input)
`XKNOT` must be nondecreasing.

NCOEF — Number of B-spline coefficients. (Input)

BSCOEF — Array of length `NCOEF` containing the B-spline coefficients. (Input)

BSVAL — Value of the spline at `X`. (Output)

Comments

1. Automatic workspace usage is

   `BSVAL` 3 * `KORDER` units, or
   `DBSVAL` 6 * `KORDER` units.

   Workspace may be explicitly provided, if desired, by use of `B2VAL`/`DB2VAL`. The reference is

   ```
   CALL B2VAL(X, KORDER, XKNOT, NCOEF, BSCOEF, WK1,
              WK2, WK3)
   ```

   The additional arguments are as follows:

   WK1 — Work array of length `KORDER`.

   WK2 — Work array of length `KORDER`.

   WK3 — Work array of length `KORDER`.

2. Informational errors

Type	Code	
4	4	Multiplicity of the knots cannot exceed the order of the spline.
4	5	The knots must be nondecreasing.

Algorithm

The function `BSVAL` evaluates a spline (given its B-spline representation) at a specific point. It is a special case of the routine `BSDER` (page 471), which evaluates the derivative of a spline given its B-spline representation. The routine `BSDER` is based on the routine `BVALUE` by de Boor (1978, page 144).

Specifically, given the knot vector **t**, the number of coefficients *N*, the coefficient vector *a*, and a point *x*, `BSVAL` returns the number

$$\sum_{j=1}^{N} a_j B_{j,k}(x)$$

where $B_{j,k}$ is the *j*-th B-spline of order *k* for the knot sequence **t**. Note that this function routine arbitrarily treats these functions as if they were right continuous near `XKNOT(KORDER)` and left continuous near `XKNOT(NCOEF` + 1). Thus, if we have `KORDER` knots stacked at the left or right end point, and if we try to evaluate at these end points, then we will get the value of the limit from the interior of the interval.

Example

For an example of the use of BSVAL, see IMSL routine BSINT (page 450).

BSDER/DBSDER (Single/Double precision)

Evaluate the derivative of a spline, given its B-spline representation.

Usage

```
BSDER(IDERIV, X, KORDER, XKNOT, NCOEF, BSCOEF)
```

Arguments

IDERIV — Order of the derivative to be evaluated. (Input)
In particular, IDERIV = 0 returns the value of the spline.

X — Point at which the spline is to be evaluated. (Input)

KORDER — Order of the spline. (Input)

XKNOT — Array of length NCOEF + KORDER containing the knot sequence. (Input)
XKNOT must be nondecreasing.

NCOEF — Number of B-spline coefficients. (Input)

BSCOEF — Array of length NCOEF containing the B-spline coefficients. (Input)

BSDER — Value of the IDERIV-th derivative of the spline at X. (Output)

Comments

1. Automatic workspace usage is

   BSDER 3 * KORDER units, or
   DBSDER 6 * KORDER units.

   Workspace may be explicitly provided, if desired, by use of B2DER/DB2DER. The reference is

   ```
   CALL B2DER(IDERIV, X, KORDER, XKNOT, NCOEF, BSCOEF,
              WK1, WK2, WK3)
   ```

   The additional arguments are as follows:

   WK1 — Array of length KORDER.

   WK2 — Array of length KORDER.

   WK3 — Array of length KORDER.

2. Informational errors

Type	Code	
4	4	Multiplicity of the knots cannot exceed the order of the spline.
4	5	The knots must be nondecreasing.

Algorithm

The function `BSDER` produces the value of a spline or one of its derivatives (given its B-spline representation) at a specific point. The function `BSDER` is based on the routine `BVALUE` by de Boor (1978, page 144).

Specifically, given the knot vector **t**, the number of coefficients *N*, the coefficient vector *a*, the order of the derivative *i* and a point *x*, `BSDER` returns the number

$$\sum_{j=1}^{N} a_j B_{j,k}^{(i)}(x)$$

where $B_{j,k}$ is the *j*-th B-spline of order *k* for the knot sequence **t**. Note that this function routine arbitrarily treats these functions as if they were right continuous near `XKNOT(KORDER)` and left continuous near `XKNOT(NCOEF` + 1). Thus, if we have `KORDER` knots stacked at the left or right end point, and if we try to evaluate at these end points, then we will get the value of the limit from the interior of the interval.

Example

A spline interpolant to the function

$$f(x) = \sqrt{x}$$

is constructed using `BSINT` (page 450). The B-spline representation, which is returned by the IMSL routine `BSINT`, is then used by `BSDER` to compute the value and derivative of the interpolant. The output consists of the interpolation values and the error at the data points and the midpoints. In addition, we display the value of the derivative and the error at these same points.

```
      INTEGER    KORDER, NDATA, NKNOT
      PARAMETER  (KORDER=3, NDATA=5, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NCOEF, NOUT
      REAL       BSCOEF(NDATA), BSDER, BT0, BT1, DF, F, FDATA(NDATA),
     &           FLOAT, SQRT, X, XDATA(NDATA), XKNOT(NKNOT), XT
      INTRINSIC  FLOAT, SQRT
      EXTERNAL   BSDER, BSINT, BSNAK, UMACH
C                                  Define function and derivative
      F(X)  = SQRT(X)
      DF(X) = 0.5/SQRT(X)
C                                  Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I)/FLOAT(NDATA)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
```

```
C                                  Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                  Interpolate
      CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
C                                  Print on a finer grid
      NCOEF = NDATA
      XT    = XDATA(1)
C                                  Evaluate spline
      BT0   = BSDER(0,XT,KORDER,XKNOT,NCOEF,BSCOEF)
      BT1   = BSDER(1,XT,KORDER,XKNOT,NCOEF,BSCOEF)
      WRITE (NOUT,99998) XT, BT0, F(XT) - BT0, BT1, DF(XT) - BT1
      DO 20  I=2, NDATA
         XT  = (XDATA(I-1)+XDATA(I))/2.0
C                                  Evaluate spline
         BT0 = BSDER(0,XT,KORDER,XKNOT,NCOEF,BSCOEF)
         BT1 = BSDER(1,XT,KORDER,XKNOT,NCOEF,BSCOEF)
         WRITE (NOUT,99998) XT, BT0, F(XT) - BT0, BT1, DF(XT) - BT1
         XT  = XDATA(I)
C                                  Evaluate spline
         BT0 = BSDER(0,XT,KORDER,XKNOT,NCOEF,BSCOEF)
         BT1 = BSDER(1,XT,KORDER,XKNOT,NCOEF,BSCOEF)
         WRITE (NOUT,99998) XT, BT0, F(XT) - BT0, BT1, DF(XT) - BT1
   20 CONTINUE
99998 FORMAT (' ', F6.4, 5X, F7.4, 3X, F10.6, 5X, F8.4, 3X, F10.6)
99999 FORMAT (6X, 'X', 8X, 'S(X)', 7X, 'Error', 8X, 'S''(X)', 8X,
     &       'Error', /)
      END
```

Output

```
     X        S(X)       Error        S'(X)        Error

0.2000      0.4472     0.000000       1.0423     0.075738
0.3000      0.5456     0.002084       0.9262    -0.013339
0.4000      0.6325     0.000000       0.8101    -0.019553
0.5000      0.7077    -0.000557       0.6940     0.013071
0.6000      0.7746     0.000000       0.6446     0.000869
0.7000      0.8366     0.000071       0.5952     0.002394
0.8000      0.8944     0.000000       0.5615    -0.002525
0.9000      0.9489    -0.000214       0.5279    -0.000818
1.0000      1.0000     0.000000       0.4942     0.005814
```

BS1GD/DBS1GD (Single/Double precision)

Evaluate the derivative of a spline on a grid, given its B-spline representation.

Usage

```
CALL BS1GD (IDERIV, N, XVEC, KORDER, XKNOT, NCOEF, BSCOEF,
            VALUE)
```

Arguments

IDERIV — Order of the derivative to be evaluated.   (Input)
In particular, IDERIV = 0 returns the value of the spline.

N — Length of vector XVEC.   (Input)

XVEC — Array of length N containing the points at which the spline is to be evaluated.   (Input)
XVEC should be strictly increasing.

KORDER — Order of the spline.   (Input)

XKNOT — Array of length NCOEF + KORDER containing the knot sequence.   (Input)
XKNOT must be nondecreasing.

NCOEF — Number of B-spline coefficients.   (Input)

BSCOEF — Array of length NCOEF containing the B-spline coefficients.   (Input)

VALUE — Array of length N containing the values of the IDERIV-th derivative of the spline at the points in XVEC.   (Output)

Comments

1. Automatic workspace usage is

   BS1GD (NCOEF − KORDER + 1) * (KORDER + 1) + 3 * N + (KORDER + 3) * KORDER + 1 units, or

   DBS1GD 2 * (NCOEF − KORDER + 1) * (KORDER +1) + 5 * N + 2 * (KORDER + 3) * KORDER + 2 units.

   Workspace may be explicitly provided, if desired, by use of B21GD/DB21GD. The reference is

   ```
   CALL B21GD (IDERIV, N, XVEC, KORDER, XKNOT, NCOEF,
               BSCOEF, VALUE, RWK1, RWK2, IWK3, RWK4,
               RWK5, RWK6)
   ```

   The additional arguments are as follows:

   RWK1 — Real array of length KORDER * (NCOEF − KORDER + 1).

   RWK2 — Real array of length NCOEF − KORDER + 2.

   IWK3 — Integer array of length N.

   RWK4 — Real array of length N.

   RWK5 — Real array of length N.

   RWK6 — Real array of length (KORDER + 3) * KORDER

2. Informational error

Type	Code	
4	5	The points in XVEC must be strictly increasing

Algorithm

The routine BS1GD evaluates a B-spline (or its derivative) at a vector of points. That is, given a vector *x* of length *n* satisfying $x_i < x_{i+1}$ for *i* = 1, …, *n* − 1, a derivative value *j*, and a B-spline *s* that is represented by a knot sequence and coefficient sequence, this routine returns the values

$$s^{(j)}\left(x_i\right) \quad i = 1, \ldots, n$$

in the array VALUE. The functionality of this routine is the same as that of BSDER (page 471) called in a loop, however BS1GD should be much more efficient. This routine converts the B-spline representation to piecewise polynomial form using the IMSL routine BSCPP (page 504), and then uses the IMSL routine PPVAL (page 505) for evaluation.

Example

To illustrate the use of BS1GD, we modify the example program for BSDER (page 471). In this example, a quadratic (order 3) spline interpolant to *F* is computed. The values and derivatives of this spline are then compared with the exact function and derivative values. The routine BS1GD is based on the routines BSPLPP and PPVALU in de Boor (1978, page 89).

```
      INTEGER    KORDER, NDATA, NKNOT, NFGRID
      PARAMETER  (KORDER=3, NDATA=5, NKNOT=NDATA+KORDER, NFGRID = 9)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, NCOEF, NOUT
      REAL       ANS0(NFGRID), ANS1(NFGRID), BSCOEF(NDATA),
     &           FDATA(NDATA),
     &           X, XDATA(NDATA), XKNOT(NKNOT), XVEC(NFGRID)
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  FLOAT, SQRT
      REAL       FLOAT, SQRT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BS1GD, BSINT, BSNAK, UMACH
      REAL       DF, F
C
      F(X)  = SQRT(X)
      DF(X) = 0.5/SQRT(X)
C
      CALL UMACH (2, NOUT)
C                                 Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I)/FLOAT(NDATA)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Interpolate
      CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
      WRITE (NOUT,99999)
C                                 Print on a finer grid
```

```
      NCOEF   = NDATA
      XVEC(1) = XDATA(1)
      DO 20  I=2, 2*NDATA - 2, 2
         XVEC(I)   = (XDATA(I/2+1)+XDATA(I/2))/2.0
         XVEC(I+1) = XDATA(I/2+1)
   20 CONTINUE
      CALL BS1GD (0, 2*NDATA-1, XVEC, KORDER, XKNOT, NCOEF, BSCOEF,
     &            ANS0)
      CALL BS1GD (1, 2*NDATA-1, XVEC, KORDER, XKNOT, NCOEF, BSCOEF,
     &            ANS1)
      DO 30  I=1, 2*NDATA - 1
         WRITE (NOUT,99998) XVEC(I), ANS0(I), F(XVEC(I)) - ANS0(I),
     &                     ANS1(I), DF(XVEC(I)) - ANS1(I)
   30 CONTINUE
99998 FORMAT (’ ’, F6.4, 5X, F7.4, 5X, F8.4, 5X, F8.4, 5X, F8.4)
99999 FORMAT (6X, ’X’, 8X, ’S(X)’, 7X, ’Error’, 8X, ’S’’(X)’, 8X,
     &        ’Error’, /)
      END
```

Output

```
     X        S(X)       Error        S’(X)        Error

0.2000      0.4472       0.0000       1.0423       0.0757
0.3000      0.5456       0.0021       0.9262      -0.0133
0.4000      0.6325       0.0000       0.8101      -0.0196
0.5000      0.7077      -0.0006       0.6940       0.0131
0.6000      0.7746       0.0000       0.6446       0.0009
0.7000      0.8366       0.0001       0.5952       0.0024
0.8000      0.8944       0.0000       0.5615      -0.0025
0.9000      0.9489      -0.0002       0.5279      -0.0008
1.0000      1.0000       0.0000       0.4942       0.0058
```

BSITG/DBSITG (Single/Double precision)

Evaluate the integral of a spline, given its B-spline representation.

Usage

`BSITG(A, B, KORDER, XKNOT, NCOEF, BSCOEF)`

Arguments

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

KORDER — Order of the spline. (Input)

XKNOT — Array of length `KORDER` + `NCOEF` containing the knot sequence. (Input)
`XKNOT` must be nondecreasing.

NCOEF — Number of B-spline coefficients. (Input)

BSCOEF — Array of length `NCOEF` containing the B-spline coefficients. (Input)

BSITG — Value of the integral of the spline from `A` to `B`. (Output)

Comments

1. Automatic workspace usage is

   `BSITG` 4 * (`KORDER` + 1) units, or

   `DBSITG` 8 * (`KORDER` + 1) units.

   Workspace may be explicitly provided, if desired, by use of `B2ITG`/`DB2ITG`. The reference is

   ```
   CALL B2ITG(A, B, KORDER, XKNOT, NCOEF, BSCOEF,
              TCOEF, AJ, DL, DR)
   ```

   The additional arguments are as follows:

   TCOEF — Work array of length `KORDER` + 1.

   AJ — Work array of length `KORDER` + 1.

   DL — Work array of length `KORDER` + 1.

   DR — Work array of length `KORDER` + 1.

2. Informational errors

Type	Code	
3	7	The upper and lower endpoints of integration are equal.
3	8	The lower limit of integration is less than `XKNOT(KORDER)`.
3	9	The upper limit of integration is greater than `XKNOT(NCOEF` + 1).
4	4	Multiplicity of the knots cannot exceed the order of the spline.
4	5	The knots must be nondecreasing.

Algorithm

The function `BSITG` computes the integral of a spline given its B-spline representation. Specifically, given the knot sequence **t** = `XKNOT`, the order *k* = `KORDER`, the coefficients *a* = `BSCOEF` , *n* = `NCOEF` and an interval [*a*, *b*], `BSITG` returns the value

$$\int_a^b \sum_{i=1}^{n} a_i B_{i,k,\mathbf{t}}(x)dx$$

This routine uses the identity (22) on page 151 of de Boor (1978), and it assumes that $\mathbf{t}_1 = \ldots = \mathbf{t}_k$ and $\mathbf{t}_{n+1} = \ldots = \mathbf{t}_{n+k}$.

Example

We integrate the quartic ($k = 5$) spline that interpolates x^3 at the points $\{i/10 : i = -10, \ldots, 10\}$ over the interval [0, 1]. The exact answer is 1/4 since the interpolant reproduces cubic polynomials.

```
      INTEGER    KORDER, NDATA, NKNOT
      PARAMETER  (KORDER=5, NDATA=21, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NCOEF, NOUT
      REAL       A, B, BSCOEF(NDATA), BSITG, ERROR, EXACT, F,
     &           FDATA(NDATA), FI, FLOAT, VAL, X, XDATA(NDATA),
     &           XKNOT(NKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BSINT, BSITG, BSNAK, UMACH
C                                 Define function and integral
      F(X)  = X*X*X
      FI(X) = X**4/4.0
C                                 Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-11)/10.0
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Interpolate
      CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      NCOEF = NDATA
      A     = 0.0
      B     = 1.0
C                                 Integrate from A to B
      VAL   = BSITG(A,B,KORDER,XKNOT,NCOEF,BSCOEF)
      EXACT = FI(B) - FI(A)
      ERROR = EXACT - VAL
C                                 Print results
      WRITE (NOUT,99999) A, B, VAL, EXACT, ERROR
99999 FORMAT (’ On the closed interval (’, F3.1, ’,’, F3.1,
     &       ’) we have :’, /, 1X, ’Computed Integral = ’, F10.5, /,
     &       1X, ’Exact Integral    = ’, F10.5, /, 1X, ’Error         ’
     &       , ’    = ’, F10.6, /, /)
      END
```

Output

```
On the closed interval (0.0,1.0) we have :
Computed Integral =    0.25000
Exact Integral    =    0.25000
Error             =   0.000000
```

BS2VL/DBS2VL (Single/Double precision)

Evaluate a two-dimensional tensor-product spline, given its tensor-product B-spline representation.

Usage

```
BS2VL(X, Y, KXORD, KYORD, XKNOT, YKNOT, NXCOEF, NYCOEF,
      BSCOEF)
```

Arguments

X — X-coordinate of the point at which the spline is to be evaluated. (Input)

Y — Y-coordinate of the point at which the spline is to be evaluated. (Input)

KXORD — Order of the spline in the X-direction. (Input)

KYORD — Order of the spline in the Y-direction. (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction. (Input)
YKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction. (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction. (Input)

BSCOEF — Array of length NXCOEF * NYCOEF containing the tensor-product B-spline coefficients. (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF.

BS2VL — Value of the spline at (X, Y). (Output)

Comments

Automatic workspace usage is

BS2VL 3 * MAX(KXORD, KYORD) + KYORD units, or
DBS2VL 6 * MAX(KXORD, KYORD) + 2 * KYORD units.

Workspace may be explicitly provided, if desired, by use of B22VL/DB22VL. The reference is

```
CALL B22VL(X, Y, KXORD, KYORD, XKNOT, YKNOT, NXCOEF,
           NYCOEF, BSCOEF, WK)
```

The additional argument is

WK — Work array of length 3 * MAX(KXORD, KYORD) + KYORD.

Algorithm

The function BS2VL evaluates a bivariate tensor product spline (represented as a linear combination of tensor product B-splines) at a given point. This routine is a special case of the routine BS2DR (page 480), which evaluates partial derivatives of such a spline. (The value of a spline is its zero-th derivative.) For more information see de Boor (1978, pages 351−353).

This routine returns the value of the function *s* at a point (*x*, *y*) given the coefficients *c* by computing

$$s(x, y) = \sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}(x) B_{m,k_y,\mathbf{t}_y}(y)$$

where k_x and k_y are the orders of the splines. (These numbers are passed to the subroutine in KXORD and KYORD, respectively.) Likewise, $\mathbf{t}_x$ and $\mathbf{t}_y$ are the corresponding knot sequences (XKNOT and YKNOT).

Example

For an example of the use of BS2VL, see IMSL routine BS2IN (page 459).

BS2DR/DBS2DR (Single/Double precision)

Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation.

Usage

```
BS2DR(IXDER, IYDER, X, Y, KXORD, KYORD, XKNOT, YKNOT,
      NXCOEF, NYCOEF, BSCOEF)
```

Arguments

IXDER — Order of the derivative in the X-direction. (Input)

IYDER — Order of the derivative in the Y-direction. (Input)

X — X-coordinate of the point at which the spline is to be evaluated. (Input)

Y — Y-coordinate of the point at which the spline is to be evaluated. (Input)

KXORD — Order of the spline in the X-direction. (Input)

KYORD — Order of the spline in the Y-direction. (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction. (Input)
YKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction. (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction. (Input)

BSCOEF — Array of length NXCOEF * NYCOEF containing the tensor-product B-spline coefficients. (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF.

BS2DR — Value of the (IXDER, IYDER) derivative of the spline at (X, Y). (Output)

Comments

1. Automatic workspace usage is

   BS2DR 3 * MAX(KXORD, KYORD) + KYORD units, or

   DBS2DR 6 * MAX(KXORD, KYORD) + 2 * KYORD units.

   Workspace may be explicitly provided, if desired, by use of B22DR/DB22DR. The reference is

   ```
   CALL B22DR(IXDER, IYDER, X, Y, KXORD, KYORD, XKNOT,
               YKNOT, NXCOEF, NYCOEF, BSCOEF, WK)
   ```

   The additional argument is

   WK — Work array of length 3 * MAX(KXORD, KYORD) + KYORD.

2. Informational errors

Type	Code	
3	1	The point X does not satisfy XKNOT(KXORD) .LE. X .LE. XKNOT(NXCOEF + 1).
3	2	The point Y does not satisfy YKNOT(KYORD) .LE. Y .LE. YKNOT(NYCOEF + 1).

Algorithm

The routine BS2DR evaluates a partial derivative of a bivariate tensor-product spline (represented as a linear combination of tensor product B-splines) at a given point; see de Boor (1978, pages 351−353).

This routine returns the value of $s^{(p,q)}$at a point (x, y) given the coefficients c by computing

$$s^{(p,q)}(x,y)=\sum_{m=1}^{N_y}\sum_{n=1}^{N_x}c_{nm}B^{(p)}_{n,k_x,\mathbf{t}_x}(x)B^{(q)}_{m,k_y,\mathbf{t}_y}(y)$$

where k_x and k_y are the orders of the splines. (These numbers are passed to the subroutine in KXORD and KYORD, respectively.) Likewise, $\mathbf{t}_x$ and $\mathbf{t}_y$ are the corresponding knot sequences (XKNOT and YKNOT).

Example

In this example, a spline interpolant *s* to a function *f* is constructed. We use the IMSL routine `BS2IN` (page 459) to compute the interpolant and then `BS2DR` is employed to compute $s^{(2,1)}(x, y)$. The values of this partial derivative and the error are computed on a 4 × 4 grid and then displayed.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, LDF, NXDATA, NXKNOT, NYDATA, NYKNOT
      PARAMETER  (KXORD=5, KYORD=3, NXDATA=21, NYDATA=6, LDF=NXDATA,
     &           NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD)
C
      INTEGER    I, J, NOUT, NXCOEF, NYCOEF
      REAL       BS2DR, BSCOEF(NXDATA,NYDATA), F, F21,
     &           FDATA(LDF,NYDATA), FLOAT, S21, X, XDATA(NXDATA),
     &           XKNOT(NXKNOT), Y, YDATA(NYDATA), YKNOT(NYKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BS2DR, BS2IN, BSNAK, UMACH
C                                 Define function and (2,1) derivative
      F(X,Y)   = X*X*X*X + X*X*X*Y*Y
      F21(X,Y) = 12.0*X*Y
C                                 Set up interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
C                                 Set up interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/5.0
   20 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
C                                 Generate FDATA
      DO 40  I=1, NYDATA
         DO 30  J=1, NXDATA
            FDATA(J,I) = F(XDATA(J),YDATA(I))
   30  CONTINUE
   40 CONTINUE
C                                 Interpolate
      CALL BS2IN (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF, KXORD,
     &            KYORD, XKNOT, YKNOT, BSCOEF)
      NXCOEF = NXDATA
      NYCOEF = NYDATA
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print (2,1) derivative over a
C                                 grid of [0.0,1.0] x [0.0,1.0]
C                                 at 16 points.
      DO 60  I=1, 4
         DO 50  J=1, 4
            X   = FLOAT(I-1)/3.0
            Y   = FLOAT(J-1)/3.0
C                                 Evaluate spline
            S21 = BS2DR(2,1,X,Y,KXORD,KYORD,XKNOT,YKNOT,NXCOEF,NYCOEF,
     &            BSCOEF)
```

```
            WRITE (NOUT,’(3F15.4, F15.6)’) X, Y, S21, F21(X,Y) - S21
   50  CONTINUE
   60 CONTINUE
99999 FORMAT (39X, ’(2,1)’, /, 13X, ’X’, 14X, ’Y’, 10X, ’S    (X,Y)’,
     &        5X, ’Error’)
      END
```

Output

```
                                       (2,1)
     X              Y              S    (X,Y)     Error
0.0000         0.0000         0.0000       0.000000
0.0000         0.3333         0.0000       0.000000
0.0000         0.6667         0.0000       0.000000
0.0000         1.0000         0.0000       0.000001
0.3333         0.0000         0.0000       0.000000
0.3333         0.3333         1.3333       0.000002
0.3333         0.6667         2.6667      -0.000002
0.3333         1.0000         4.0000       0.000008
0.6667         0.0000         0.0000       0.000006
0.6667         0.3333         2.6667      -0.000011
0.6667         0.6667         5.3333       0.000028
0.6667         1.0000         8.0001      -0.000134
1.0000         0.0000        -0.0004       0.000439
1.0000         0.3333         4.0003      -0.000319
1.0000         0.6667         7.9996       0.000363
1.0000         1.0000        12.0005      -0.000458
```

BS2GD/DBS2GD (Single/Double precision)

Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

Usage

```
CALL BS2GD (IXDER, IYDER, NX, XVEC, NY, YVEC, KXORD, KYORD,
            XKNOT, YKNOT, NXCOEF, NYCOEF, BSCOEF, VALUE,
            LDVALU)
```

Arguments

IXDER — Order of the derivative in the X-direction. (Input)

IYDER — Order of the derivative in the Y-direction. (Input)

NX — Number of grid points in the X-direction. (Input)

XVEC — Array of length NX containing the X-coordinates at which the spline is to be evaluated. (Input)
The points in XVEC should be strictly increasing.

NY — Number of grid points in the Y-direction. (Input)

YVEC — Array of length NY containing the Y-coordinates at which the spline is to be evaluated. (Input)
The points in YVEC should be strictly increasing.

KXORD — Order of the spline in the X-direction.   (Input)

KYORD — Order of the spline in the Y-direction.   (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction.   (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction.   (Input)
YKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction.   (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction.   (Input)

BSCOEF — Array of length NXCOEF * NYCOEF containing the tensor-product B-spline coefficients.   (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF.

VALUE — Value of the (IXDER, IYDER) derivative of the spline on the NX by NY grid.   (Output)
VALUE (I, J) contains the derivative of the spline at the point
(XVEC(I), YVEC(J)).

LDVALU — Leading dimension of VALUE exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   BS2GD KXORD * (NX + KXORD+ IXDER + 1) + KYORD * (NY + KYORD + IYDER + 1) + NX + NY units, or

   DBS2GD 2 * (KXORD * (NX + KXORD + IXDER + 1) + KYORD * (NY + KYORD + IYDER + 1)) + NX + NY

   Workspace may be explicitly provided, if desired, by use of B22GD/DB22GD. The reference is

```
CALL B22GD (IXDER, IYDER, XVEC, NX, YVEC, NY, KXORD,
            KYORD, XKNOT, YKNOT, NXCOEF, NYCOEF,
            BSCOEF, VALUE, LDVALU, LEFTX, LEFTY, A,
            B, DBIATX, DBIATY, BX, BY)
```

   The additional arguments are as follows:

   LEFTX — Integer work array of length NX.

   LEFTY — Integer work array of length NY.

   A — Work array of length KXORD * KXORD.

   B — Work array of length KYORD * KYORD.

   DBIATX — Work array of length KXORD * (IXDER + 1).

DBIATY — Work array of length KYORD * (IYDER + 1).

BX — Work array of length KXORD * NX.

BY — Work array of length KYORD * NY.

2 Informational errors

Type	Code	
3	1	XVEC(I) does not satisfy XKNOT (KXORD) .LE. XVEC(I) .LE. XKNOT(NXCOEF + 1)
3	2	YVEC(I) does not satisfy YKNOT (KYORD) .LE. YVEC(I) .LE. YKNOT(NYCOEF + 1)
4	3	XVEC is not strictly increasing.
4	4	YVEC is not strictly increasing.

Algorithm

The routine BS2GD evaluates a partial derivative of a bivariate tensor-product spline (represented as a linear combination of tensor-product B-splines) on a grid of points; see de Boor (1978, pages 351−353).

This routine returns the values of s$^{(p,q)}$on the grid (x_i, y_j) for i = 1, …, nx and j = 1, …, ny given the coefficients c by computing (for all (x, y) in the grid)

$$s^{(p,q)}(x, y) = \sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nm} B_{n,k_x,\mathbf{t}_x}^{(p)}(x) B_{m,k_y,\mathbf{t}_y}^{(q)}(y)$$

where k_x and k_y are the orders of the splines. (These numbers are passed to the subroutine in KXORD and KYORD, respectively.) Likewise, $\mathbf{t}_x$ and $\mathbf{t}_y$ are the corresponding knot sequences (XKNOT and YKNOT). The grid must be ordered in the sense that $x_i < x_{i+1}$ and $y_j < y_{j+1}$.

Example

In this example, a spline interpolant s to a function f is constructed. We use the IMSL routine BS2IN (page 459) to compute the interpolant and then BS2GD is employed to compute $s^{(2,1)}$ (x, y) on a grid. The values of this partial derivative and the error are computed on a 4 × 4 grid and then displayed.

```
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, J, KXORD, KYORD, LDF, NOUT, NXCOEF, NXDATA,
     &          NYCOEF, NYDATA
      REAL       DCCFD(21,6), DOCBSC(21,6), DOCXD(21), DOCXK(26),
     &          DOCYD(6), DOCYK(9), F, F21, FLOAT, VALUE(4,4),
     &          X, XVEC(4), Y, YVEC(4)
      INTRINSIC  FLOAT
      EXTERNAL   BS2GD, BS2IN, BSNAK, UMACH
C                                 Define function and derivative
      F(X,Y)   = X*X*X*X + X*X*X*Y*Y
      F21(X,Y) = 12.0*X*Y
```

```
C               yj                  Initialize/Setup
      CALL UMACH (2, NOUT)
      KXORD  = 5
      KYORD  = 3
      NXDATA = 21
      NYDATA = 6
      LDF    = NXDATA
C                                 Set up interpolation points
      DO 10  I=1, NXDATA
         DOCXD(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Set up interpolation points
      DO 20  I=1, NYDATA
         DOCYD(I) = FLOAT(I-1)/5.0
   20 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NXDATA, DOCXD, KXORD, DOCXK)
C                                 Generate knot sequence
      CALL BSNAK (NYDATA, DOCYD, KYORD, DOCYK)
C                                 Generate FDATA
      DO 40  I=1, NYDATA
         DO 30  J=1, NXDATA
            DCCFD(J,I) = F(DOCXD(J),DOCYD(I))
   30  CONTINUE
   40 CONTINUE
C                                 Interpolate
      CALL BS2IN (NXDATA, DOCXD, NYDATA, DOCYD, DCCFD, LDF, KXORD,
     &            KYORD, DOCXK, DOCYK, DOCBSC)
C                                 Print (2,1) derivative over a
C                                 grid of [0.0,1.0] x [0.0,1.0]
C                                 at 16 points.
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      WRITE (NOUT,99999)
      DO 50  I=1, 4
         XVEC(I) = FLOAT(I-1)/3.0
         YVEC(I) = XVEC(I)
   50 CONTINUE
      CALL BS2GD (2, 1, 4, XVEC, 4, YVEC, KXORD, KYORD, DOCXK, DOCYK,
     &            NXCOEF, NYCOEF, DOCBSC, VALUE, 4)
      DO 70  I=1, 4
         DO 60  J=1, 4
            WRITE (NOUT,'(3F15.4,F15.6)') XVEC(I), YVEC(J),
     &                                  VALUE(I,J),
     &                                  F21(XVEC(I),YVEC(J)) -
     &                                  VALUE(I,J)
   60  CONTINUE
   70 CONTINUE
99999 FORMAT (39X, '(2,1)', /, 13X, 'X', 14X, 'Y', 10X, 'S    (X,Y)',
     &       5X, 'Error')
      END
```

Output

```
                               (2,1)
    X              Y          S    (X,Y)     Error
0.0000         0.0000         0.0000       0.000000
0.0000         0.3333         0.0000       0.000000
0.0000         0.6667         0.0000       0.000000
```

```
0.0000          1.0000          0.0000        0.000001
0.3333          0.0000          0.0000       -0.000001
0.3333          0.3333          1.3333        0.000001
0.3333          0.6667          2.6667       -0.000003
0.3333          1.0000          4.0000        0.000008

0.6667          0.0000          0.0000       -0.000001
0.6667          0.3333          2.6667       -0.000009
0.6667          0.6667          5.3333        0.000037
0.6667          1.0000          8.0001       -0.000120
1.0000          0.0000         -0.0005        0.000488
1.0000          0.3333          4.0003       -0.000320
1.0000          0.6667          7.9994        0.000610
1.0000          1.0000         12.0005       -0.000488
```

BS2IG/DBS2IG (Single/Double precision)

Evaluate the integral of a tensor-product spline on a rectangular domain, given its tensor-product B-spline representation.

Usage

```
BS2IG(A, B, C, D, KXORD, KYORD, XKNOT, YKNOT, NXCOEF,
      NYCOEF, BSCOEF)
```

Arguments

A — Lower limit of the X-variable. (Input)

B — Upper limit of the X-variable. (Input)

C — Lower limit of the Y-variable. (Input)

D — Upper limit of the Y-variable. (Input)

KXORD — Order of the spline in the X-direction. (Input)

KYORD — Order of the spline in the Y-direction. (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction. (Input)
YKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction. (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction. (Input)

BSCOEF — Array of length NXCOEF * NYCOEF containing the tensor-product B-spline coefficients. (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF.

BS2IG — Integral of the spline over the rectangle (A, B) by (C, D). (Output)

Comments

1. Automatic workspace usage is

   `BS2IG` 4 * (`MAX`(`KXORD`, `KYORD`) + 1) + `NYCOEF` units, or

   `DBS2IG` 8 * (`MAX`(`KXORD`, `KYORD`) + 1) + 2 * `NYCOEF` units.

   Workspace may be explicitly provided, if desired, by use of `B22IG`/`DB22IG`. The reference is

   ```
   CALL B22IG(A, B, C, D, KXORD, KYORD, XKNOT, YKNOT,
               NXCOEF, NYCOEF, BSCOEF, WK)
   ```

   The additional argument is

   WK — Work array of length 4 * (`MAX`(`KXORD`, `KYORD`) + 1) + `NYCOEF`.

2. Informational errors

Type	Code	
3	1	The lower limit of the `X`-integration is less than `XKNOT`(`KXORD`).
3	2	The upper limit of the `X`-integration is greater than `XKNOT`(`NXCOEF` + 1).
3	3	The lower limit of the `Y`-integration is less than `YKNOT`(`KYORD`).
3	4	The upper limit of the `Y`-integration is greater than `YKNOT`(`NYCOEF` + 1).
4	13	Multiplicity of the knots cannot exceed the order of the spline.
4	14	The knots must be nondecreasing.

Algorithm

The function `BS2IG` computes the integral of a tensor-product two-dimensional spline given its B-spline representation. Specifically, given the knot sequence $\mathbf{t}_x$ = `XKNOT`, $\mathbf{t}_y$ = `YKNOT`, the order k_x = `KXORD`, k_y = `KYORD`, the coefficients β = `BSCOEF`, the number of coefficients n_x = `NXCOEF`, n_y = `NYCOEF` and a rectangle $[a, b]$ by $[c, d]$, `BS2IG` returns the value

$$\int_a^b \int_c^d \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \beta_{ij} B_{ij} \; dy \; dx$$

where

$$B_{i,j}(x, y) = B_{i,k_x,\mathbf{t}_x}(x) B_{j,k_y,\mathbf{t}_y}(y)$$

This routine uses the identity (22) on page 151 of de Boor (1978). It assumes (for all knot sequences) that the first and last k knots are stacked, that is, $t_1 = \ldots = t_k$ and $t_{n+1} = \ldots = t_{n+k}$, where k is the order of the spline in the x or y direction.

Example

We integrate the two-dimensional tensor-product quartic ($k_x = 5$) by linear ($k_y = 2$) spline that interpolates $x^3 + xy$ at the points $\{(i/10, j/5) : i = -10, \ldots, 10$ and $j = 0, \ldots, 5\}$ over the rectangle $[0, 1] \times [.5, 1]$. The exact answer is 5/16.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, LDF, NXDATA, NXKNOT, NYDATA, NYKNOT
      PARAMETER  (KXORD=5, KYORD=2, NXDATA=21, NYDATA=6, LDF=NXDATA,
     &           NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD)
C
      INTEGER    I, J, NOUT, NXCOEF, NYCOEF
      REAL       A, B, BS2IG, BSCOEF(NXDATA,NYDATA), C, D, F,
     &           FDATA(LDF,NYDATA), FI, FLOAT, VAL, X, XDATA(NXDATA),
     &           XKNOT(NXKNOT), Y, YDATA(NYDATA), YKNOT(NYKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BS2IG, BS2IN, BSNAK, UMACH
C                                 Define function and integral
      F(X,Y)      = X*X*X + X*Y
      FI(A,B,C,D) = .25*((B**4-A**4)*(D-C)+(B*B-A*A)*(D*D-C*C))
C                                 Set up interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
C                                 Set up interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/5.0
   20 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
C                                 Generate FDATA
      DO 40  I=1, NYDATA
         DO 30  J=1, NXDATA
            FDATA(J,I) = F(XDATA(J),YDATA(I))
   30  CONTINUE
   40 CONTINUE
C                                 Interpolate
      CALL BS2IN (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF, KXORD,
     &            KYORD, XKNOT, YKNOT, BSCOEF)
C                                 Integrate over rectangle
C                                 [0.0,1.0] x [0.0,0.5]
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      A      = 0.0
      B      = 1.0
      C      = 0.5
      D      = 1.0
      VAL    = BS2IG(A,B,C,D,KXORD,KYORD,XKNOT,YKNOT,NXCOEF,NYCOEF,
     &         BSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print results
      WRITE (NOUT,99999) VAL, FI(A,B,C,D), FI(A,B,C,D) - VAL
99999 FORMAT (’ Computed Integral = ’, F10.5, /, ’ Exact Integral    ’
     &       , ’= ’, F10.5, /, ’ Error             ’
     &       , '= ', F10.6, /)
      END
```

Output

```
Computed Integral =    0.31250
Exact Integral    =    0.31250
Error             =   0.000000
```

BS3VL/DBS3VL (Single/Double precision)

Evaluate a three-dimensional tensor-product spline, given its tensor-product B-spline representation.

Usage

```
BS3VL(X, Y, Z, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
      NXCOEF, NYCOEF, NZCOEF, BSCOEF)
```

Arguments

X — X-coordinate of the point at which the spline is to be evaluated.   (Input)

Y — Y-coordinate of the point at which the spline is to be evaluated.   (Input)

Z — Z-coordinate of the point at which the spline is to be evaluated.   (Input)

KXORD — Order of the spline in the X-direction.   (Input)

KYORD — Order of the spline in the Y-direction.   (Input)

KZORD — Order of the spline in the Z-direction.   (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction.   (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction.   (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length NZCOEF + KZORD containing the knot sequence in the Z-direction.   (Input)
ZKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction.   (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction.   (Input)

NZCOEF — Number of B-spline coefficients in the Z-direction.   (Input)

BSCOEF — Array of length NXCOEF * NYCOEF * NZCOEF containing the tensor-product B-spline coefficients.   (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF by NZCOEF.

BS3VL — Value of the spline at (X, Y, Z).   (Output)

Comments

Automatic workspace usage is

`BS3VL` 3 `* MAX(KXORD, KYORD, KZORD) + KYORD * KZORD + KZORD` units, or

`DBS3VL` 6 `* MAX(KXORD, KYORD, KZORD) +` 2 `* KYORD * KZORD +` 2 `* KZORD` units.

Workspace may be explicitly provided, if desired, by use of `B23VL/DB23VL`. The reference is

```
CALL B23VL(X, Y, Z, KXORD, KYORD, KZORD, XKNOT, YKNOT,
          ZKNOT, NXCOEF, NYCOEF, NZCOEF, BSCOEF, WK)
```

The additional argument is

WK — Work array of length 3 `* MAX(KXORD, KYORD, KZORD) + KYORD * KZORD + KZORD`.

Algorithm

The function `BS3VL` evaluates a trivariate tensor-product spline (represented as a linear combination of tensor-product B-splines) at a given point. This routine is a special case of the IMSL routine `BS3DR` (page 491), which evaluates a partial derivative of such a spline. (The value of a spline is its zero-th derivative.) For more information, see de Boor (1978, pages 351−353).

This routine returns the value of the function *s* at a point (*x*, *y*, *z*) given the coefficients *c* by computing

$$s(x, y, z) = \sum_{l=1}^{N_z} \sum_{m=1}^{N_y} \sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}(x) B_{m,k_y,\mathbf{t}_y}(y) B_{l,k_z,\mathbf{t}_z}(z)$$

where k_x, k_y, and k_z are the orders of the splines. (These numbers are passed to the subroutine in `KXORD`, `KYORD`, and `KZORD`, respectively.) Likewise, $\mathbf{t}_x$, $\mathbf{t}_y$, and $\mathbf{t}_z$ are the corresponding knot sequences (`XKNOT`, `YKNOT`, and `ZKNOT`).

Example

For an example of the use of `BS3VL`, see IMSL routine `BS3IN` (page 464).

BS3DR/DBS3DR (Single/Double precision)

Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation.

Usage

```
BS3DR(IXDER, IYDER, IZDER, X, Y, Z, KXORD, KYORD, KZORD,
      XKNOT, YKNOT, ZKNOT, NXCOEF, NYCOEF, NZCOEF, BSCOEF)
```

Arguments

IXDER — Order of the X-derivative.   (Input)

IYDER — Order of the Y-derivative.   (Input)

IZDER — Order of the Z-derivative.   (Input)

X — X-coordinate of the point at which the spline is to be evaluated.   (Input)

Y — Y-coordinate of the point at which the spline is to be evaluated.   (Input)

Z — Z-coordinate of the point at which the spline is to be evaluated.   (Input)

KXORD — Order of the spline in the X-direction.   (Input)

KYORD — Order of the spline in the Y-direction.   (Input)

KZORD — Order of the spline in the Z-direction.   (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction.   (Input)
KNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction.   (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length NZCOEF + KZORD containing the knot sequence in the Z-direction.   (Input)
ZKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction.   (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction.   (Input)

NZCOEF — Number of B-spline coefficients in the Z-direction.   (Input)

BSCOEF — Array of length NXCOEF * NYCOEF * NZCOEF containing the tensor-product B-spline coefficients.   (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF by NZCOEF.

BS3DR — Value of the (IXDER, IYDER, IZDER) derivative of the spline at (X, Y, Z).   (Output)

Comments

1. Automatic workspace usage is

   BS3DR   3 * MAX(KXORD, KYORD, KZORD) + KYORD * KZORD + KZORD units, or

   DBS3DR 6 * MAX(KXORD, KYORD, KZORD) + 2 * KYORD * KZORD + 2 * KZORD units.

   Workspace may be explicitly provided, if desired, by use of B23DR/DB23DR. The reference is

```
CALL B23DR(IXDER, IYDER, IZDER, X, Y, Z, KXORD,
           KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
           NXCOEF, NYCOEF, NZCOEF, BSCOEF, WK)
```

The additional argument is

WK — Work array of length 3 * MAX0(KXORD, KYORD, KZORD) + KYORD * KZORD + KZORD.

2. Informational errors

Type	Code	
3	1	The point X does not satisfy XKNOT(KXORD) .LE. X .LE. XKNOT(NXCOEF + 1).
3	2	The point Y does not satisfy YKNOT(KYORD) .LE. Y .LE. YKNOT(NYCOEF + 1).
3	3	The point Z does not satisfy ZKNOT (KZORD) .LE. Z .LE. ZKNOT(NZCOEF + 1).

Algorithm

The function BS3DR evaluates a partial derivative of a trivariate tensor-product spline (represented as a linear combination of tensor-product B-splines) at a given point. For more information, see de Boor (1978, pages 351−353).

This routine returns the value of the function $s^{(p,\ q,\ r)}$ at a point (x, y, z) given the coefficients c by computing

$$s^{(p,q,r)}(x,y,z)=\sum_{l=1}^{N_z}\sum_{m=1}^{N_y}\sum_{n=1}^{N_x}c_{nml}\,B_{n,k_x,\mathbf{t}_x}^{(p)}(x)B_{m,k_y,\mathbf{t}_y}^{(q)}(y)B_{l,k_z,\mathbf{t}_z}^{(r)}(z)$$

where k_x, k_y, and k_z are the orders of the splines. (These numbers are passed to the subroutine in KXORD, KYORD, and KZORD, respectively.) Likewise, $\mathbf{t}_x$, $\mathbf{t}_y$, and $\mathbf{t}_z$ are the corresponding knot sequences (XKNOT, YKNOT, and ZKNOT).

Example

In this example, a spline interpolant s to a function $f(x, y, z) = x^4 + y(xz)^3$ is constructed using BS3IN (page 464). Next, BS3DR is used to compute $s^{(2,0,1)}(x, y, z)$. The values of this partial derivative and the error are computed on a 4 × 4 × 2 grid and then displayed.

```
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, KZORD, LDF, MDF, NXDATA, NXKNOT,
     &           NYDATA, NYKNOT, NZDATA, NZKNOT
      PARAMETER  (KXORD=5, KYORD=2, KZORD=3, NXDATA=21, NYDATA=6,
     &           NZDATA=8, LDF=NXDATA, MDF=NYDATA,
     &           NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD,
     &           NZKNOT=NZDATA+KZORD)
C
      INTEGER    I, J, K, L, NOUT, NXCOEF, NYCOEF, NZCOEF
      REAL       BS3DR, BSCOEF(NXDATA,NYDATA,NZDATA), F, F201,
     &           FDATA(LDF,MDF,NZDATA), FLOAT, S201, X, XDATA(NXDATA),
&           XKNOT(NXKNOT), Y, YDATA(NYDATA), YKNOT(NYKNOT), Z,
     &           ZDATA(NZDATA), ZKNOT(NZKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BS3DR, BS3IN, BSNAK, UMACH
C                                 Define function and (2,0,1)
C                                 derivative
      F(X,Y,Z)    = X*X*X*X + X*X*X*Y*Z*Z*Z
      F201(X,Y,Z) = 18.0*X*Y*Z
C                                 Set up X-interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Set up Y-interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/FLOAT(NYDATA-1)
   20 CONTINUE
C                                 Set up Z-interpolation points
      DO 30  I=1, NZDATA
         ZDATA(I) = FLOAT(I-1)/FLOAT(NZDATA-1)
   30 CONTINUE
C                                 Generate knots
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
      CALL BSNAK (NZDATA, ZDATA, KZORD, ZKNOT)
C                                 Generate FDATA
      DO 50  K=1, NZDATA
         DO 40  I=1, NYDATA
            DO 40  J=1, NXDATA
               FDATA(J,I,K) = F(XDATA(J),YDATA(I),ZDATA(K))
   40  CONTINUE
   50 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Interpolate
      CALL BS3IN (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA, FDATA,
     &            LDF, MDF, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
     &            BSCOEF)
C
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      NZCOEF = NZDATA
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print over a grid of
C                                 [-1.0,1.0] x [0.0,1.0] x [0.0,1.0]
C                                 at 32 points.
      DO 80  I=1, 4
         DO 70  J=1, 4
            DO 60  L=1, 2
               X    = 2.0*(FLOAT(I-1)/3.0) - 1.0
               Y    = FLOAT(J-1)/3.0
               Z    = FLOAT(L-1)
C                                 Evaluate spline
               S201 = BS3DR(2,0,1,X,Y,Z,KXORD,KYORD,KZORD,XKNOT,YKNOT,
     &                ZKNOT,NXCOEF,NYCOEF,NZCOEF,BSCOEF)
               WRITE (NOUT,'(3F12.4,2F12.6)') X, Y, Z, S201,
     &                F201(X,Y,Z) - S201
   60     CONTINUE
   70  CONTINUE
   80 CONTINUE
99999 FORMAT (38X, ’(2,0,1)’, /, 9X, ’X’, 11X,
     &       ’Y’, 11X, ’Z’, 4X, ’S     (X,Y,Z)    Error’)
      END
```

Output

```
                                  (2,0,1)
    X           Y           Z    S     (X,Y,Z)   Error
-1.0000      0.0000      0.0000   -0.000107    0.000107
-1.0000      0.0000      1.0000    0.000053   -0.000053
-1.0000      0.3333      0.0000    0.064051   -0.064051
-1.0000      0.3333      1.0000   -5.935941   -0.064059
-1.0000      0.6667      0.0000    0.127542   -0.127542
-1.0000      0.6667      1.0000  -11.873034   -0.126966
-1.0000      1.0000      0.0000    0.191166   -0.191166
-1.0000      1.0000      1.0000  -17.808527   -0.191473
-0.3333      0.0000      0.0000   -0.000002    0.000002
-0.3333      0.0000      1.0000    0.000000    0.000000
-0.3333      0.3333      0.0000    0.021228   -0.021228
-0.3333      0.3333      1.0000   -1.978768   -0.021232
-0.3333      0.6667      0.0000    0.042464   -0.042464
-0.3333      0.6667      1.0000   -3.957536   -0.042464
-0.3333      1.0000      0.0000    0.063700   -0.063700
-0.3333      1.0000      1.0000   -5.936305   -0.063694
 0.3333      0.0000      0.0000   -0.000003    0.000003
 0.3333      0.0000      1.0000    0.000000    0.000000
 0.3333      0.3333      0.0000   -0.021229    0.021229
 0.3333      0.3333      1.0000    1.978763    0.021238
 0.3333      0.6667      0.0000   -0.042465    0.042465
 0.3333      0.6667      1.0000    3.957539    0.042462
 0.3333      1.0000      0.0000   -0.063700    0.063700
 0.3333      1.0000      1.0000    5.936304    0.063697
 1.0000      0.0000      0.0000   -0.000098    0.000098
 1.0000      0.0000      1.0000    0.000053   -0.000053
 1.0000      0.3333      0.0000   -0.063855    0.063855
 1.0000      0.3333      1.0000    5.936146    0.063854
 1.0000      0.6667      0.0000   -0.127631    0.127631
 1.0000      0.6667      1.0000   11.873067    0.126933
 1.0000      1.0000      0.0000   -0.191442    0.191442
 1.0000      1.0000      1.0000   17.807940    0.192060
```

BS3GD/DBS3GD (Single/Double precision)

Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

Usage

```
CALL BS3GD (IXDER, IYDER, IZDER, NX, XVEC, NY, YVEC, NZ,
            ZVEC, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
            NXCOEF, NYCOEF, NZCOEF, BSCOEF, VALUE, LDVALU,
            MDVALU)
```

Arguments

IXDER — Order of the X-derivative.   (Input)

IYDER — Order of the Y-derivative. (Input)

IZDER — Order of the Z-derivative. (Input)

NX — Number of grid points in the *x*-direction. (Input)

XVEC — Array of length NX containing the *x*-coordinates at which the spline is to be evaluated. (Input)
The points in XVEC should be strictly increasing.

NY — Number of grid points in the *y*-direction. (Input)

YVEC — Array of length NY containing the *y*-coordinates at which the spline is to be evaluated. (Input)
The points in YVEC should be strictly increasing.

NZ — Number of grid points in the *z*-direction. (Input)

ZVEC — Array of length NY containing the *y*-coordinates at which the spline is to be evaluated. (Input)
The points in YVEC should be strictly increasing.

KXORD — Order of the spline in the *x*-direction. (Input)

KYORD — Order of the spline in the *y*-direction. (Input)

KZORD — Order of the spline in the *z*-direction. (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the *x*-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the *y*-direction. (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length NZCOEF + KZORD containing the knot sequence in the *z*-direction. (Input)
ZKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the *x*-direction. (Input)

NYCOEF — Number of B-spline coefficients in the *y*-direction. (Input)

NZCOEF — Number of B-spline coefficients in the *z*-direction. (Input)

BSCOEF — Array of length NXCOEF * NYCOEF * NZCOEF containing the tensor-product B-spline coefficients. (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF by NZCOEF.

VALUE — Array of size NX by NY by NZ containing the values of the (IXDER, IYDER, IZDER) derivative of the spline on the NX by NY by NZ grid. (Output)
VALUE(I, J, K) contains the derivative of the spline at the point
(XVEC(I), YVEC(J), ZVEC(K)).

LDVALU — Leading dimension of VALUE exactly as specified in the dimension statement of the calling program. (Input)

MDVALU — Middle dimension of VALUE exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   BS3GD KXORD * (IXDER + NX + 1) + KYORD * (IYDER + NY + 1) + KZORD * (IZDER + NZ + 1) + KXORD * KXORD + KYORD * KYORD + KZORD * KZORD + NZ + NY + NZ units, or + KZORD units, or

   DBS3GD 2 * (KXORD * (IXDER + NX + 1) + KYORD * (IYDER + NY + 1) + KZORD * (IZDER + NZ + 1) + KXORD * KXORD + KYORD * KYORD + KZORD * KZORD) + NX + NY + NZ units.

   Workspace may be explicitly provided, if desired, by use of B23GD/DB23GD. The reference is

   ```
   CALL B23GD (IXDER, IYDER, IZDER, X, Y, Z, KXORD,
               KYORD, KZORD,XKNOT, YKNOT, ZKNOT,
               NXCOEF, NYCOEF, NZCOEF, BSCOEF, VALUE,
               LDVALU, MDVALU, LEFTX, LEFTY, LEFTZ, A,
               B, C, DBIATX, DBIATY, DBIATZ, BX, BY,
               BZ)
   ```

   The additional arguments are as follows:

   LEFTX — Work array of length NX.

   LEFTY — Work array of length NY.

   LEFTZ — Work array of length NZ.

   A — Work array of length KXORD * KXORD.

   B — Work array of length KYORD * KYORD.

   C — Work array of length KZORD * KZORD.

   DBIATX — Work array of length KXORD * (IXDER + 1).

   DBIATY — Work array of length KYORD * (IYDER + 1).

   DBIATZ — Work array of length KZORD * (IZDER + 1).

   BX — Work array of length KXORD * NX.

   BY — Work array of length KYORD * NY.

   BZ — Work array of length KZORD * NZ.

2. Informational errors

Type	Code	
3	1	XVEC(I) does not satisfy XKNOT(KXORD) ≤ XVEC(I) ≤ XKNOT(NXCOEF + 1).
3	2	YVEC(I) does not satisfy YKNOT(KYORD) ≤ YVEC(I) ≤ YKNOT(NYCOEF + 1).

3	3	ZVEC(I) does not satisfy ZKNOT(KZORD) ≤ ZVEC(I) ≤ ZKNOT(NZCOEF + 1).
4	4	XVEC is not strictly increasing.
4	5	YVEC is not strictly increasing.
4	6	ZVEC is not strictly increasing.

Algorithm

The routine BS3GD evaluates a partial derivative of a trivariate tensor-product spline (represented as a linear combination of tensor-product B-splines) on a grid. For more information, see de Boor (1978, pages 351−353).

This routine returns the value of the function $s^{(p,q,r)}$ on the grid (x_i, y_j, z_k) for $i = 1, \ldots, nx$, $j = 1, \ldots, ny$, and $k = 1, \ldots, nz$ given the coefficients c by computing (for all (x, y, z) on the grid)

$$s^{(p,q,r)}(x, y, z) = \sum_{l=1}^{N_z}\sum_{m=1}^{N_y}\sum_{n=1}^{N_x} c_{nml} B_{n,k_x,\mathbf{t}_x}^{(p)}(x) B_{m,k_y,\mathbf{t}_y}^{(q)}(y) B_{l,k_z,\mathbf{t}_z}^{(r)}(z)$$

where k_x, k_y, and k_z are the orders of the splines. (These numbers are passed to the subroutine in KXORD, KYORD, and KZORD, respectively.) Likewise, $\mathbf{t}_x$, $\mathbf{t}_y$, and $\mathbf{t}_z$ are the corresponding knot sequences (XKNOT, YKNOT, and ZKNOT). The grid must be ordered in the sense that $x_i < x_{i+1}$, $y_j < y_{j+1}$, and $z_k < z_{k+1}$.

Example

In this example, a spline interpolant s to a function $f(x, y, z) = x^4 + y(xz)^3$ is constructed using BS3IN (page 464). Next, BS3GD is used to compute $s^{(2,0,1)}(x, y, z)$ on the grid. The values of this partial derivative and the error are computed on a 4 × 4 × 2 grid and then displayed.

```
      INTEGER    KXORD, KYORD, KZORD, LDF, LDVAL, MDF, MDVAL, NXDATA,
     &           NXKNOT, NYDATA, NYKNOT, NZ, NZDATA, NZKNOT
      PARAMETER  (KXORD=5, KYORD=2, KZORD=3, LDVAL=4, MDVAL=4,
     &           NXDATA=21, NYDATA=6, NZ=2, NZDATA=8, LDF=NXDATA,
     &           MDF=NYDATA, NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD,
     &           NZKNOT=NZDATA+KZORD)
C
      INTEGER    I, J, K, L, NOUT, NXCOEF, NYCOEF, NZCOEF
      REAL       BSCOEF(NXDATA,NYDATA,NZDATA), F, F201,
     &           FDATA(LDF,MDF,NZDATA), FLOAT, VALUE(LDVAL,MDVAL,NZ),
     &           X, XDATA(NXDATA), XKNOT(NXKNOT), XVEC(LDVAL), Y,
     &           YDATA(NYDATA), YKNOT(NYKNOT), YVEC(MDVAL), Z,
     &           ZDATA(NZDATA), ZKNOT(NZKNOT), ZVEC(NZ)
      INTRINSIC  FLOAT
      EXTERNAL   BS3GD, BS3IN, BSNAK, UMACH
C
C
C
      F(X,Y,Z)    = X*X*X*X + X*X*X*Y*Z*Z*Z
      F201(X,Y,Z) = 18.0*X*Y*Z
C
```

```
      CALL UMACH (2, NOUT)
C                                 Set up X interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NXDATA-1)) - 1.0
   10 CONTINUE
C                                 Set up Y interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/FLOAT(NYDATA-1)
   20 CONTINUE
C                                 Set up Z interpolation points
      DO 30  I=1, NZDATA
         ZDATA(I) = FLOAT(I-1)/FLOAT(NZDATA-1)
   30 CONTINUE
C                                 Generate knots
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
      CALL BSNAK (NZDATA, ZDATA, KZORD, ZKNOT)
C                                 Generate FDATA
      DO 50  K=1, NZDATA
         DO 40  I=1, NYDATA
            DO 40  J=1, NXDATA
               FDATA(J,I,K) = F(XDATA(J),YDATA(I),ZDATA(K))
   40  CONTINUE
   50 CONTINUE
C                                 Interpolate
      CALL BS3IN (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA, FDATA,
     &            LDF, MDF, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
     &            BSCOEF)
C
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      NZCOEF = NZDATA
C                                 Print over a grid of
C                                 [-1.0,1.0] x [0.0,1.0] x [0.0,1.0]
C                                 at 32 points.
      DO 60  I=1, 4
         XVEC(I) = 2.0*(FLOAT(I-1)/3.0) - 1.0
   60 CONTINUE
      DO 70  J=1, 4
         YVEC(J) = FLOAT(J-1)/3.0
   70 CONTINUE
      DO 80  L=1, 2
         ZVEC(L) = FLOAT(L-1)
   80 CONTINUE
      CALL BS3GD (2, 0, 1, 4, XVEC, 4, YVEC, 2, ZVEC, KXORD, KYORD,
     &            KZORD, XKNOT, YKNOT, ZKNOT, NXCOEF, NYCOEF, NZCOEF,
     &            BSCOEF, VALUE, LDVAL, MDVAL)
C
C
      WRITE (NOUT,99999)
      DO 110  I=1, 4
         DO 100  J=1, 4
            DO 90  L=1, 2
               WRITE (NOUT,’(5F13.4)’) XVEC(I), YVEC(J), ZVEC(L),
     &                               VALUE(I,J,L),
     &                               F201(XVEC(I),YVEC(J),ZVEC(L)) -
     &                               VALUE(I,J,L)
   90   CONTINUE
  100 CONTINUE
  110 CONTINUE
99999 FORMAT (44X, ’(2,0,1)’, /, 10X, ’X’, 11X, ’Y’, 10X, ’Z’, 10X,
     &       ’S     (X,Y,Z)  Error’)
      STOP
      END
```

Output

```
                                     (2,0,1)
    X           Y          Z        S     (X,Y,Z)  Error
-1.0000       0.0000       0.0000     -0.0003      0.0003
-1.0000       0.0000       1.0000      0.0001     -0.0001
-1.0000       0.3333       0.0000      0.0642     -0.0642
-1.0000       0.3333       1.0000     -5.9360     -0.0640
-1.0000       0.6667       0.0000      0.1273     -0.1273
-1.0000       0.6667       1.0000    -11.8730     -0.1270
-1.0000       1.0000       0.0000      0.1911     -0.1911
-1.0000       1.0000       1.0000    -17.8086     -0.1914
-0.3333       0.0000       0.0000      0.0000      0.0000
-0.3333       0.0000       1.0000      0.0000      0.0000
-0.3333       0.3333       0.0000      0.0212     -0.0212
-0.3333       0.3333       1.0000     -1.9788     -0.0212
-0.3333       0.6667       0.0000      0.0425     -0.0425
-0.3333       0.6667       1.0000     -3.9575     -0.0425
-0.3333       1.0000       0.0000      0.0637     -0.0637
-0.3333       1.0000       1.0000     -5.9363     -0.0637
 0.3333       0.0000       0.0000      0.0000      0.0000
 0.3333       0.0000       1.0000      0.0000      0.0000
 0.3333       0.3333       0.0000     -0.0212      0.0212
 0.3333       0.3333       1.0000      1.9788      0.0212
 0.3333       0.6667       0.0000     -0.0425      0.0425
 0.3333       0.6667       1.0000      3.9575      0.0425
 0.3333       1.0000       0.0000     -0.0637      0.0637
 0.3333       1.0000       1.0000      5.9363      0.0637
 1.0000       0.0000       0.0000     -0.0006      0.0006
 1.0000       0.0000       1.0000      0.0000      0.0000
 1.0000       0.3333       0.0000     -0.0636      0.0636
 1.0000       0.3333       1.0000      5.9359      0.0641
 1.0000       0.6667       0.0000     -0.1273      0.1273
 1.0000       0.6667       1.0000     11.8734      0.1266
 1.0000       1.0000       0.0000     -0.1910      0.1910
 1.0000       1.0000       1.0000     17.8098      0.1902
```

BS3IG/DBS3IG (Single/Double precision)

Evaluate the integral of a tensor-product spline in three dimensions over a three-dimensional rectangle, given its tensor-product B-spline representation.

Usage

```
BS3IG(A, B, C, D, E, F, KXORD, KYORD, KZORD, XKNOT, YKNOT,
      ZKNOT, NXCOEF, NYCOEF, NZCOEF, BSCOEF)
```

Arguments

A — Lower limit of the X-variable.   (Input)

B — Upper limit of the X-variable.   (Input)

C — Lower limit of the Y-variable.   (Input)

D — Upper limit of the Y-variable.   (Input)

E — Lower limit of the Z-variable.   (Input)

F — Upper limit of the Z-variable.   (Input)

KXORD — Order of the spline in the X-direction.   (Input)

KYORD — Order of the spline in the Y-direction.   (Input)

KZORD — Order of the spline in the Z-direction.   (Input)

XKNOT — Array of length NXCOEF + KXORD containing the knot sequence in the X-direction.   (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length NYCOEF + KYORD containing the knot sequence in the Y-direction.   (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length NZCOEF + KZORD containing the knot sequence in the Z-direction.   (Input)
ZKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction.   (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction.   (Input)

NZCOEF — Number of B-spline coefficients in the Z-direction.   (Input)

BSCOEF — Array of length NXCOEF * NYCOEF * NZCOEF containing the tensor-product B-spline coefficients.   (Input)
BSCOEF is treated internally as a matrix of size NXCOEF by NYCOEF by NZCOEF.

BS3IG — Integral of the spline over the three-dimensional rectangle (A, B) by (C, D) by (E, F).   (Output)

Comments

1. Automatic workspace usage is

   BS3IG   4 * (MAX(KXORD, KYORD, KZORD) + 1) + NYCOEF + NZCOEF units, or

   DBS3IG 8 * (MAX(KXORD, KYORD, KZORD) + 1) + 2 * NYCOEF + 2 * NZCOEF units.

   Workspace may be explicitly provided, if desired, by use of B23IG/DB23IG. The reference is

```
CALL B23IG(A, B, C, D, E, F, KXORD, KYORD, KZORD,
           XKNOT, YKNOT, ZKNOT, NXCOEF, NYCOEF,
           NZCOEF, BSCOEF, WK)
```

The additional argument is

WK — Work array of length 4 * (MAX(KXORD, KYORD, KZORD) + 1) + NYCOEF + NZCOEF.

2. Informational errors

Type	Code	
3	1	The lower limit of the X-integration is less than XKNOT(KXORD).
3	2	The upper limit of the X-integration is greater than XKNOT(NXCOEF + 1).
3	3	The lower limit of the Y-integration is less than YKNOT(KYORD).
3	4	The upper limit of the Y-integration is greater than YKNOT(NYCOEF + 1).
3	5	The lower limit of the Z- integration is less than ZKNOT(KZORD).
3	6	The upper limit of the Z-integration is greater than ZKNOT(NZCOEF + 1).
4	13	Multiplicity of the knots cannot exceed the order of the spline.
4	14	The knots must be nondecreasing.

Algorithm

The routine BS3IG computes the integral of a tensor-product three-dimensional spline, given its B-spline representation. Specifically, given the knot sequence $\mathbf{t}_x$ = XKNOT, $\mathbf{t}_y$ = YKNOT, $\mathbf{t}_z$ = ZKNOT, the order k_x = KXORD, k_y = KYORD, k_z = KZORD, the coefficients β = BSCOEF, the number of coefficients n_x = NXCOEF, n_y = NYCOEF, n_z = NZCOEF, and a three-dimensional rectangle [a, b] by [c, d] by [e, f], BS3IG returns the value

$$\int_a^b\int_c^d\int_e^f \sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\sum_{m=1}^{n_z} \beta_{ijm} B_{ijm}\ dz\ dy\ dx$$

where

$$B_{ijm}(x, y, z) = B_{i,k_x,\mathbf{t}_x}(x) B_{j,k_y,\mathbf{t}_y}(y) B_{m,k_z\mathbf{t}_z}(z)$$

This routine uses the identity (22) on page 151 of de Boor (1978). It assumes (for all knot sequences) that the first and last k knots are stacked, that is, $\mathbf{t}_1 = \ldots = \mathbf{t}_k$ and $\mathbf{t}_{n+1} = \ldots = \mathbf{t}_{n+k}$, where k is the order of the spline in the x, y, or z direction.

Example

We integrate the three-dimensional tensor-product quartic (k_x = 5) by linear (k_y = 2) by quadratic (k_z = 3) spline which interpolates $x^3 + xyz$ at the points

$$\left\{(i/10,\ j/5,\ m/7) : i = -10, \ldots, 10,\ j = 0, \ldots, 5,\ \text{and}\ m = 0, \ldots, 7\right\}$$

over the rectangle [0, 1] × [.5, 1] × [0, .5]. The exact answer is 11/128.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    KXORD, KYORD, KZORD, LDF, MDF, NXDATA, NXKNOT,
     &           NYDATA, NYKNOT, NZDATA, NZKNOT
      PARAMETER  (KXORD=5, KYORD=2, KZORD=3, NXDATA=21, NYDATA=6,
     &           NZDATA=8, LDF=NXDATA, MDF=NYDATA,
     &           NXKNOT=NXDATA+KXORD, NYKNOT=NYDATA+KYORD,
     &           NZKNOT=NZDATA+KZORD)
C
      INTEGER    I, J, K, NOUT, NXCOEF, NYCOEF, NZCOEF
      REAL       A, B, BS3IG, BSCOEF(NXDATA,NYDATA,NZDATA), C, D, E,
     &           F, FDATA(LDF,MDF,NZDATA), FF, FIG, FLOAT, G, H, RI,
     &           RJ, VAL, X, XDATA(NXDATA), XKNOT(NXKNOT), Y,
     &           YDATA(NYDATA), YKNOT(NYKNOT), Z, ZDATA(NZDATA),
     &           ZKNOT(NZKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BS3IG, BS3IN, BSNAK, UMACH
C                                 Define function
      F(X,Y,Z) = X*X*X + X*Y*Z
C                                 Set up interpolation points
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-11)/10.0
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NXDATA, XDATA, KXORD, XKNOT)
C                                 Set up interpolation points
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/FLOAT(NYDATA-1)
   20 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NYDATA, YDATA, KYORD, YKNOT)
C                                 Set up interpolation points
      DO 30  I=1, NZDATA
         ZDATA(I) = FLOAT(I-1)/FLOAT(NZDATA-1)
   30 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NZDATA, ZDATA, KZORD, ZKNOT)
C                                 Generate FDATA
      DO 50  K=1, NZDATA
         DO 40  I=1, NYDATA
            DO 40  J=1, NXDATA
               FDATA(J,I,K) = F(XDATA(J),YDATA(I),ZDATA(K))
   40  CONTINUE
   50 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Interpolate
      CALL BS3IN (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA, FDATA,
     &            LDF, MDF, KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT,
     &            BSCOEF)
C
      NXCOEF = NXDATA
      NYCOEF = NYDATA
      NZCOEF = NZDATA
      A      = 0.0
      B      = 1.0
      C      = 0.5
      D      = 1.0
      E      = 0.0
      FF     = 0.5
C                                  Integrate
      VAL    = BS3IG(A,B,C,D,E,FF,KXORD,KYORD,KZORD,XKNOT,YKNOT,ZKNOT,
     &          NXCOEF,NYCOEF,NZCOEF,BSCOEF)
C                                  Calculate integral directly
      G   = .5*(B**4-A**4)
      H   = (B-A)*(B+A)
      RI  = G*(D-C)
      RJ  = .5*H*(D-C)*(D+C)
      FIG = .5*(RI*(FF-E)+.5*RJ*(FF-E)*(FF+E))
C                                  Print results
      WRITE (NOUT,99999) VAL, FIG, FIG - VAL
99999 FORMAT (' Computed Integral = ', F10.5, /, ' Exact Integral    '
     &       , '= ', F10.5,/, ' Error             '
     &       , '= ', F10.6, /)
      END
```

Output

```
Computed Integral =    0.08594
Exact Integral    =    0.08594
Error             =   0.000000
```

BSCPP/DBSCPP (Single/Double precision)

Convert a spline in B-spline representation to piecewise polynomial representation.

Usage

```
CALL BSCPP (KORDER, XKNOT, NCOEF, BSCOEF, NPPCF, BREAK,
            PPCOEF)
```

Arguments

KORDER — Order of the spline.   (Input)

XKNOT — Array of length KORDER + NCOEF containing the knot sequence. (Input)
XKNOT must be nondecreasing.

NCOEF — Number of B-spline coefficients.   (Input)

BSCOEF — Array of length NCOEF containing the B-spline coefficients.   (Input)

NPPCF — Number of piecewise polynomial pieces.   (Output)
NPPCF is always less than or equal to NCOEF − KORDER + 1.

BREAK — Array of length (NPPCF + 1) containing the breakpoints of the piecewise polynomial representation.   (Output)
BREAK must be dimensioned at least NCOEF − KORDER + 2.

PPCOEF — Array of length `KORDER * NPPCF` containing the local coefficients of the polynomial pieces. (Output)
`PPCOEF` is treated internally as a matrix of size `KORDER` by `NPPCF`.

Comments

1. Automatic workspace usage is

   `BSCPP` (`KORDER` + 3) `*` `KORDER` units, or
   `DBSCPP` 2 `*` (`KORDER` + 3) `*` `KORDER` units.

   Workspace may be explicitly provided, if desired, by use of `B2CPP`/`DB2CPP`. The reference is

   ```
   CALL B2CPP (KORDER, XKNOT, NCOEF, BSCOEFF, NPPCF,
               BREAK, PPCOEF, WK)
   ```

   The additional argument is

   WK — Work array of length (`KORDER` + 3) `*` `KORDER`.

2. Informational errors

Type	Code	
4	4	Multiplicity of the knots cannot exceed the order of the spline.
4	5	The knots must be nondecreasing.

Algorithm

The routine `BSCPP` is based on the routine `BSPLPP` by de Boor (1978, page 140). This routine is used to convert a spline in B-spline representation to a piecewise polynomial (pp) representation which can then be evaluated more efficiently. There is some overhead in converting from the B-spline representation to the pp representation, but the conversion to pp form is recommended when 3 or more function values are needed per polynomial piece.

Example

For an example of the use of `BSCPP`, see `PPDER` (page 507).

PPVAL/DPPVAL (Single/Double precision)

Evaluate a piecewise polynomial.

Usage

```
PPVAL(X, KORDER, NINTV, BREAK, PPCOEF)
```

Arguments

X — Point at which the polynomial is to be evaluated. (Input)

KORDER — Order of the polynomial. (Input)

NINTV — Number of polynomial pieces. (Input)

BREAK — Array of length NINTV + 1 containing the breakpoints of the piecewise polynomial representation. (Input)
BREAK must be strictly increasing.

PPCOEF — Array of size KORDER * NINTV containing the local coefficients of the piecewise polynomial pieces. (Input)
PPCOEF is treated internally as a matrix of size KORDER by NINTV.

PPVAL — Value of the piecewise polynomial at X. (Output)

Algorithm

The routine PPVAL evaluates a piecewise polynomial at a given point. This routine is a special case of the routine PPDER (page 507), which evaluates the derivative of a piecewise polynomial. (The value of a piecewise polynomial is its zero-th derivative.)

The routine PPDER is based on the routine PPVALU in de Boor (1978, page 89).

Example

In this example, a spline interpolant to a function *f* is computed using the IMSL routine BSINT (page 450). This routine represents the interpolant as a linear combination of B-splines. This representation is then converted to piecewise polynomial representation by calling the IMSL routine BSCPP (page 504). The piecewise polynomial is evaluated using PPVAL. These values are compared to the corresponding values of *f*.

```
      INTEGER    KORDER, NCOEF, NDATA, NKNOT
      PARAMETER  (KORDER=4, NCOEF=20, NDATA=20, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NOUT, NPPCF
      REAL       BREAK(NCOEF), BSCOEF(NCOEF), EXP, F, FDATA(NDATA),
     &           FLOAT, PPCOEF(KORDER,NCOEF), PPVAL, S, X,
     &           XDATA(NDATA), XKNOT(NKNOT)
      INTRINSIC  EXP, FLOAT
      EXTERNAL   BSCPP, BSINT, PPVAL, UMACH
C                                 Define function
      F(X) = X*EXP(X)
C                                 Set up interpolation points
      DO 30  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   30 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Compute the B-spline interpolant
      CALL BSINT (NCOEF, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                 Convert to piecewise polynomial
      CALL BSCPP (KORDER, XKNOT, NCOEF, BSCOEF, NPPCF, BREAK, PPCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
```

```
C                                  Print the interpolant on a uniform
C                                  grid
      DO 40  I=1, NDATA
         X = FLOAT(I-1)/FLOAT(NDATA-1)
C                                  Compute value of the piecewise
C                                  polynomial
         S = PPVAL(X,KORDER,NPPCF,BREAK,PPCOEF)
         WRITE (NOUT,’(2F12.3, E14.3)’) X, S, F(X) - S
   40 CONTINUE
99999 FORMAT (11X, ’X’, 8X, ’S(X)’, 7X, ’Error’)
      END
```

Output

```
    X         S(X)       Error
0.000       0.000     0.000E+00
0.053       0.055    -0.745E-08
0.105       0.117     0.000E+00
0.158       0.185     0.000E+00
0.211       0.260    -0.298E-07
0.263       0.342     0.298E-07
0.316       0.433     0.000E+00
0.368       0.533     0.000E+00
0.421       0.642     0.000E+00
0.474       0.761     0.596E-07
0.526       0.891     0.000E+00
0.579       1.033     0.000E+00
0.632       1.188     0.000E+00
0.684       1.356     0.000E+00
0.737       1.540    -0.119E-06
0.789       1.739     0.000E+00
0.842       1.955     0.000E+00
0.895       2.189     0.238E-06
0.947       2.443     0.238E-06
1.000       2.718     0.238E-06
```

PPDER/DPPDER (Single/Double precision)

Evaluate the derivative of a piecewise polynomial.

Usage

`PPDER(IDERIV, X, KORDER, NINTV, BREAK, PPCOEF)`

Arguments

IDERIV — Order of the derivative to be evaluated. (Input)
In particular, `IDERIV` = 0 returns the value of the polynomial.

X — Point at which the polynomial is to be evaluated. (Input)

KORDER — Order of the polynomial. (Input)

NINTV — Number of polynomial pieces. (Input)

BREAK — Array of length `NINTV` + 1 containing the breakpoints of the piecewise polynomial representation. (Input)
`BREAK` must be strictly increasing.

PPCOEF — Array of size `KORDER * NINTV` containing the local coefficients of the piecewise polynomial pieces. (Input)
`PPCOEF` is treated internally as a matrix of size `KORDER` by `NINTV`.

PPDER — Value of the `IDERIV`-th derivative of the piecewise polynomial at `X`. (Output)

Algorithm

The routine `PPDER` evaluates the derivative of a piecewise polynomial function *f* at a given point. This routine is based on the subroutine `PPVALU` by de Boor (1978, page 89). In particular, if the breakpoint sequence is stored in ξ (a vector of length *N* = `NINTV` + 1), and if the coefficients of the piecewise polynomial representation are stored in **c**, then the value of the *j*-th derivative of *f* at *x* in $[\xi_i, \xi_{i+1})$ is

$$f^{(j)}(x) = \sum_{m=j}^{k-1} c_{m+1,i} \frac{(x-\xi_i)^{m-j}}{(m-j)!}$$

when $j = 0$ to $k - 1$ and zero otherwise. Notice that this representation forces the function to be right continuous. If *x* is less than ξ_1, then *i* is set to 1 in the above formula; if *x* is greater than or equal to ξ_N, then *i* is set to $N - 1$. This has the effect of extending the piecewise polynomial representation to the real axis by extrapolation of the first and last pieces.

Example

In this example, a spline interpolant to a function *f* is computed using the IMSL routine `BSINT` (page 450). This routine represents the interpolant as a linear combination of B-splines. This representation is then converted to piecewise polynomial representation by calling the IMSL routine `BSCPP` (page 504). The piecewise polynomial's zero-th and first derivative are evaluated using `PPDER`. These values are compared to the corresponding values of *f*.

```
      INTEGER    KORDER, NCOEF, NDATA, NKNOT
      PARAMETER  (KORDER=4, NCOEF=20, NDATA=20, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NOUT, NPPCF
      REAL       BREAK(NCOEF), BSCOEF(NCOEF), DF, DS, EXP, F,
     &           FDATA(NDATA), FLOAT, PPCOEF(KORDER,NCOEF), PPDER, S,
     &           X, XDATA(NDATA), XKNOT(NKNOT)
      INTRINSIC  EXP, FLOAT
      EXTERNAL   BSCPP, BSINT, BSNAK, PPDER, UMACH
C
      F(X)  = X*EXP(X)
      DF(X) = (X+1.)*EXP(X)
C                                 Set up interpolation points
      DO 10  I=1, NDATA
```

```
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                 Compute the B-spline interpolant
      CALL BSINT (NCOEF, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                 Convert to piecewise polynomial
      CALL BSCPP (KORDER, XKNOT, NCOEF, BSCOEF, NPPCF, BREAK, PPCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print the interpolant on a uniform
C                                 grid
      DO 20  I=1, NDATA
         X = FLOAT(I-1)/FLOAT(NDATA-1)
C                                 Compute value of the piecewise
C                                 polynomial
         S = PPDER(0,X,KORDER,NPPCF,BREAK,PPCOEF)
C                                 Compute derivative of the piecewise
C                                 polynomial
         DS = PPDER(1,X,KORDER,NPPCF,BREAK,PPCOEF)
         WRITE (NOUT,’(2F12.3,F12.6,F12.3,F12.6)’) X, S, F(X) - S, DS,
     &                                            DF(X) - DS
   20 CONTINUE
99999 FORMAT (11X, ’X’, 8X, ’S(X)’, 7X, ’Error’, 7X, ’S’’(X)’, 7X,
     &       ’Error’)
      END
```

Output

```
    X        S(X)       Error       S’(X)       Error
0.000       0.000    0.000000       1.000   -0.000112
0.053       0.055    0.000000       1.109    0.000030
0.105       0.117    0.000000       1.228   -0.000008
0.158       0.185    0.000000       1.356    0.000002
0.211       0.260    0.000000       1.494    0.000000
0.263       0.342    0.000000       1.643    0.000000
0.316       0.433    0.000000       1.804   -0.000001
0.368       0.533    0.000000       1.978    0.000002
0.421       0.642    0.000000       2.165    0.000001
0.474       0.761    0.000000       2.367    0.000000
0.526       0.891    0.000000       2.584   -0.000001
0.579       1.033    0.000000       2.817    0.000001
0.632       1.188    0.000000       3.068    0.000001
0.684       1.356    0.000000       3.338    0.000001
0.737       1.540    0.000000       3.629    0.000001
0.789       1.739    0.000000       3.941    0.000000
0.842       1.955    0.000000       4.276   -0.000006
0.895       2.189    0.000000       4.636    0.000024
0.947       2.443    0.000000       5.022   -0.000090
1.000       2.718    0.000000       5.436    0.000341
```

PP1GD/DPP1GD (Single/Double precision)

Evaluate the derivative of a piecewise polynomial on a grid.

Usage

```
CALL PP1GD (IDERIV, N, XVEC, KORDER, NINTV, BREAK, PPCOEF,
            VALUE)
```

Arguments

IDERIV — Order of the derivative to be evaluated. (Input)
In particular, IDERIV = 0 returns the values of the piecewise polynomial.

N — Length of vector XVEC. (Input)

XVEC — Array of length N containing the points at which the piecewise polynomial is to be evaluated. (Input)
The points in XVEC should be strictly increasing.

KORDER — Order of the polynomial. (Input)

NINTV — Number of polynomial pieces. (Input)

BREAK — Array of length NINTV + 1 containing the breakpoints for the piecewise polynomial representation. (Input)
BREAK must be strictly increasing.

PPCOEF — Matrix of size KORDER by NINTV containing the local coefficients of the polynomial pieces. (Input)

VALUE — Array of length N containing the values of the IDERIV-th derivative of the piecewise polynomial at the points in XVEC. (Output)

Comments

1. Automatic workspace usage is

   PP1GD 3 * N units, or

   DPP1GD 5 * N units.

   Workspace may be explicitly provided, if desired, by use of P21GD/DP21GD. The reference is

   ```
   CALL P21GD (IDERIV, N, XVEC, KORDER, NINTV, BREAK,
               PPCOEF, VALUE, IWK, WORK1, WORK2)
   ```

   The additional arguments are as follows:

   IWK — Array of length N.

   WORK1 — Array of length N.

   WORK2 — Array of length N.

2. Informational error

Type	Code	
4	4	The points in XVEC must be strictly increasing.

Algorithm

The routine PP1GD evaluates a piecewise polynomial function *f* (or its derivative) at a vector of points. That is, given a vector *x* of length *n* satisfying $x_i < x_{i+1}$ for $i = 1, \ldots, n - 1$, a derivative value *j*, and a piecewise polynomial function *f* that is represented by a breakpoint sequence and coefficient matrix this routine returns the values

$$f^{(j)}\left(x_i\right) \quad i = 1, \ldots, n$$

in the array VALUE. The functionality of this routine is the same as that of PPDER (page 507) called in a loop, however PP1GD is much more efficient.

Example

To illustrate the use of PP1GD, we modify the example program for PPDER (page 507). In this example, a piecewise polynomial interpolant to *F* is computed. The values of this polynomial are then compared with the exact function values. The routine PP1GD is based on the routine PPVALU in de Boor (1978, page 89).

```
      INTEGER    KORDER, N, NCOEF, NDATA, NKNOT
      PARAMETER  (KORDER=4, N=20, NCOEF=20, NDATA=20,
     &           NKNOT=NDATA+KORDER)
C
      INTEGER    I, NINTV, NOUT, NPPCF
      REAL       BREAK(NCOEF), BSCOEF(NCOEF), DF, EXP, F,
     &           FDATA(NDATA), FLOAT, PPCOEF(KORDER,NCOEF), VALUE1(N),
     &           VALUE2(N), X, XDATA(NDATA), XKNOT(NKNOT), XVEC(N)
      INTRINSIC  EXP, FLOAT
      EXTERNAL   BSCPP, BSINT, BSNAK, PP1GD, UMACH
C
      F(X)  = X*EXP(X)
      DF(X) = (X+1.)*EXP(X)
C                                  Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                  Compute the B-spline interpolant
      CALL BSINT (NCOEF, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                  Convert to piecewise polynomial
      CALL BSCPP (KORDER, XKNOT, NCOEF, BSCOEF, NPPCF, BREAK, PPCOEF)
C                                  Compute evaluation points
      DO 20  I=1, N
         XVEC(I) = FLOAT(I-1)/FLOAT(N-1)
   20 CONTINUE
C                                  Compute values of the piecewise
C                                  polynomial
```

```
      NINTV = NPPCF
      CALL PP1GD (0, N, XVEC, KORDER, NINTV, BREAK, PPCOEF, VALUE1)
C                                 Compute the values of the first
C                                 derivative of the piecewise
C                                 polynomial
      CALL PP1GD (1, N, XVEC, KORDER, NINTV, BREAK, PPCOEF, VALUE2)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99998)
C                                 Print the results on a uniform
C                                 grid
      DO 30  I=1, N
         WRITE (NOUT,99999) XVEC(I), VALUE1(I), F(XVEC(I)) - VALUE1(I)
     &                     , VALUE2(I), DF(XVEC(I)) - VALUE2(I)
   30 CONTINUE
99998 FORMAT (11X, ’X’, 8X, ’S(X)’, 7X, ’Error’, 7X, ’S’’(X)’, 7X,
     &       ’Error’)
99999 FORMAT (’ ’, 2F12.3, F12.6, F12.3, F12.6)
      END
```

Output

```
   X        S(X)       Error       S’(X)       Error
0.000       0.000    0.000000       1.000   -0.000112
0.053       0.055    0.000000       1.109    0.000030
0.105       0.117    0.000000       1.228   -0.000008
0.158       0.185    0.000000       1.356    0.000002
0.211       0.260    0.000000       1.494    0.000000
0.263       0.342    0.000000       1.643    0.000000
0.316       0.433    0.000000       1.804   -0.000001
0.368       0.533    0.000000       1.978    0.000002
0.421       0.642    0.000000       2.165    0.000001
0.474       0.761    0.000000       2.367    0.000000
0.526       0.891    0.000000       2.584   -0.000001
0.579       1.033    0.000000       2.817    0.000001
0.632       1.188    0.000000       3.068    0.000001
0.684       1.356    0.000000       3.338    0.000001
0.737       1.540    0.000000       3.629    0.000001
0.789       1.739    0.000000       3.941    0.000000
0.842       1.955    0.000000       4.276   -0.000006
0.895       2.189    0.000000       4.636    0.000024
0.947       2.443    0.000000       5.022   -0.000090
1.000       2.718    0.000000       5.436    0.000341
```

PPITG/DPPITG (Single/Double precision)

Evaluate the integral of a piecewise polynomial.

Usage

`PPITG(A, B, KORDER, NINTV, BREAK, PPCOEF)`

Arguments

A — Lower limit of integration.   (Input)

B — Upper limit of integration.   (Input)

KORDER — Order of the polynomial.   (Input)

NINTV — Number of piecewise polynomial pieces.   (Input)

BREAK — Array of length NINTV + 1 containing the breakpoints for the piecewise polynomial.   (Input)
BREAK must be strictly increasing.

PPCOEF — Array of size KORDER * NINTV containing the local coefficients of the piecewise polynomial pieces.   (Input)
PPCOEF is treated internally as a matrix of size KORDER by NINTV.

PPITG — Value of the integral from A to B of the piecewise polynomial. (Output)

Algorithm

The routine PPITG evaluates the integral of a piecewise polynomial over an interval.

Example

In this example, we compute a quadratic spline interpolant to the function x^2 using the IMSL routine BSINT (page 450). We then evaluate the integral of the spline interpolant over the intervals [0, 1/2] and [0, 2]. The interpolant reproduces x^2, and hence, the values of the integrals are 1/24 and 8/3, respectively.

```
      INTEGER    KORDER, NDATA, NKNOT
      PARAMETER  (KORDER=3, NDATA=10, NKNOT=NDATA+KORDER)
C
      INTEGER    I, NOUT, NPPCF
      REAL       A, B, BREAK(NDATA), BSCOEF(NDATA), EXACT, F,
     &           FDATA(NDATA), FI, FLOAT, PPCOEF(KORDER,NDATA), PPITG,
     &           VALUE, X, XDATA(NDATA), XKNOT(NKNOT)
      INTRINSIC  FLOAT
      EXTERNAL   BSCPP, BSINT, BSNAK, PPITG, UMACH
C
      F(X)  = X*X
      FI(X) = X*X*X/3.0
C                                Set up interpolation points
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA-1)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                Generate knot sequence
      CALL BSNAK (NDATA, XDATA, KORDER, XKNOT)
C                                Interpolate
      CALL BSINT (NDATA, XDATA, FDATA, KORDER, XKNOT, BSCOEF)
C                                Convert to piecewise polynomial
      CALL BSCPP (KORDER, XKNOT, NDATA, BSCOEF, NPPCF, BREAK, PPCOEF)
C                                Compute the integral of F over
C                                [0.0,0.5]
      A     = 0.0
      B     = 0.5
      VALUE = PPITG(A,B,KORDER,NPPCF,BREAK,PPCOEF)
      EXACT = FI(B) - FI(A)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print the result
      WRITE (NOUT,99999) A, B, VALUE, EXACT, EXACT - VALUE
C                                 Compute the integral of F over
C                                 [0.0,2.0]
      A     = 0.0
      B     = 2.0
      VALUE = PPITG(A,B,KORDER,NPPCF,BREAK,PPCOEF)
      EXACT = FI(B) - FI(A)
C                                 Print the result
      WRITE (NOUT,99999) A, B, VALUE, EXACT, EXACT - VALUE
99999 FORMAT (' On the closed interval (', F3.1, ',', F3.1,
     &       ') we have :', /, 1X, 'Computed Integral = ', F10.5, /,
     &       1X, 'Exact Integral    = ', F10.5, /, 1X, 'Error          '
     &       , '    = ', F10.6, /, /)
C
      END
```

Output

```
On the closed interval (0.0,0.5) we have :
Computed Integral =    0.04167
Exact Integral    =    0.04167
Error             =   0.000000

On the closed interval (0.0,2.0) we have :
Computed Integral =    2.66667
Exact Integral    =    2.66667
Error             =   0.000001
```

QDVAL/DQDVAL (Single/Double precision)

Evaluate a function defined on a set of points using quadratic interpolation.

Usage

QDVAL(X, NDATA, XDATA, FDATA, CHECK)

Arguments

X — Coordinate of the point at which the function is to be evaluated. (Input)

NDATA — Number of data points. (Input)
NDATA must be at least 3.

XDATA — Array of length NDATA containing the location of the data points. (Input) XDATA must be strictly increasing.

FDATA — Array of length NDATA containing the function values. (Input)
FDATA(I) is the value of the function at XDATA(I).

CHECK — Logical variable that is `.TRUE.` if checking of `XDATA` is required or `.FALSE.` if checking is not required. (Input)

QDVAL — Value of the quadratic interpolant at `X`. (Output)

Comments

Informational error

Type	Code	
4	3	The `XDATA` values must be strictly increasing.

Algorithm

The function `QDVAL` interpolates a table of values, using quadratic polynomials, returning an approximation to the tabulated function. Let (x_i, f_i) for $i = 1, \ldots, n$ be the tabular data. Given a number x at which an interpolated value is desired, we first find the nearest interior grid point x_i. A quadratic interpolant q is then formed using the three points (x_{i-1}, f_{i-1}), (x_i, f_i), and (x_{i+1}, f_{i+1}). The number returned by `QDVAL` is $q(x)$. One should note that, in general, the function `QDVAL` is discontinuous at the midpoints of the mesh.

Example

In this example, the value of sin x is approximated at $\pi/4$ by using `QDVAL` on a table of 33 equally spaced values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=33)
C
      INTEGER    I, NOUT
      REAL       CONST, F, FDATA(NDATA), H, PI, QDVAL, QT, SIN, X,
     &           XDATA(NDATA)
      LOGICAL    CHECK
      INTRINSIC  SIN
      EXTERNAL   CONST, QDVAL, UMACH
C                                 Define function
      F(X) = SIN(X)
C                                 Generate data points
      XDATA(1) = 0.0
      FDATA(1) = F(XDATA(1))
      H        = 1.0/32.0
      DO 10  I=2, NDATA
         XDATA(I) = XDATA(I-1) + H
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Get value of PI and set X
      PI = CONST('PI')
      X  = PI/4.0
C                                 Check XDATA
      CHECK = .TRUE.
C                                 Evaluate at PI/4
      QT = QDVAL(X,NDATA,XDATA,FDATA,CHECK)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print results
      WRITE (NOUT,99999) X, F(X), QT, (F(X)-QT)
```

```
C
99999 FORMAT (15X, 'X', 6X, 'F(X)', 6X, 'QDVAL', 5X, 'ERROR', //, 6X,
     &       4F10.3, /)
      END
```

Output

```
    X      F(X)      QDVAL     ERROR

0.785     0.707     0.707     0.000
```

QDDER/DQDDER (Single/Double precision)

Evaluate the derivative of a function defined on a set of points using quadratic interpolation.

Usage

`QDDER(IDERIV, X, NDATA, XDATA, FDATA, CHECK)`

Arguments

IDERIV — Order of the derivative. (Input)

X — Coordinate of the point at which the function is to be evaluated. (Input)

NDATA — Number of data points. (Input)
`NDATA` must be at least three.

XDATA — Array of length `NDATA` containing the location of the data points. (Input) `XDATA` must be strictly increasing.

FDATA — Array of length `NDATA` containing the function values. (Input)
`FDATA`(`I`) is the value of the function at `XDATA`(`I`).

CHECK — Logical variable that is `.TRUE.` if checking of `XDATA` is required or `.FALSE.` if checking is not required. (Input)

QDDER — Value of the `IDERIV`-th derivative of the quadratic interpolant at `X`. (Output)

Comments

1. Informational error

Type	Code	
4	3	The `XDATA` values must be strictly increasing.

2. Because quadratic interpolation is used, if the order of the derivative is greater than two, then the returned value is zero.

Algorithm

The function `QDDER` interpolates a table of values, using quadratic polynomials, returning an approximation to the derivative of the tabulated function. Let (x_i, f_i)

for $i = 1, \ldots, n$ be the tabular data. Given a number x at which an interpolated value is desired, we first find the nearest interior grid point x_i. A quadratic interpolant q is then formed using the three points (x_{i-1}, f_{i-1}), (x_i, f_i), and (x_{i+1}, f_{i+1}). The number returned by QDDER is $q^{(j)}(x)$, where j = IDERIV. One should note that, in general, the function QDDER is discontinuous at the midpoints of the mesh.

Example

In this example, the value of sin x and its derivatives are approximated at π/4 by using QDDER on a table of 33 equally spaced values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=33)
C
      INTEGER    I, IDERIV, NOUT
      REAL       CONST, COS, F, F1, F2, FDATA(NDATA), H, PI,
     &           QDDER, QT, SIN, X, XDATA(NDATA)
      LOGICAL    CHECK
      INTRINSIC  COS, SIN
      EXTERNAL   CONST, QDDER, UMACH
C                                 Define function and derivatives
      F(X)  = SIN(X)
      F1(X) = COS(X)
      F2(X) = -SIN(X)
C                                 Generate data points
      XDATA(1) = 0.0
      FDATA(1) = F(XDATA(1))
      H        = 1.0/32.0
      DO 10  I=2, NDATA
         XDATA(I) = XDATA(I-1) + H
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Get value of PI and set X
      PI = CONST('PI')
      X  = PI/4.0
C                                 Check XDATA
      CHECK = .TRUE.
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99998)
C                                 Evaluate quadratic at PI/4
      IDERIV = 0
      QT     = QDDER(IDERIV,X,NDATA,XDATA,FDATA,CHECK)
      WRITE (NOUT,99999) X, IDERIV, F(X), QT, (F(X)-QT)
      CHECK = .FALSE.
C                                 Evaluate first derivative at PI/4
      IDERIV = 1
      QT     = QDDER(IDERIV,X,NDATA,XDATA,FDATA,CHECK)
      WRITE (NOUT,99999) X, IDERIV, F1(X), QT, (F1(X)-QT)
C                                 Evaluate second derivative at PI/4
      IDERIV = 2
      QT     = QDDER(IDERIV,X,NDATA,XDATA,FDATA,CHECK)
      WRITE (NOUT,99999) X, IDERIV, F2(X), QT, (F2(X)-QT)
C
```

```
99998 FORMAT (33X, 'IDER', /, 15X, 'X', 6X, 'IDER', 6X, 'F    (X)',
     &       5X, 'QDDER', 6X, 'ERROR', //)
99999 FORMAT (7X, F10.3, I8, 3F12.3/)
      END
```

Output

```
                        IDER
   X       IDER       F    (X)     QDDER       ERROR

0.785       0       0.707       0.707       0.000

0.785       1       0.707       0.707       0.000

0.785       2      -0.707      -0.704      -0.003
```

QD2VL/DQD2VL (Single/Double precision)

Evaluate a function defined on a rectangular grid using quadratic interpolation.

Usage

```
QD2VL(X, Y, NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF,
      CHECK)
```

Arguments

X — *x*-coordinate of the point at which the function is to be evaluated. (Input)

Y — *y*-coordinate of the point at which the function is to be evaluated. (Input)

NXDATA — Number of data points in the *x*-direction. (Input)
NXDATA must be at least three.

XDATA — Array of length NXDATA containing the location of the data points in the *x*-direction. (Input)
XDATA must be increasing.

NYDATA — Number of data points in the *y*-direction. (Input)
NYDATA must be at least three.

YDATA — Array of length NYDATA containing the location of the data points in the *y*-direction. (Input)
YDATA must be increasing.

FDATA — Array of size NXDATA by NYDATA containing function values. (Input)
FDATA (I, J) is the value of the function at (XDATA (I), YDATA(J)).

LDF — Leading dimension of FDATA exactly as specified in the dimension statement of the calling program. (Input)
LDF must be at least as large as NXDATA.

CHECK — Logical variable that is .TRUE. if checking of XDATA and YDATA is required or .FALSE. if checking is not required. (Input)

QD2VL — Value of the function at (X, Y). (Output)

Comments

Informational errors

Type	Code	
4	6	The XDATA values must be strictly increasing.
4	7	The YDATA values must be strictly increasing.

Algorithm

The function QD2VL interpolates a table of values, using quadratic polynomials, returning an approximation to the tabulated function. Let (x_i, y_j, f_{ij}) for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$ be the tabular data. Given a point (x, y) at which an interpolated value is desired, we first find the nearest interior grid point (x_i, y_j). A bivariate quadratic interpolant q is then formed using six points near (x, y). Five of the six points are (x_i, y_j), $(x_{i\pm1}, y_j)$, and $(x_i, y_{j\pm1})$. The sixth point is the nearest point to (x, y) of the grid points $(x_{i\pm1}, y_{j\pm1})$. The value $q(x, y)$ is returned by QD2VL. One should note that, in general, the function QD2VL is discontinuous at the midlines of the mesh.

Example

In this example, the value of $\sin(x + y)$ at $x = y = \pi/4$ is approximated by using QDVAL on a table of size 21×42 equally spaced values on the unit square.

```
      INTEGER    LDF, NXDATA, NYDATA
      PARAMETER  (NXDATA=21, NYDATA=42, LDF=NXDATA)
C
      INTEGER    I, J, NOUT
      REAL       CONST, F, FDATA(LDF,NYDATA), FLOAT, PI, Q, QD2VL,
     &           SIN, X, XDATA(NXDATA), Y, YDATA(NYDATA)
      LOGICAL    CHECK
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, QD2VL, UMACH
C                                  Define function
      F(X,Y) = SIN(X+Y)
C                                  Set up X-grid
      DO 10  I=1, NXDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NXDATA-1)
   10 CONTINUE
C                                  Set up Y-grid
      DO 20  I=1, NYDATA
         YDATA(I) = FLOAT(I-1)/FLOAT(NYDATA-1)
   20 CONTINUE
C                                  Evaluate function on grid
      DO 30  I=1, NXDATA
         DO 30  J=1, NYDATA
            FDATA(I,J) = F(XDATA(I),YDATA(J))
   30 CONTINUE
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
C                                  Check XDATA and YDATA
```

```
      CHECK = .TRUE.
C                                 Get value for PI and set X and Y
      PI = CONST('PI')
      X  = PI/4.0
      Y  = PI/4.0
C                                 Evaluate quadratic at (X,Y)
      Q = QD2VL(X,Y,NXDATA,XDATA,NYDATA,YDATA,FDATA,NXDATA,CHECK)
C                                 Print results
      WRITE (NOUT,'(5F12.4)') X, Y, F(X,Y), Q, (Q-F(X,Y))
99999 FORMAT (10X, 'X', 11X, 'Y', 7X, 'F(X,Y)', 7X, 'QD2VL', 9X,
     &       'DIF')
      END
```

Output

```
     X           Y       F(X,Y)       QD2VL         DIF
0.7854      0.7854      1.0000      1.0000      0.0000
```

QD2DR/DQD2DR (Single/Double precision)

Evaluate the derivative of a function defined on a rectangular grid using quadratic interpolation.

Usage

```
QD2DR(IXDER, IYDER, X, Y, NXDATA, XDATA, NYDATA, YDATA,
      FDATA, LDF, CHECK)
```

Arguments

IXDER — Order of the *x*-derivative.   (Input)

IYDER — Order of the *y*-derivative.   (Input)

X — X-coordinate of the point at which the function is to be evaluated.   (Input)

Y — Y-coordinate of the point at which the function is to be evaluated.   (Input)

NXDATA — Number of data points in the *x*-direction.   (Input)
NXDATA must be at least three.

XDATA — Array of length NXDATA containing the location of the data points in the *x*-direction.   (Input)
XDATA must be increasing.

NYDATA – Number of data points in the *y*-direction.   (Input)
NYDATA must be at least three.

YDATA — Array of length NYDATA containing the location of the data points in the *y*-direction.   (Input)
YDATA must be increasing.

FDATA — Array of size NXDATA by NYDATA containing function values.   (Input)
FDATA(I, J) is the value of the function at (XDATA(I), YDATA(J)).

LDF — Leading dimension of `FDATA` exactly as specified in the dimension statement of the calling program. (Input)
`LDF` must be at least as large as `NXDATA`.

CHECK — Logical variable that is `.TRUE.` if checking of `XDATA` and `YDATA` is required or `.FALSE.` if checking is not required. (Input)

QD2DR — Value of the (`IXDER`, `IYDER`) derivative of the function at (`X`, `Y`). (Output)

Comments

1. Informational errors

Type	Code	
4	6	The `XDATA` values must be strictly increasing.
4	7	The `YDATA` values must be strictly increasing.

2. Because quadratic interpolation is used, if the order of any derivative is greater than two, then the returned value is zero.

Algorithm

The function `QD2DR` interpolates a table of values, using quadratic polynomials, returning an approximation to the tabulated function. Let (x_i, y_j, f_{ij}) for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_y$ be the tabular data. Given a point (x, y) at which an interpolated value is desired, we first find the nearest interior grid point (x_i, y_j). A bivariate quadratic interpolant q is then formed using six points near (x, y). Five of the six points are (x_i, y_j), $(x_{i\pm1}, y_j)$, and $(x_i, y_{j\pm1})$. The sixth point is the nearest point to (x, y) of the grid points $(x_{i\pm1}, y_{j\pm1})$. The value $q^{(p,\ r)}(x, y)$ is returned by `QD2DR`, where p = `IXDER` and r = `IYDER`. One should note that, in general, the function `QD2DR` is discontinuous at the midlines of the mesh.

Example

In this example, the partial derivatives of $\sin(x + y)$ at $x = y = \pi/3$ are approximated by using `QD2DR` on a table of size 21 × 42 equally spaced values on the rectangle $[0, 2] \times [0, 2]$.

```
      INTEGER    LDF, NXDATA, NYDATA
      PARAMETER  (NXDATA=21, NYDATA=42, LDF=NXDATA)
C
      INTEGER    I, IXDER, IYDER, J, NOUT
      REAL       CONST, F, FDATA(LDF,NYDATA), FLOAT, FU, FUNC, PI, Q,
     &           QD2DR, SIN, X, XDATA(NXDATA), Y, YDATA(NYDATA)
      LOGICAL    CHECK
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, FUNC, QD2DR, UMACH
C                                  Define function
      F(X,Y) = SIN(X+Y)
C                                  Set up X-grid
      DO 10  I=1, NXDATA
         XDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NXDATA-1))
```

```
   10 CONTINUE
C                                 Set up Y-grid
      DO 20  I=1, NYDATA
         YDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NYDATA-1))
   20 CONTINUE
C                                 Evaluate function on grid
      DO 30  I=1, NXDATA
         DO 30  J=1, NYDATA
            FDATA(I,J) = F(XDATA(I),YDATA(J))
   30 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99998)
C                                 Check XDATA and YDATA
      CHECK = .TRUE.
C                                 Get value for PI and set X and Y
      PI = CONST('PI')
      X  = PI/3.0
      Y  = PI/3.0
C                                 Evaluate and print the function
C                                 and its derivatives at X=PI/3 and
C                                 Y=PI/3.
      DO 40  IXDER=0, 1
         DO 40  IYDER=0, 1
            Q  = QD2DR(IXDER,IYDER,X,Y,NXDATA,XDATA,NYDATA,YDATA,FDATA,
     &           LDF,CHECK)
            FU = FUNC(IXDER,IYDER,X,Y)
            WRITE (NOUT,99999) X, Y, IXDER, IYDER, FU, Q, (FU-Q)
   40 CONTINUE
C
99998 FORMAT (32X, '(IDX,IDY)', /, 8X, 'X', 8X, 'Y', 3X, 'IDX', 2X,
     &       'IDY', 3X, 'F       (X,Y)', 3X, 'QD2DR', 6X, 'ERROR')
99999 FORMAT (2F9.4, 2I5, 3X, F9.4, 2X, 2F11.4)
      END
      REAL FUNCTION FUNC (IX, IY, X, Y)
      INTEGER    IX, IY
      REAL       X, Y
C
      REAL       COS, SIN
      INTRINSIC  COS, SIN
C
      IF (IX.EQ.0 .AND. IY.EQ.0) THEN
C                                 Define (0,0) derivative
         FUNC = SIN(X+Y)
      ELSE IF (IX.EQ.0 .AND. IY.EQ.1) THEN
C                                 Define (0,1) derivative
         FUNC = COS(X+Y)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.0) THEN
C                                 Define (1,0) derivative
         FUNC = COS(X+Y)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.1) THEN
C                                 Define (1,1) derivative
         FUNC = -SIN(X+Y)
      ELSE
         FUNC = 0.0
      END IF
      RETURN
      END
```

Output

```
                              (IDX,IDY)
     X          Y    IDX  IDY   F       (X,Y)   QD2DR       ERROR
1.0472   1.0472    0    0      0.8660       0.8661    -0.0001
1.0472   1.0472    0    1     -0.5000      -0.4993    -0.0007
1.0472   1.0472    1    0     -0.5000      -0.4995    -0.0005
1.0472   1.0472    1    1     -0.8660      -0.8634    -0.0026
```

QD3VL/DQD3VL (Single/Double precision)

Evaluate a function defined on a rectangular three-dimensional grid using quadratic interpolation.

Usage

```
QD3VL(X, Y, Z, NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA,
      FDATA, LDF, MDF, CHECK)
```

Arguments

X — *x*-coordinate of the point at which the function is to be evaluated. (Input)

Y — *y*-coordinate of the point at which the function is to be evaluated. (Input)

Z — *z*-coordinate of the point at which the function is to be evaluated. (Input)

NXDATA — Number of data points in the *x*-direction. (Input)
NXDATA must be at least three.

XDATA — Array of length NXDATA containing the location of the data points in the *x*-direction. (Input)
XDATA must be increasing.

NYDATA — Number of data points in the *y*-direction. (Input)
NYDATA must be at least three.

YDATA — Array of length NYDATA containing the location of the data points in the *y*-direction. (Input)
YDATA must be increasing.

NZDATA — Number of data points in the *z*-direction. (Input)
NZDATA must be at least three.

ZDATA — Array of length NZDATA containing the location of the data points in the *z*-direction. (Input)
ZDATA must be increasing.

FDATA — Array of size NXDATA by NYDATA by NZDATA containing function values. (Input)
FDATA(I, J, K) is the value of the function at (XDATA(I), YDATA(J), ZDATA(K)).

LDF — Leading dimension of FDATA exactly as specified in the dimension statement of the calling program. (Input)
LDF must be at least as large as NXDATA.

MDF — Middle (second) dimension of FDATA exactly as specified in the dimension statement of the calling program.   (Input)
MDF must be at least as large as NYDATA.

CHECK — Logical variable that is .TRUE. if checking of XDATA, YDATA, and ZDATA is required or .FALSE. if checking is not required.   (Input)

QD3VL — Value of the function at (X, Y, Z).   (Output)

Comments

Informational errors

Type	Code	
4	9	The XDATA values must be strictly increasing.
4	10	The YDATA values must be strictly increasing.
4	11	The ZDATA values must be strictly increasing.

Algorithm

The function QD3VL interpolates a table of values, using quadratic polynomials, returning an approximation to the tabulated function. Let (x_i, y_j, z_k, f_{ijk}) for i = 1, …, n_x, j = 1, …, n_y, and k = 1, …, n_z be the tabular data. Given a point (x, y, z) at which an interpolated value is desired, we first find the nearest interior grid point (x_i, y_j, z_k,). A trivariate quadratic interpolant q is then formed. Ten points are needed for this purpose. Seven points have the form

$$\left(x_i, y_j, z_k\right), \left(x_{i\pm1}, y_j, z_k\right), \left(x_i, y_{j\pm1}, z_k\right) \text{ and } \left(x_i, y_j, z_{k\pm1}\right)$$

The last three points are drawn from the vertices of the octant containing (x, y, z). There are four of these vertices remaining, and we choose to exclude the vertex farthest from the center. This has the slightly deleterious effect of not reproducing the tabular data at the eight exterior corners of the table. The value q(x, y, z) is returned by QD3VL. One should note that, in general, the function QD3VL is discontinuous at the midplanes of the mesh.

Example

In this example, the value of sin(x + y + z) at $x = y = z = \pi/3$ is approximated by using QD3VL on a grid of size 21 × 42 × 18 equally spaced values on the cube $[0, 2]^3$.

```
      INTEGER    LDF, MDF, NXDATA, NYDATA, NZDATA
      PARAMETER  (NXDATA=21, NYDATA=42, NZDATA=18, LDF=NXDATA,
     &           MDF=NYDATA)
C
      INTEGER    I, J, K, NOUT
      REAL       CONST, F, FDATA(LDF,MDF,NZDATA), FLOAT, PI, Q, QD3VL,
     &           SIN, X, XDATA(NXDATA), Y, YDATA(NYDATA), Z,
     &           ZDATA(NZDATA)
      LOGICAL    CHECK
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, QD3VL, UMACH
C                                 Define function
```

```
      F(X,Y,Z) = SIN(X+Y+Z)
C                                 Set up X-grid
      DO 10  I=1, NXDATA
         XDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NXDATA-1))
   10 CONTINUE
C                                 Set up Y-grid
      DO 20  J=1, NYDATA
         YDATA(J) = 2.0*(FLOAT(J-1)/FLOAT(NYDATA-1))
   20 CONTINUE
C                                 Set up Z-grid
      DO 30  K=1, NZDATA
         ZDATA(K) = 2.0*(FLOAT(K-1)/FLOAT(NZDATA-1))
   30 CONTINUE
C                                 Evaluate function on grid
      DO 40  I=1, NXDATA
         DO 40  J=1, NYDATA
            DO 40  K=1, NZDATA
               FDATA(I,J,K) = F(XDATA(I),YDATA(J),ZDATA(K))
   40 CONTINUE
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Check XDATA, YDATA, and ZDATA
      CHECK = .TRUE.
C                                 Get value for PI and set values
C                                 for X, Y, and Z
      PI = CONST('PI')
      X  = PI/3.0
      Y  = PI/3.0
      Z  = PI/3.0
C                                 Evaluate quadratic at (X,Y,Z)
      Q = QD3VL(X,Y,Z,NXDATA,XDATA,NYDATA,YDATA,NZDATA,ZDATA,FDATA,LDF,
     &    MDF,CHECK)
C                                 Print results
      WRITE (NOUT,'(6F11.4)') X, Y, Z, F(X,Y,Z), Q, (Q-F(X,Y,Z))
99999 FORMAT (10X, 'X', 10X, 'Y', 10X, 'Z', 5X, 'F(X,Y,Z)', 4X,
     &        'QD3VL', 6X, 'ERROR')
      END
```

Output

```
      X          Y          Z     F(X,Y,Z)    QD3VL      ERROR
1.0472     1.0472     1.0472     0.0000     0.0002     0.0002
```

QD3DR/DQD3DR (Single/Double precision)

Evaluate the derivative of a function defined on a rectangular three-dimensional grid using quadratic interpolation.

Usage

```
QD3DR(IXDER, IYDER, IZDER, X, Y, Z, NXDATA, XDATA, NYDATA,
      YDATA, NZDATA, ZDATA, FDATA, LDF, MDF, CHECK)
```

Arguments

IXDER — Order of the *x*-derivative. (Input)

IYDER — Order of the *y*-derivative. (Input)

IZDER — Order of the *z*-derivative. (Input)

X — *x*-coordinate of the point at which the function is to be evaluated. (Input)

Y — *y*-coordinate of the point at which the function is to be evaluated. (Input)

Z — *z*-coordinate of the point at which the function is to be evaluated. (Input)

NXDATA — Number of data points in the *x*-direction. (Input)
`NXDATA` must be at least three.

XDATA — Array of length `NXDATA` containing the location of the data points in the *x*-direction. (Input)
`XDATA` must be increasing.

NYDATA — Number of data points in the *y*-direction. (Input)
`NYDATA` must be at least three.

YDATA — Array of length `NYDATA` containing the location of the data points in the *y*-direction. (Input)
`YDATA` must be increasing.

NZDATA — Number of data points in the *z*-direction. (Input)
`NZDATA` must be at least three.

ZDATA — Array of length `NZDATA` containing the location of the data points in the *z*-direction. (Input)
`ZDATA` must be increasing.

FDATA — Array of size `NXDATA` by `NYDATA` by `NZDATA` containing function values. (Input)
`FDATA`(`I`, `J`, `K`) is the value of the function at (`XDATA`(`I`), `YDATA`(`J`), `ZDATA`(`K`)).

LDF — Leading dimension of `FDATA` exactly as specified in the dimension statement of the calling program. (Input)
`LDF` must be at least as large as `NXDATA`.

MDF — Middle (second) dimension of `FDATA` exactly as specified in the dimension statement of the calling program. (Input)
`MDF` must be at least as large as `NYDATA`.

CHECK — Logical variable that is `.TRUE.` if checking of `XDATA`, `YDATA`, and `ZDATA` is required or `.FALSE.` if checking is not required. (Input)

QD3DR — Value of the appropriate derivative of the function at (`X`, `Y`, `Z`). (Output)

Comments

1. Informational errors

Type	Code	
4	9	The XDATA values must be strictly increasing.
4	10	The YDATA values must be strictly increasing.
4	11	The ZDATA values must be strictly increasing.

2. Because quadratic interpolation is used, if the order of any derivative is greater than two, then the returned value is zero.

Algorithm

The function QD3DR interpolates a table of values, using quadratic polynomials, returning an approximation to the partial derivatives of the tabulated function. Let

$$(x_i, y_j, z_k, f_{ijk})$$

for $i = 1, \ldots, n_x$, $j = 1, \ldots, n_y$, and $k = 1, \ldots, n_z$ be the tabular data. Given a point (x, y, z) at which an interpolated value is desired, we first find the nearest interior grid point (x_i, y_j, z_k). A trivariate quadratic interpolant q is then formed. Ten points are needed for this purpose. Seven points have the form

$$\left(x_i, y_j, z_k\right), \left(x_{i\pm1}, y_j, z_k\right), \left(x_i, y_{j\pm1}, z_k\right) \text{ and } \left(x_i, y_j, z_{k\pm1}\right)$$

The last three points are drawn from the vertices of the octant containing (x, y, z). There are four of these vertices remaining, and we choose to exclude the vertex farthest from the center. This has the slightly deleterious effect of not reproducing the tabular data at the eight exterior corners of the table. The value $q^{(p,r,t)}(x, y, z)$ is returned by QD3DR, where p = IXDER, r = IYDER, and t = IZDER. One should note that, in general, the function QD3DR is discontinuous at the midplanes of the mesh.

Example

In this example, the derivatives of $\sin(x + y + z)$ at $x = y = z = \pi/5$ are approximated by using QD3DR on a grid of size 21 × 42 × 18 equally spaced values on the cube $[0, 2]^3$.

```
      INTEGER    LDF, MDF, NXDATA, NYDATA, NZDATA
      PARAMETER  (NXDATA=21, NYDATA=42, NZDATA=18, LDF=NXDATA,
     &           MDF=NYDATA)
C
      INTEGER    I, IXDER, IYDER, IZDER, J, K, NOUT
      REAL       CONST, F, FDATA(NXDATA,NYDATA,NZDATA), FLOAT, FU,
     &           FUNC, PI, Q, QD3DR, SIN, X, XDATA(NXDATA), Y,
     &           YDATA(NYDATA), Z, ZDATA(NZDATA)
      LOGICAL    CHECK
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, FUNC, QD3DR, UMACH
C                                Define function
      F(X,Y,Z) = SIN(X+Y+Z)
C                                 Set up X-grid
```

```
      DO 10  I=1, NXDATA
         XDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NXDATA-1))
   10 CONTINUE
C                                  Set up Y-grid
      DO 20  J=1, NYDATA
         YDATA(J) = 2.0*(FLOAT(J-1)/FLOAT(NYDATA-1))
   20 CONTINUE
C                                  Set up Z-grid
      DO 30  K=1, NZDATA
         ZDATA(K) = 2.0*(FLOAT(K-1)/FLOAT(NZDATA-1))
   30 CONTINUE
C                                  Evaluate function on grid
      DO 40  I=1, NXDATA
         DO 40  J=1, NYDATA
            DO 40  K=1, NZDATA
               FDATA(I,J,K) = F(XDATA(I),YDATA(J),ZDATA(K))
   40 CONTINUE
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
C                                  Check XDATA, YDATA, and ZDATA
      CHECK = .TRUE.
C                                  Get value for PI and set X, Y, and Z
      PI = CONST('PI')
      X  = PI/5.0
      Y  = PI/5.0
      Z  = PI/5.0
C                                  Compute derivatives at (X,Y,Z)
C                                  and print results
      DO 50  IXDER=0, 1
         DO 50  IYDER=0, 1
            DO 50  IZDER=0, 1
               Q  = QD3DR(IXDER,IYDER,IZDER,X,Y,Z,NXDATA,XDATA,NYDATA,
     &              YDATA,NZDATA,ZDATA,FDATA,NXDATA,NYDATA,CHECK)
               FU = FUNC(IXDER,IYDER,IZDER,X,Y,Z)
               WRITE (NOUT,99998) X, Y, Z, IXDER, IYDER, IZDER, FU, Q,
     &                             (FU-Q)
   50 CONTINUE
C
99998 FORMAT (3F7.4, 3I5, 4X, F7.4, 8X, 2F10.4)
99999 FORMAT (39X, '(IDX,IDY,IDZ)', /, 6X, 'X', 6X, 'Y', 6X,
     &       'Z', 3X, 'IDX', 2X, 'IDY', 2X, 'IDZ', 2X, 'F         ',
     &       '(X,Y,Z)', 3X, 'QD3DR', 5X, 'ERROR')
      END
C
      REAL FUNCTION FUNC (IX, IY, IZ, X, Y, Z)
      INTEGER    IX, IY, IZ
      REAL       X, Y, Z
C
      REAL       COS, SIN
      INTRINSIC  COS, SIN
C
      IF (IX.EQ.0 .AND. IY.EQ.0 .AND. IZ.EQ.0) THEN
C                                  Define (0,0,0) derivative
         FUNC = SIN(X+Y+Z)
      ELSE IF (IX.EQ.0 .AND. IY.EQ.0 .AND. IZ.EQ.1) THEN
C                                  Define (0,0,1) derivative
```

```
         FUNC = COS(X+Y+Z)
      ELSE IF (IX.EQ.0 .AND. IY.EQ.1 .AND. IZ.EQ.0) THEN
C                                 Define (0,1,0,) derivative
         FUNC = COS(X+Y+Z)
      ELSE IF (IX.EQ.0 .AND. IY.EQ.1 .AND. IZ.EQ.1) THEN
C                                 Define (0,1,1) derivative
         FUNC = -SIN(X+Y+Z)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.0 .AND. IZ.EQ.0) THEN
C                                 Define (1,0,0) derivative
         FUNC = COS(X+Y+Z)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.0 .AND. IZ.EQ.1) THEN
C                                 Define (1,0,1) derivative
         FUNC = -SIN(X+Y+Z)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.1 .AND. IZ.EQ.0) THEN
C                                 Define (1,1,0) derivative
         FUNC = -SIN(X+Y+Z)
      ELSE IF (IX.EQ.1 .AND. IY.EQ.1 .AND. IZ.EQ.1) THEN
C                                 Define (1,1,1) derivative
         FUNC = -COS(X+Y+Z)
      ELSE
         FUNC = 0.0
      END IF
      RETURN
      END
```

Output

```
                                    (IDX,IDY,IDZ)
    X      Y      Z   IDX  IDY  IDZ  F          (X,Y,Z)   QD3DR     ERROR
0.6283 0.6283 0.6283    0    0    0    0.9511           0.9511   -0.0001
0.6283 0.6283 0.6283    0    0    1    -0.3090           -0.3083   -0.0007
0.6283 0.6283 0.6283    0    1    0    -0.3090           -0.3091    0.0000
0.6283 0.6283 0.6283    0    1    1    -0.9511           -0.9587    0.0077
0.6283 0.6283 0.6283    1    0    0    -0.3090           -0.3082   -0.0008
0.6283 0.6283 0.6283    1    0    1    -0.9511           -0.9509   -0.0002
0.6283 0.6283 0.6283    1    1    0    -0.9511           -0.9613    0.0103
0.6283 0.6283 0.6283    1    1    1     0.3090            0.0000    0.3090
```

SURF/DSURF (Single/Double precision)

Compute a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.

Usage

```
CALL SURF (NDATA, XYDATA, FDATA, NXOUT, NYOUT, XOUT, YOUT,
           SUR, LDSUR)
```

Arguments

NDATA — Number of data points. (Input)
`NDATA` must be at least four.

XYDATA — A 2 by `NDATA` array containing the coordinates of the interpolation points. (Input)
These points must be distinct. The *x*-coordinate of the `I`-th data point is stored in

XYDATA(1, I) and the *y*-coordinate of the I-th data point is stored in XYDATA(2, I).

FDATA — Array of length NDATA containing the interpolation values. (Input)
FDATA(I) contains the value at (XYDATA(1, I), XYDATA(2, I)).

NXOUT — The number of elements in XOUT. (Input)

NYOUT — The number of elements in YOUT. (Input)

XOUT — Array of length NXOUT containing an increasing sequence of points. (Input)
These points are the *x*-coordinates of a grid on which the interpolated surface is to be evaluated.

YOUT — Array of length NYOUT containing an increasing sequence of points. (Input)
These points are the *y*-coordinates of a grid on which the interpolated surface is to be evaluated.

SUR — Matrix of size NXOUT by NYOUT. (Output)
This matrix contains the values of the surface on the XOUT by YOUT grid, i.e. SUR(I, J) contains the interpolated value at (XOUT(I), YOUT(J)).

LDSUR — Leading dimension of SUR exactly as specified in the dimension statement of the calling program. (Input)
LDSUR must be at least as large as NXOUT.

Comments

1. Automatic workspace usage is

   SURF 31 * NDATA + NXOUT * NYOUT + 6 * NDATA units, or

   DSURF 31 * NDATA+ NXOUT * NYOUT + 12 * NDATA units.

   Workspace may be explicitly provided, if desired, by use of S2RF/DS2RF. The reference is

   ```
   CALL S2RF (NDATA, XYDATA, FDATA, NXOUT, NYOUT, XOUT,
                 YOUT, SUR, LDSUR, IWK, WK)
   ```

   The additional arguments are as follows:

   IWK — Work array of length 31 * NDATA + NXOUT * NYOUT.

   WK — Work array of length 6 * NDATA.

2. Informational errors

Type	Code	
4	5	The data point values must be distinct.
4	6	The XOUT values must be strictly increasing.
4	7	The YOUT values must be strictly increasing.

3. This method of interpolation reproduces linear functions.

Algorithm

This routine is designed to compute a C^1 interpolant to scattered data in the plane. Given the data points

$$\left\{(x_i,\, y_i,\, f_i)\right\}_{i=1}^{N} \text{ in } \mathbf{R}^3$$

`SURF` returns (in `SUR`, the user-specified grid) the values of the interpolant *s*. The computation of *s* is as follows: First the Delaunay triangulation of the points

$$\left\{(x_i,\, y_i)\right\}_{i=1}^{N}$$

is computed. On each triangle *T* in this triangulation, *s* has the form

$$s(x,\, y) = \sum_{m+n\le 5} c_{mn}^{T} x^m y^n \qquad \forall x,\, y \in T$$

Thus, *s* is a bivariate quintic polynomial on each triangle of the triangulation. In addition, we have

$$s(x_i, y_i) = f_i \qquad \text{for } i = 1, \ldots, N$$

and *s* is continuously differentiable across the boundaries of neighboring triangles. These conditions do not exhaust the freedom implied by the above representation. This additional freedom is exploited in an attempt to produce an interpolant that is faithful to the global shape properties implied by the data. For more information on this routine, we refer the reader to the article by Akima (1978). The grid is specified by the two integer variables `NXOUT`, `NYOUT` that represent, respectively, the number of grid points in the first (second) variable and by two real vectors that represent, respectively, the first (second) coordinates of the grid.

Example

In this example, the interpolant to the linear function 3 + 7*x* + 2*y* is computed from 20 data points equally spaced on the circle of radius 3. We then print the values on a 3 × 3 grid.

```
      INTEGER    LDSUR, NDATA, NXOUT, NYOUT
      PARAMETER  (NDATA=20, NXOUT=3, NYOUT=3, LDSUR=NXOUT)
C
      INTEGER    I, J, NOUT
      REAL       ABS, CONST, COS, F, FDATA(NDATA), FLOAT, PI,
     &           SIN, SUR(LDSUR,NYOUT), X, XOUT(NXOUT),
     &           XYDATA(2,NDATA), Y, YOUT(NYOUT)
      INTRINSIC  ABS, COS, FLOAT, SIN
      EXTERNAL   CONST, SURF, UMACH
C                                 Define function
      F(X,Y) = 3.0 + 7.0*X + 2.0*Y
C                                 Get value for PI
      PI     = CONST('PI')
C                                 Set up X, Y, and F data on a circle
      DO 10  I=1, NDATA
         XYDATA(1,I) = 3.0*SIN(2.0*PI*FLOAT(I-1)/FLOAT(NDATA))
         XYDATA(2,I) = 3.0*COS(2.0*PI*FLOAT(I-1)/FLOAT(NDATA))
         FDATA(I)    = F(XYDATA(1,I),XYDATA(2,I))
   10 CONTINUE
C                                 Set up XOUT and YOUT data on [0,1] by
C                                 [0,1] grid.
      DO 20  I=1, NXOUT
         XOUT(I) = FLOAT(I-1)/FLOAT(NXOUT-1)
   20 CONTINUE
      DO 30  I=1, NXOUT
         YOUT(I) = FLOAT(I-1)/FLOAT(NYOUT-1)
   30 CONTINUE
C                                 Interpolate scattered data
      CALL SURF (NDATA, XYDATA, FDATA, NXOUT, NYOUT, XOUT, YOUT, SUR,
     &           LDSUR)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99998)
C                                 Print results
      DO 40  I=1, NYOUT
         DO 40  J=1, NXOUT
            WRITE (NOUT,99999) XOUT(J), YOUT(I), SUR(J,I),
     &                        F(XOUT(J),YOUT(I)),
     &                        ABS(SUR(J,I)-F(XOUT(J),YOUT(I)))
   40 CONTINUE
99998 FORMAT (' ', 10X, 'X', 11X, 'Y', 9X, 'SURF', 6X, 'F(X,Y)', 7X,
     &       'ERROR', /)
99999 FORMAT (1X, 5F12.4)
      END
```

Output

```
     X           Y         SURF      F(X,Y)       ERROR

0.0000      0.0000      3.0000      3.0000      0.0000
0.5000      0.0000      6.5000      6.5000      0.0000
1.0000      0.0000     10.0000     10.0000      0.0000
0.0000      0.5000      4.0000      4.0000      0.0000
0.5000      0.5000      7.5000      7.5000      0.0000
1.0000      0.5000     11.0000     11.0000      0.0000
0.0000      1.0000      5.0000      5.0000      0.0000
0.5000      1.0000      8.5000      8.5000      0.0000
1.0000      1.0000     12.0000     12.0000      0.0000
```

RLINE/DRLINE (Single/Double precision)

Fit a line to a set of data points using least squares.

Usage

`CALL RLINE (NOBS, XDATA, YDATA, B0, B1, STAT)`

Arguments

NOBS — Number of observations. (Input)

XDATA — Vector of length `NOBS` containing the *x*-values. (Input)

YDATA — Vector of length `NOBS` containing the *y*-values. (Input)

B0 — Estimated intercept of the fitted line. (Output)

B1 — Estimated slope of the fitted line. (Output)

STAT — Vector of length 12 containing the statistics described below. (Output)

I	**ISTAT(I)**
1	Mean of `XDATA`
2	Mean of `YDATA`
3	Sample variance of `XDATA`
4	Sample variance of `YDATA`
5	Correlation
6	Estimated standard error of `B0`
7	Estimated standard error of `B1`
8	Degrees of freedom for regression
9	Sum of squares for regression
10	Degrees of freedom for error
11	Sum of squares for error
12	Number of (*x*, *y*) points containing NaN (not a number) as either the *x* or *y* value

Comments

Informational error

Type	Code	
4	1	Each (*x*, *y*) point contains NaN (not a number). There are no valid data.

Algorithm

Routine `RLINE` fits a line to a set of (*x*, *y*) data points using the method of least squares. Draper and Smith (1981, pages 1–69) discuss the method. The fitted model is

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

where

$$\hat{\beta}_0$$

(stored in `B0`) is the estimated intercept and

$$\hat{\beta}_1$$

(stored in `B1`) is the estimated slope. In addition to the fit, `RLINE` produces some summary statistics, including the means, sample variances, correlation, and the error (residual) sum of squares. The estimated standard errors of

$$\hat{\beta}_0 \text{ and } \hat{\beta}_1$$

are computed under the simple linear regression model. The errors in the model are assumed to be uncorrelated and with constant variance.

If the *x* values are all equal, the model is degenerate. In this case, RLINE sets

$$\hat{\beta}_1$$

to zero and

$$\hat{\beta}_0$$

to the mean of the *y* values.

Example

This example fits a line to a set of data discussed by Draper and Smith (1981, Table 1.1, pages 9−33). The response *y* is the amount of steam used per month (in pounds), and the independent variable *x* is the average atmospheric temperature (in degrees Fahrenheit).

```
      INTEGER    NOBS
      PARAMETER  (NOBS=25)
C
      INTEGER    NOUT
      REAL       B0, B1, STAT(12), XDATA(NOBS), YDATA(NOBS)
      CHARACTER  CLABEL(13)*15, RLABEL(1)*4
      EXTERNAL   RLINE, UMACH, WRRRL
C
      DATA XDATA/35.3, 29.7, 30.8, 58.8, 61.4, 71.3, 74.4, 76.7, 70.7,
     &     57.5, 46.4, 28.9, 28.1, 39.1, 46.8, 48.5, 59.3, 70.0, 70.0,
     &     74.5, 72.1, 58.1, 44.6, 33.4, 28.6/
      DATA YDATA/10.98, 11.13, 12.51, 8.4, 9.27, 8.73, 6.36, 8.5,
     &     7.82, 9.14, 8.24, 12.19, 11.88, 9.57, 10.94, 9.58, 10.09,
     &     8.11, 6.83, 8.88, 7.68, 8.47, 8.86, 10.36, 11.08/
      DATA RLABEL/’NONE’/, CLABEL/’ ’, ’Mean of X’, ’Mean of Y’,
     &     ’Variance X’, ’Variance Y’, ’Corr.’, ’Std. Err. B0’,
     &     ’Std. Err. B1’, ’DF Reg.’, ’SS Reg.’, ’DF Error’,
     &     ’SS Error’, ’Pts. with NaN’/
C
      CALL RLINE (NOBS, XDATA, YDATA, B0, B1, STAT)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) B0, B1
99999 FORMAT (’ B0 = ’, F7.2, ’  B1 = ’, F9.5)
      CALL WRRRL (’%/STAT’, 1, 12, STAT, 1, 0, ’(12W10.4)’, RLABEL,
     &             CLABEL)
C
      END
```

Output

```
B0 =   13.62  B1 =  -0.07983

                                STAT
Mean of X   Mean of Y  Variance X  Variance Y       Corr.  Std. Err. B0
      52.6          9.424       298.1       2.659     -0.8452         0.5815

Std. Err. B1     DF Reg.     SS Reg.    DF Error    SS Error  Pts. with NaN
0.01052           1       45.59          23       18.22              0
```

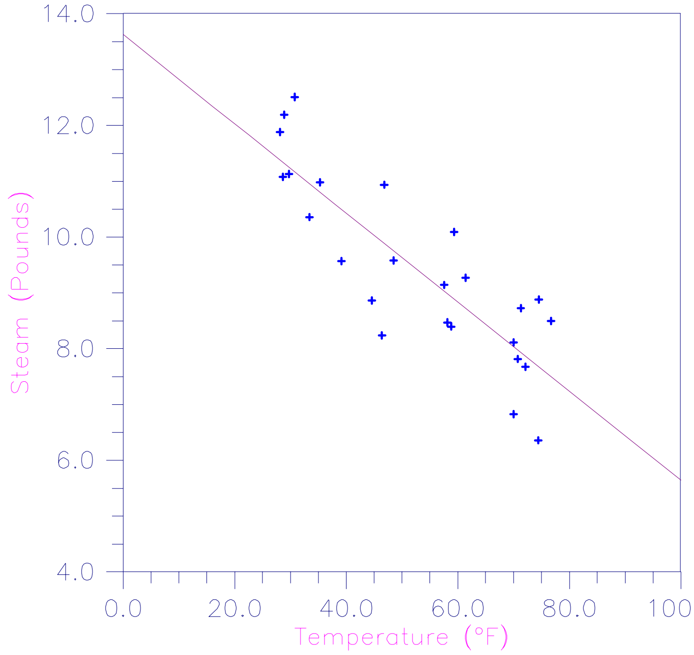


Figure 3-5   Plot of the Data and the Least Squares Line

RCURV/DRCURV (Single/Double precision)

Fit a polynomial curve using least squares.

Usage

```
CALL RCURV (NOBS, XDATA, YDATA, NDEG, B, SSPOLY, STAT)
```

Arguments

NOBS — Number of observations.   (Input)

XDATA — Vector of length `NOBS` containing the *x* values.   (Input)

YDATA — Vector of length `NOBS` containing the *y* values. (Input)

NDEG — Degree of polynomial. (Input)

B — Vector of length `NDEG` + 1 containing the coefficients

$$\hat{\beta}$$

(Output)
The fitted polynomial is

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_k x^k$$

SSPOLY — Vector of length `NDEG` + 1 containing the sequential sums of squares. (Output)
`SSPOLY`(1) contains the sum of squares due to the mean. For *i* = 1, 2, …, `NDEG`, `SSPOLY(i` + 1) contains the sum of squares due to x^i adjusted for the mean, *x*, x^2, …, and x^{i-1}.

STAT — Vector of length 10 containing statistics described below. (Output)

i	**Statistics**
1	Mean of *x*
2	Mean of *y*
3	Sample variance of *x*
4	Sample variance of *y*
5	R-squared (in percent)
6	Degrees of freedom for regression
7	Regression sum of squares
8	Degrees of freedom for error
9	Error sum of squares
10	Number of data points (*x*, *y*) containing NaN (not a number) as a *x* or *y* value

Comments

1. Automatic workspace usage is

   `RCURV` 12 * `NOBS` + 11 * `NDEG` + (`NDEG` + 1) * (`NDEG` + 3) + 5 units, or

   `DRCURV` 23 * `NOBS` + 22 * `NDEG` + 2 * (`NDEG` + 1) * (`NDEG` + 3) + 10 units.

   Workspace may be explicitly provided, if desired, by use of `R2URV/DR2URV`. The reference is

   ```
   CALL R2URV (NOBS, XDATA, YDATA, NDEG, B, SSPOLY,
               STAT, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 11 * `NOBS` + 11 * `NDEG` + 5 + (`NDEG` + 1) * (`NDEG` + 3).

IWK — Work vector of length `NOBS`.

2. Informational errors

Type	Code	
4	3	Each (x, y) point contains NaN (not a number). There are no valid data.
4	7	The x values are constant. At least `NDEG` + 1 distinct x values are needed to fit a `NDEG` polynomial.
3	4	The y values are constant. A zero order polynomial is fit. High order coefficients are set to zero.
3	5	There are too few observations to fit the desired degree polynomial. High order coefficients are set to zero.
3	6	A perfect fit was obtained with a polynomial of degree less than `NDEG`. High order coefficients are set to zero.

3. If `NDEG` is greater than 10, the accuracy of the results may be questionable.

Algorithm

Routine `RCURV` computes estimates of the regression coefficients in a polynomial (curvilinear) regression model. In addition to the computation of the fit, `RCURV` computes some summary statistics. Sequential sums of squares attributable to each power of the independent variable (stored in `SSPOLY`) are computed. These are useful in assessing the importance of the higher order powers in the fit. Draper and Smith (1981, pages 101−102) and Neter and Wasserman (1974, pages 278−287) discuss the interpretation of the sequential sums of squares. The statistic R^2 (stored in `STAT(5)`) is the percentage of the sum of squares of y about its mean explained by the polynomial curve. Specifically,

$$R^2 = \frac{\sum_{i=1}^{n}\left(\hat{y}_i - \overline{y}\right)^2}{\sum_{i=1}^{n}\left(y_i - \overline{y}\right)^2} 100\%$$

where

$$\hat{y}_i$$

is the fitted y value at x_i and

$$\overline{y}$$

(stored in `STAT(2)`) is the mean of y. This statistic is useful in assessing the overall fit of the curve to the data. R^2 must be between 0% and 100%, inclusive. R^2 = 100% indicates a perfect fit to the data.

Routine `RCURV` computes estimates of the regression coefficients in a polynomial model using orthogonal polynomials as the regressor variables. This reparameterization of the polynomial model in terms of orthogonal polynomials has the advantage that the loss of accuracy resulting from forming powers of the *x*-values is avoided. All results are returned to the user for the original model.

The routine RCURV is based on the algorithm of Forsythe (1957). A modification to Forsythe's algorithm suggested by Shampine (1975) is used for computing the polynomial coefficients. A discussion of Forsythe's algorithm and Shampine's modification appears in Kennedy and Gentle (1980, pages 342−347).

Example

A polynomial model is fitted to data discussed by Neter and Wasserman (1974, pages 279−285). The data set contains the response variable *y* measuring coffee sales (in hundred gallons) and the number of self-service coffee dispensers. Responses for fourteen similar cafeterias are in the data set.

```
      INTEGER    NDEG, NOBS
      PARAMETER  (NDEG=2, NOBS=14)
C
      REAL       B(NDEG+1), SSPOLY(NDEG+1), STAT(10), XDATA(NOBS),
     &           YDATA(NOBS)
      CHARACTER  CLABEL(11)*15, RLABEL(1)*4
      EXTERNAL   RCURV, WRRRL, WRRRN
C
      DATA RLABEL/'NONE'/, CLABEL/' ', 'Mean of X', 'Mean of Y',
     &           'Variance X', 'Variance Y', 'R-squared',
     &           'DF Reg.', 'SS Reg.', 'DF Error', 'SS Error',
     &           'Pts. with NaN'/
      DATA XDATA/0., 0., 1., 1., 2., 2., 4., 4., 5., 5., 6., 6., 7.,
     &     7./
      DATA YDATA/508.1, 498.4, 568.2, 577.3, 651.7, 657.0, 755.3,
     &     758.9, 787.6, 792.1, 841.4, 831.8, 854.7, 871.4/
C
      CALL RCURV (NOBS, XDATA, YDATA, NDEG, B, SSPOLY, STAT)
C
      CALL WRRRN ('B', 1, NDEG+1, B, 1, 0)
      CALL WRRRN ('SSPOLY', 1, NDEG+1, SSPOLY, 1, 0)
      CALL WRRRL ('%/STAT', 1, 10, STAT, 1, 0, '(2W10.4)', RLABEL,
     &            CLABEL)
      END
```

Output

```
          B
    1       2       3
503.3    78.9    -4.0

             SSPOLY
        1          2          3
7077152.0    220644.2      4387.7

                               STAT
Mean of X   Mean of Y  Variance X  Variance Y   R-squared    DF Reg.
     3.571       711.0       6.418     17364.8       99.69           2

 SS Reg.    DF Error    SS Error  Pts. with NaN
225031.9          11       710.5              0
```

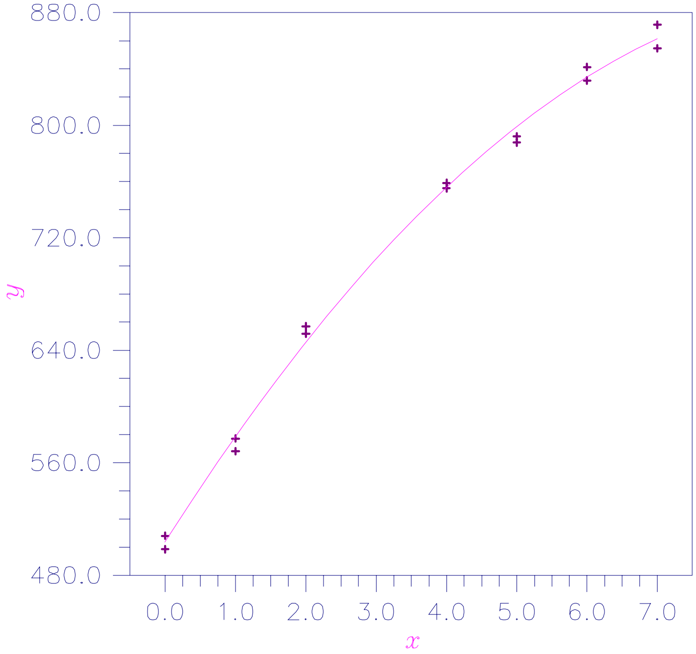


Figure 3-6   Plot of Data and Second Degree Polynomial Fit

FNLSQ/DFNLSQ (Single/Double precision)

Compute a least-squares approximation with user-supplied basis functions.

Usage

```
CALL FNLSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA, IWT,
            WEIGHT, A, SSE)
```

Arguments

F — User-supplied `FUNCTION` to evaluate basis functions. The form is `F`(`K`, `X`), where

`K` – Number of the basis function.   (Input)
`K` may be equal to 1, 2, …, `NBASIS`.
`X` – Argument for evaluation of the `K`-th basis function.   (Input)
`F` – The function value.   (Output)
`F` must be declared `EXTERNAL` in the calling program. The data `FDATA` is approximated by `A`(1) `*` `F`(1, `X`) + `A`(2) `*` `F`(2, `X`) +…+ `A`(`NBASIS`) `*` `F`(`NBASIS`, `X`) if `INTCEP` = 0 and is approximated by `A`(1) + `A`(2) `*` `F`(1, `X`) +…+ `A`(`NBASIS` + 1) `*` `F`(`NBASIS`, `X`) if `INTCEP` = 1.

INTCEP — Intercept option.   (Input)

INTCEP	Action
0	No intercept is automatically included in the model.
1	An intercept is automatically included in the model.

NBASIS — Number of basis functions.   (Input)

NDATA — Number of data points.   (Input)

XDATA — Array of length NDATA containing the abscissas of the data points. (Input)

FDATA — Array of length NDATA containing the ordinates of the data points. (Input)

IWT — Weighting option.   (Input)

IWT	Action
0	Weights of one are assumed.
1	Weights are supplied in WEIGHT.

WEIGHT — Array of length NDATA containing the weights.   (Input if IWT = 1)
If IWT = 0, WEIGHT is not referenced and may be dimensioned of length one.

A — Array of length INTCEP + NBASIS containing the coefficients of the approximation.   (Output)
If INTCEP = 1, A(1) contains the intercept. A(INTCEP + I) contains the coefficient of the I-th basis function.

SSE — Sum of squares of the errors.   (Output)

Comments

1. Automatic workspace usage is

   FNLSQ (INTCEP + NBASIS)**2 + 4 * (INTCEP + NBASIS) + IWT + 1 units, or

   DFNLSQ 2 * (INTCEP +NBASIS)**2 + 8 * (INTCEP + NBASIS) + 2 * IWT + 2 units.

   Workspace may be explicitly provided, if desired, by use of F2LSQ/DF2LSQ. The reference is

   ```
   CALL F2LSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA,
                IWT, WEIGHT, A, SSE, WK)
   ```

   The additional argument is

   WK — Work vector of length (INTCEP + NBASIS)**2 + 4 * (INTCEP + NBASIS) + IWT + 1. On output, the first (INTCEP + NBASIS)**2 elements of WK contain the R matrix from a QR decomposition of the matrix containing a column of ones (if INTCEP = 1) and the evaluated basis functions in columns INTCEP + 1 through INTCEP + NBASIS.

2. Informational errors

Type	Code	
3	1	Linear dependence of the basis functions exists. One or more components of `A` are set to zero.
3	2	Linear dependence of the constant function and basis functions exists. One or more components of `A` are set to zero.
4	1	Negative weight encountered.

Algorithm

The routine `FNLSQ` computes a best least-squares approximation to given univariate data of the form

$$\left\{\left(x_i, f_i\right)\right\}_{i=1}^{N}$$

by M basis functions

$$\left\{F_j\right\}_{j=1}^{M}$$

(where M = `NBASIS`). In particular, if `INTCEP` = 0, this routine returns the error sum of squares `SSE` and the coefficients a which minimize

$$\sum_{i=1}^{N} w_i \left(f_i - \sum_{j=1}^{M} a_j F_j\left(x_i\right) \right)^2$$

where w = `WEIGHT`, N = `NDATA`, x = `XDATA`, and, f = `FDATA`.

If `INTCEP` = 1, then an intercept is placed in the model; and the coefficients a, returned by `FNLSQ`, minimize the error sum of squares as indicated below.

$$\sum_{i=1}^{N} w_i \left(f_i - a_1 - \sum_{j=1}^{M} a_{j+1} F_j\left(x_i\right) \right)^2$$

That is, the first element of the vector a is now the coefficient of the function that is identically 1 and the coefficients of the F_j's are now a_{j+1}.

One additional parameter in the calling sequence for `FNLSQ` is `IWT`. If `IWT` is set to 0, then $w_i = 1$ is assumed. If `IWT` is set to 1, then the user must supply the weights.

Example

In this example, we fit the following two functions (indexed by δ)

$$1 + \sin x + 7 \sin 3x + \delta\varepsilon$$

where ε is random uniform deviate over the range [–1, 1], and δ is 0 for the first function and 1 for the second. These functions are evaluated at 90 equally spaced points on the interval [0, 6]. We use 4 basis functions, sin *kx* for *k* = 1, …, 4, with and without the intercept.

```
      INTEGER    NBASIS, NDATA
      PARAMETER  (NBASIS=4, NDATA=90)
C
      INTEGER    I, INTCEP, IWT, NOUT
      REAL       A(NBASIS+1), F, FDATA(NDATA), FLOAT, G, RNOISE,
     &           RNUNF, SIN, SSE, WEIGHT(NDATA), X, XDATA(NDATA)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   F, FNLSQ, RNSET, RNUNF, UMACH
C
      G(X) = 1.0 + SIN(X) + 7.0*SIN(3.0*X)
C                                 Set random number seed
      CALL RNSET (1234579)
C                                 Set up data values
      DO 10  I=1, NDATA
         XDATA(I) = 6.0*(FLOAT(I-1)/FLOAT(NDATA-1))
         FDATA(I) = G(XDATA(I))
   10 CONTINUE
      INTCEP = 0
      IWT    = 0
C                                 Compute least squares fit with no
C                                 intercept
      CALL FNLSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA, IWT, WEIGHT,
     &            A, SSE)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99996)
C                                 Write output
      WRITE (NOUT,99999) SSE, (A(I),I=1,NBASIS)
C
      INTCEP = 1
C                                 Compute least squares fit with
C                                 intercept
      CALL FNLSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA, IWT, WEIGHT,
     &            A, SSE)
C                                 Write output
      WRITE (NOUT,99998) SSE, A(1), (A(I),I=2,NBASIS+1)
C                                 Introduce noise
      DO 20  I=1, NDATA
         RNOISE   = 2.0*RNUNF() - 1.0
         FDATA(I) = FDATA(I) + RNOISE
   20 CONTINUE
      INTCEP = 0
      IWT    = 0
C                                 Compute least squares fit with no
C                                 intercept
      CALL FNLSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA, IWT, WEIGHT,
     &            A, SSE)
C                                 Write heading
      WRITE (NOUT,99997)
C                                 Write output
      WRITE (NOUT,99999) SSE, (A(I),I=1,NBASIS)
C
      INTCEP = 1
C                                 Compute least squares fit with
C                                 intercept
```

```
      CALL FNLSQ (F, INTCEP, NBASIS, NDATA, XDATA, FDATA, IWT, WEIGHT,
     &            A, SSE)
C                                Write output
      WRITE (NOUT,99998) SSE, A(1), (A(I),I=2,NBASIS+1)
C
99996 FORMAT (//, ' Without error introduced we have :', /,
     &       '    SSE          Intercept      Coefficients  ', /)
99997 FORMAT (//, ' With error introduced we have :', /, '    SSE      '
     &       , '     Intercept      Coefficients  ', /)
99998 FORMAT (1X, F8.4, 5X, F9.4, 5X, 4F9.4, /)
99999 FORMAT (1X, F8.4, 14X, 5X, 4F9.4, /)
      END
      REAL FUNCTION F (K, X)
      INTEGER    K
      REAL       X
C
      REAL       SIN
      INTRINSIC  SIN
C
      F = SIN(K*X)
      RETURN
      END
```

Output

```
Without error introduced we have :
SSE          Intercept      Coefficients

89.8776                      1.0101   0.0199   7.0291   0.0374
 0.0000        1.0000        1.0000   0.0000   7.0000   0.0000

With error introduced we have :
SSE          Intercept      Coefficients

112.4662                     0.9963  -0.0675   6.9825   0.0133
 30.9831       0.9522        0.9867  -0.0864   6.9548  -0.0223
```

BSLSQ/DBSLSQ (Single/Double precision)

Compute the least-squares spline approximation, and return the B-spline coefficients.

Usage

```
CALL BSLSQ (NDATA, XDATA, FDATA, WEIGHT, KORDER, XKNOT,
            NCOEF, BSCOEF)
```

Arguments

NDATA — Number of data points. (Input)

XDATA — Array of length `NDATA` containing the data point abscissas. (Input)

FDATA — Array of length `NDATA` containing the data point ordinates. (Input)

WEIGHT — Array of length `NDATA` containing the weights. (Input)

KORDER — Order of the spline. (Input)
KORDER must be less than or equal to NDATA.

XKNOT — Array of length NCOEF + KORDER containing the knot sequence. (Input)
XKNOT must be nondecreasing.

NCOEF — Number of B-spline coefficients. (Input)
NCOEF cannot be greater than NDATA.

BSCOEF — Array of length NCOEF containing the B-spline coefficients. (Output)

Comments

1. Automatic workspace usage is

   BSLSQ 4 * NDATA + (3 + NCOEF) * KORDER units, or

   DBSLSQ 7 * NDATA + 2 * (3 + NCOEF) * KORDER units.

   Workspace may be explicitly provided, if desired, by use of B2LSQ/DB2LSQ. The reference is

   ```
   CALL B2LSQ (NDATA, XDATA, FDATA, WEIGHT, KORDER,
               XKNOT, NCOEF, BSCOEF, WK1, WK2, WK3,
               WK4, IWK)
   ```

   The additional arguments are as follows:

   WK1 — Work array of length (3 + NCOEF) * KORDER.

   WK2 — Work array of length NDATA.

   WK3 — Work array of length NDATA.

   WK4 — Work array of length NDATA.

   IWK — Work array of length NDATA.

2. Informational errors

Type	Code	
4	5	Multiplicity of the knots cannot exceed the order of the spline.
4	6	The knots must be nondecreasing.
4	7	All weights must be greater than zero.
4	8	The smallest element of the data point array must be greater than or equal to the KORDth knot.
4	9	The largest element of the data point array must be less than or equal to the (NCOEF + 1)st knot.

3. The B-spline representation can be evaluated using BSVAL (page 469), and its derivative can be evaluated using BSDER (page 471).

Algorithm

The routine BSLSQ is based on the routine L2APPR by de Boor (1978, page 255). The IMSL routine BSLSQ computes a weighted discrete L_2 approximation from a spline subspace to a given data set (x_i, f_i) for $i = 1, \ldots, N$ (where N = NDATA). In other words, it finds B-spline coefficients, a = BSCOEF, such that

$$\sum_{i=1}^{N} \left| f_i - \sum_{j=1}^{m} a_j B_j(x_i) \right|^2 w_i$$

is a minimum, where m = NCOEF and B_j denotes the j-th B-spline for the given order, KORDER, and knot sequence, XKNOT. This linear least squares problem is solved by computing and solving the normal equations. While the normal equations can sometimes cause numerical difficulties, their use here should not cause a problem because the B-spline basis generally leads to well-conditioned banded matrices.

The choice of weights depends on the problem. In some cases, there is a natural choice for the weights based on the relative importance of the data points. To approximate a continuous function (if the location of the data points can be chosen), then the use of Gauss quadrature weights and points is reasonable. This follows because BSLSQ is minimizing an approximation to the integral

$$\int |F - s|^2 dx$$

The Gauss quadrature weights and points can be obtained using the IMSL routine GQRUL (page 621).

Example

In this example, we try to recover a quadratic polynomial using a quadratic spline with one interior knot from two different data sets. The first data set is generated by evaluating the quadratic at 50 equally spaced points in the interval (0, 1) and then adding uniformly distributed noise to the data. The second data set includes the first data set, and, additionally, the values at 0 and at 1 with no noise added. Since the first and last data points are uncontaminated by noise, we have chosen weights equal to 10^5 for these two points in this second problem. The quadratic, the first approximation, and the second approximation are then evaluated at 11 equally spaced points. This example illustrates the use of the weights to enforce interpolation at certain of the data points.

```
      INTEGER    KORDER, NCOEF
      PARAMETER  (KORDER=3, NCOEF=4)
C
      INTEGER    I, NDATA, NOUT
      REAL       ABS, BSCOF1(NCOEF), BSCOF2(NCOEF), BSVAL, F,
     &           FDATA1(50), FDATA2(52), FLOAT, RNOISE, RNUNF, S1,
     &           S2, WEIGHT(52), X, XDATA1(50), XDATA2(52),
     &           XKNOT(KORDER+NCOEF), XT, YT
      INTRINSIC  ABS, FLOAT
      EXTERNAL   BSLSQ, BSVAL, RNSET, RNUNF, SCOPY, UMACH
C
      DATA WEIGHT/52*1.0/
C                                  Define function
      F(X) = 8.0*X*(1.0-X)
C                                  Set random number seed
      CALL RNSET (12345679)
      NDATA = 50
C                                  Set up interior knots
      DO 10  I=1, NCOEF - KORDER + 2
         XKNOT(I+KORDER-1) = FLOAT(I-1)/FLOAT(NCOEF-KORDER+1)
   10 CONTINUE
C                                  Stack knots
      DO 20  I=1, KORDER - 1
         XKNOT(I) = XKNOT(KORDER)
         XKNOT(I+NCOEF+1) = XKNOT(NCOEF+1)
   20 CONTINUE
C                                  Set up data points excluding
C                                  the endpoints 0 and 1.
C                                  The function values have noise
C                                  introduced.
      DO 30  I=1, NDATA
         XDATA1(I) = FLOAT(I)/51.0
         RNOISE    = (RNUNF()-0.5)
         FDATA1(I) = F(XDATA1(I)) + RNOISE
   30 CONTINUE
C                                  Compute least squares B-spline
C                                  representation.
      CALL BSLSQ (NDATA, XDATA1, FDATA1, WEIGHT, KORDER, XKNOT, NCOEF,
     &            BSCOF1)
C                                  Now use same XDATA values but with
C                                  the endpoints included.  These
C                                  points will have large weights.
      NDATA = 52
      CALL SCOPY (50, XDATA1, 1, XDATA2(2), 1)
      CALL SCOPY (50, FDATA1, 1, FDATA2(2), 1)
C
      WEIGHT(1) = 1.0E5
      XDATA2(1) = 0.0
      FDATA2(1) = F(XDATA2(1))
      WEIGHT(NDATA) = 1.0E5
      XDATA2(NDATA) = 1.0
      FDATA2(NDATA) = F(XDATA2(NDATA))
C                                  Compute least squares B-spline
C                                  representation.
      CALL BSLSQ (NDATA, XDATA2, FDATA2, WEIGHT, KORDER, XKNOT, NCOEF,
     &            BSCOF2)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99998)
C                                  Print the two interpolants
C                                  at 11 points.
      DO 40  I=1, 11
         XT = FLOAT(I-1)/10.0
         YT = F(XT)
C                                  Evaluate splines
         S1 = BSVAL(XT,KORDER,XKNOT,NCOEF,BSCOF1)
         S2 = BSVAL(XT,KORDER,XKNOT,NCOEF,BSCOF2)
         WRITE (NOUT,99999) XT, YT, S1, S2, (S1-YT), (S2-YT)
   40 CONTINUE
C
99998 FORMAT (7X, ’X’, 9X, ’F(X)’, 6X, ’S1(X)’, 5X, ’S2(X)’, 7X,
     &       ’F(X)-S1(X)’, 7X, ’F(X)-S2(X)’)
99999 FORMAT (’ ’, 4F10.4, 4X, F10.4, 7X, F10.4)
      END
```

Output

```
   X         F(X)      S1(X)     S2(X)       F(X)-S1(X)       F(X)-S2(X)
0.0000    0.0000    0.0515    0.0000       0.0515           0.0000
0.1000    0.7200    0.7594    0.7490       0.0394           0.0290
0.2000    1.2800    1.3142    1.3277        0.0342          0.0477
0.3000    1.6800    1.7158    1.7362       0.0358           0.0562
0.4000    1.9200    1.9641    1.9744       0.0441           0.0544
0.5000    2.0000    2.0593    2.0423       0.0593           0.0423
0.6000    1.9200    1.9842    1.9468       0.0642           0.0268
0.7000    1.6800    1.7220    1.6948       0.0420           0.0148
0.8000    1.2800    1.2726    1.2863      -0.0074           0.0063
0.9000    0.7200    0.6360    0.7214      -0.0840           0.0014
1.0000    0.0000   -0.1878    0.0000      -0.1878           0.0000
```

BSVLS/DBSVLS (Single/Double precision)

Compute the variable knot B-spline least squares approximation to given data.

Usage

```
CALL BSVLS (NDATA, XDATA, FDATA, WEIGHT, KORDER, NCOEF,
            XGUESS, XKNOT, BSCOEF, SSQ)
```

Arguments

NDATA — Number of data points.   (Input)
NDATA must be at least 2.

XDATA — Array of length NDATA containing the data point abscissas.   (Input)

FDATA — Array of length NDATA containing the data point ordinates.   (Input)

WEIGHT — Array of length NDATA containing the weights.   (Input)

KORDER — Order of the spline.   (Input)
KORDER must be less than or equal to NDATA.

NCOEF — Number of B-spline coefficients.   (Input)
NCOEF must be less than or equal to NDATA.

XGUESS — Array of length NCOEF + KORDER containing the initial guess of knots.   (Input)
XGUESS must be nondecreasing.

XKNOT — Array of length NCOEF + KORDER containing the (nondecreasing) knot sequence.   (Output)

BSCOEF — Array of length `NCOEF` containing the B-spline representation. (Output)

SSQ — The square root of the sum of the squares of the error. (Output)

Comments

1. Automatic workspace usage is

   `BSVLS` `NCOEF` * (6 + 2 * `KORDER`) + `KORDER` * (7 − `KORDER`) + 3 * `NDATA` + 3 + `NDATA` units, or

   `DBSVLS` 2 * (`NCOEF` * (6 + 2 * `KORDER`) + `KORDER` * (7 − `KORDER`) + 3 * `NDATA` 3) + `NDATA` units.

   Workspace may be explicitly provided, if desired, by use of `B2VLS`/`DB2VLS`. The reference is

```
CALL B2VLS (NDATA, XDATA, FDATA, WEIGHT, KORDER,
            NCOEF, XGUESS, XKNOT, BSCOEF, SSQ, IWK,
            WK)
```

   The additional arguments are as follows:

   IWK — Work array of length `NDATA`.

   WK — Work array of length `NCOEF` * (6 + 2 * `KORDER`) + `KORDER` * (7 − `KORDER`) + 3 * `NDATA` + 3.

2. Informational errors

Type	Code	
3	12	The knots found to be optimal are stacked more than `KORDER`. This indicates fewer knots will produce the same error sum of squares. The knots have been separated slightly.
4	9	The multiplicity of the knots in `XGUESS` cannot exceed the order of the spline.
4	10	`XGUESS` must be nondecreasing.

Algorithm

The routine `BSVLS` attempts to find the best placement of knots that will minimize the leastsquares error to given data by a spline of order k = `KORDER` with N = `NCOEF` coefficients. The user provides the order k of the spline and the number of coefficients N. For this problem to make sense, it is necessary that $N > k$. We then attempt to find the minimum of the functional

$$F(a,\mathbf{t}) = \sum_{i=1}^{M} w_i \left(f_i - \sum_{j=1}^{N} a_j B_{j,k,\mathrm{t}}\left(x_j\right) \right)^2$$

The user must provide the weights w = `WEIGHT`, the data x_i = `XDATA` and f_i = `FDATA`, and M = `NDATA`. The minimum is taken over all admissible knot sequences **t**.

The technique employed in `BSVLS` uses the fact that for a fixed knot sequence **t** the minimization in a is a linear least-squares problem that can be solved by calling the IMSL routine `BSLSQ` (page 543). Thus, we can think of our objective function F as a function of just **t** by setting

$$G(\mathbf{t}) = \min_a F(a, \mathbf{t})$$

A Gauss-Seidel (cyclic coordinate) method is then used to reduce the value of the new objective function G. In addition to this local method, there is a global heuristic built into the algorithm that will be useful if the data arise from a smooth function. This heuristic is based on the routine `NEWNOT` of de Boor (1978, pages 184 and 258−261).

The user must input an initial guess, $\mathbf{t}^g$ = `XGUESS`, for the knot sequence. This guess must be a *valid* knot sequence for the splines of order k with

$$\mathbf{t}_1^g \le \ldots \le \mathbf{t}_k^g \le x_i \le \mathbf{t}_{N+1}^g \le \ldots \le \mathbf{t}_{N+k}^g, \qquad i = 1, \ldots, M$$

with $\mathbf{t}^g$ nondecreasing, and

$$\mathbf{t}_i^g < \mathbf{t}_{i+k}^g \quad i = 1, \ldots, N$$

The routine `BSVLS` returns the B-spline representation of the best fit found by the algorithm as well as the square root of the sum of squares error in `SSQ`. If this answer is unsatisfactory, you may reinitialize `BSVLS` with the return from `BSVLS` to see if an improvement will occur. We have found that this option does not usually (substantially) improve the result. In regard to execution speed, this routine can be several orders of magnitude slower than one call to the least-squares routine `BSLSQ`.

Example

In this example, we try to fit the function $|x - .33|$ evaluated at 100 equally spaced points on [0, 1]. We first use quadratic splines with 2 interior knots initially at .2 and .8. The eventual error should be zero since the function is a quadratic spline with two knots stacked at .33. As a second example, we try to fit the same data with cubic splines with three interior knots initially located at .1, .2, and, .5. Again, the theoretical error is zero when the three knots are stacked at .33.

We include a graph of the initial least-squares fit using the IMSL routine `BSLSQ` (page 543) for the above quadratic spline example with knots at .2 and .8. This graph overlays the graph of the spline computed by `BSVLS`, which is indistinguishable from the data.

```
INTEGER    KORD1, KORD2, NCOEF1, NCOEF2, NDATA
PARAMETER  (KORD1=3, KORD2=4, NCOEF1=5, NCOEF2=7, NDATA=100)
```

```
C
      INTEGER    I, NOUT
      REAL       ABS, BSCOEF(NCOEF2), F, FDATA(NDATA), FLOAT, SSQ,
     &           WEIGHT(NDATA), X, XDATA(NDATA), XGUES1(NCOEF1+KORD1),
     &           XGUES2(KORD2+NCOEF2), XKNOT(NCOEF2+KORD2)
      INTRINSIC  ABS, FLOAT
      EXTERNAL   BSVLS, UMACH
C
      DATA XGUES1/3*0.0, .2, .8, 3*1.0001/
      DATA XGUES2/4*0.0, .1, .2, .5, 4*1.0001/
      DATA WEIGHT/NDATA*.01/
C                                 Define function
      F(X) = ABS(X-.33)
C                                 Set up data
      DO 10  I=1, NDATA
         XDATA(I) = FLOAT(I-1)/FLOAT(NDATA)
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Compute least squares B-spline
C                                 representation with KORD1, NCOEF1,
C                                 and XGUES1.
      CALL BSVLS (NDATA, XDATA, FDATA, WEIGHT, KORD1, NCOEF1, XGUES1,
     &            XKNOT, BSCOEF, SSQ)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print heading
      WRITE (NOUT,99998) 'quadratic'
C                                 Print SSQ and the knots
      WRITE (NOUT,99999) SSQ, (XKNOT(I),I=1,KORD1+NCOEF1)
C                                 Compute least squares B-spline
C                                 representation with KORD2, NCOEF2,
C                                 and XGUES2.
      CALL BSVLS (NDATA, XDATA, FDATA, WEIGHT, KORD2, NCOEF2, XGUES2,
     &            XKNOT, BSCOEF, SSQ)
C                                 Print SSQ and the knots
      WRITE (NOUT,99998) 'cubic'
      WRITE (NOUT,99999) SSQ, (XKNOT(I),I=1,KORD2+NCOEF2)
C
99998 FORMAT (' Piecewise ', A, /)
99999 FORMAT (' Square root of the sum of squares : ', F9.4, /,
     &       ' Knot sequence : ', /, 1X, 11(F9.4,/,1X))
      END
```

Output

```
Piecewise quadratic

Square root of the sum of squares :    0.0008
Knot sequence :
   0.0000
   0.0000
   0.0000
   0.3137
   0.3464
   1.0001
   1.0001
   1.0001

Piecewise cubic
```

```
Square root of the sum of squares :    0.0005
Knot sequence :
   0.0000
   0.0000
   0.0000
   0.0000
   0.3167
   0.3273
   0.3464
   1.0001
   1.0001
   1.0001
   1.0001
```

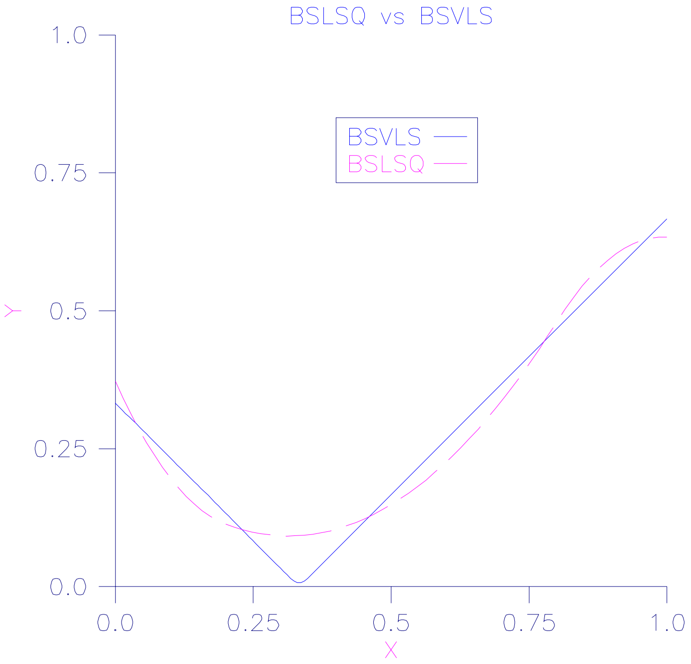


Figure 3-7   BSVLS vs. BSLSQ

CONFT/DCONFT (Single/Double precision)

Compute the least-squares constrained spline approximation, returning the B-spline coefficients.

Usage

```
CALL CONFT (NDATA, XDATA, FDATA, WEIGHT, NXVAL, XVAL,
            NHARD, IDER, ITYPE, BL, BU, KORDER, XKNOT,
            NCOEF, BSCOEF)
```

Arguments

NDATA — Number of data points.   (Input)

XDATA — Array of length `NDATA` containing the data point abscissas.   (Input)

FDATA — Array of size `NDATA` containing the values to be approximated. (Input)
`FDATA(I)` contains the value at `XDATA(I)`.

WEIGHT — Array of length `NDATA` containing the weights.   (Input)

NXVAL — Number of points in the vector `XVAL`.   (Input)

XVAL — Array of length `NXVAL` containing the abscissas at which the fit is to be constrained.   (Input)

NHARD — Number of entries of `XVAL` involved in the 'hard' constraints. (Input)
Note: (0 ≤ `NHARD` ≤ `NXVAL`). Setting `NHARD` to zero always results in a fit, while setting `NHARD` to `NXVAL` forces all constraints to be met. The 'hard' constraints must be satisfied or else the routine signals failure. The 'soft' constraints need not be satisfied, but there will be an attempt to satisfy the 'soft' constraints. The constraints must be ordered in terms of priority with the most important constraints first. Thus, all of the 'hard' constraints must preceed the 'soft' constraints. If infeasibility is detected among the soft constraints, we satisfy (in order) as many of the soft constraints as possible.

IDER — Array of length `NXVAL` containing the derivative value of the spline that is to be constrained.   (Input)
If we want to constrain the integral of the spline over the closed interval (c, d), then we set `IDER(I)` = `IDER(I + 1)` = – 1 and `XVAL(I)` = c and `XVAL(I + 1)` = d. For consistency, we insist that `ITYPE(I)` = `ITYPE(I + 1)` `.GE.` 0 and c `.LE.` d. Note that every entry in `IDER` must be at least – 1.

ITYPE — Array of length `NXVAL` indicating the types of general constraints. (Input)

ITYPE(I)	I - th Constraint
1	$\mathrm{BL(I)} \; = \; f^{(d_i)}(x_i)$
2	$f^{(d_i)}(x_i) \le \mathrm{BU(I)}$
3	$f^{(d_i)}(x_i) \ge \mathrm{BL(I)}$
4	$\mathrm{BL(I)} =\le f^{(d_i)}(x_i) \le \mathrm{BU(I)}$
$(d_i = -1)1$	$\mathrm{BL(I)} = \int_c^d f(t)dt$
$(d_i = -1)2$	$\int_c^d f(t)dt \le \mathrm{BU(I)}$
$(d_i = -1)3$	$\int_c^d f(t)dt \ge \mathrm{BL(I)}$
$(d_i = -1)4$	$\mathrm{BL(I)} \le \int_c^d f(t)dt \le \mathrm{BU(I)}$
10	periodic end conditions
99	disregard this constraint

In order to set two point constraints, we must have `ITYPE(I)` = `ITYPE(I` + 1) and `ITYPE(I)` must be negative.

ITYPE(I)	I – th Contraint
–1	$\mathrm{BL(I)} = f^{(d_i)}(x_i) - f^{\left(d_{i+1}\right)}(x_{i+1})$
–2	$f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \le \mathrm{BU(I)}$
–3	$f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \ge \mathrm{BL(I)}$
–4	$\mathrm{BL(I)} \le f^{(d_i)}(x_i) - f^{(d_{i+1})}(x_{i+1}) \le \mathrm{BU(I)}$

BL — Array of length `NXVAL` containing the lower limit of the general constraints, if there is no lower limit on the `I`-th constraint, then `BL(I)` is not referenced. (Input)

BU — Array of length `NXVAL` containing the upper limit of the general constraints, if there is no upper limit on the `I`-th constraint, then `BU(I)` is not referenced; if there is no range constraint, `BL` and `BU` can share the same storage locations. (Input)
If the `I`-th constraint is an equality constraint, `BU(I)` is not referenced.

KORDER — Order of the spline. (Input)

XKNOT — Array of length `NCOEF` + `KORDER` containing the knot sequence. (Input) The entries of `XKNOT` must be nondecreasing.

NCOEF — Number of B-spline coefficients. (Input)

BSCOEF — Array of length NCOEF containing the B-spline coefficients. (Output)

Comments

1. Automatic workspace usage is

CONFT (5 * NCOEF * NCOEF + 23 * NCOEF)/2 + (KORDER + 1) * (2 * KORDER + 1) + (2 * NXVAL + KORDER) * (2 * NXVAL + KORDER + NCOEF + 32) + 3 * NDATA + 1 + 4 * NCOEF + NDATA + 30 * (2 * NXVAL + KORDER)

DCONFT 2 * ((5 * NCOEF * NCOEF + 23 * NCOEF)/2 + (KORDER + 1) * (2 * KORDER + 1) + (2 * NXVAL + KORDER) * (2 * NXVAL + KORDER + NCOEF + 32) + 3 * NDATA + 1) + 4 * NCOEF + NDATA + 30 * (2 * NXVAL + KORDER)

Workspace may be explicitly provided, if desired, by use of C2NFT/DC2NFT. The reference is

```
CALL C2NFT (NDATA, XDATA, FDATA, WEIGHT, NXVAL, XVAL,
            NHARD, IDER, ITYPE, BL, BU, KORDER, XKNOT,
            NCOEF, BSCOEF, H, G, A, RHS, WK, IPERM, IWK)
```

The additional arguments are as follows:

H — Work array of size NCOEF by NCOEF. Upon output, H contains the Hessian matrix of the objective function used in the call to QPROG (page 982).

G — Work array of size NCOEF. Upon output, G contains the coefficients of the linear term used in the call to QPROG.

A — Work array of size (2 * NXVAL + KORDER) by (NCOEF + 1). Upon output, A contains the constraint matrix used in the call QPROG. The last column of A is used to keep record of the original order of the constraints.

RHS — Work array of size 2 * NXVAL + KORDER . Upon output, RHS contains the right hand side of the constraint matrix *A* used in the call to QPROG.

WK — Work array of size (KORDER + 1) * (2 * KORDER + 1) + (3 * NCOEF * NCOEF + 13 * NCOEF)/2 + (2 * NXVAL + KORDER +30)*(2*NXVAL + KORDER) + NDATA + 1.

IPERM — Work array of size NXVAL. Upon output, IPERM contains the permutaion of the original constraints used to generate the matrix A.

IWK — Work array of size NDATA + 30 * (2 * NXVAL + KORDER) + 4 * NCOEF.

2. Informational errors

Type	Code	
3	11	Soft constraints had to be removed in order to get a fit.
4	12	Multiplicity of the knots cannot exceed the order of the spline.
4	13	The knots must be nondecreasing.

4	14	The smallest element of the data point array must be greater than or equal to the `KORD`-th knot.
4	15	The largest element of the data point array must be less than or equal to the (`NCOEF` + 1)st knot.
4	16	All weights must be greater than zero.
4	17	The hard constraints could not be met.
4	18	The abscissas of the constrained points must lie within knot interval.
4	19	The upperbound must be greater than or equal to the lowerbound for a range constaint.
4	20	The upper limit of integration must be greater than the lower limit of integration for constraints involving the integral of the approximation.

Algorithm

The routine `CONFT` produces a constrained, weighted least-squares fit to data from a spline subspace. Constraints involving one point, two points, or integrals over an interval are allowed. The types of constraints supported by the routine are of four types.

$$\begin{aligned} E_p[f] \quad &= f^{(j_p)}(y_p) \\ \text{or} \quad &= f^{(j_p)}(y_p) - f^{(j_{p+1})}(y_{p+1}) \\ \text{or} \quad &= \int_{y_p}^{y_{p+1}} f(t)dt \\ \text{or} \quad &= \text{periodic end conditions} \end{aligned}$$

An interval, I_p, (which may be a point, a finite interval , or semi-infinite interval) is associated with each of these constraints.

The input for this routine consists of several items, first, the data set (x_i, f_i) for $i = 1, \ldots, N$ (where N = `NDATA`), that is the data which is to be fit. Second, we have the weights to be used in the least squares fit (w = `WEIGHT`). The vector `XVAL` of length `NXVAL` contains the abscissas of the points involved in specifying the constraints. The algorithm tries to satisfy all the constraints, but if the constraints are inconsistent then it will drop constraints, in the reverse order specified, until either a consistent set of constraints is found or the “hard” constraints are determined to be inconsistent (the “hard” constraints are those involving `XVAL`(1), …, `XVAL`(`NHARD`)). Thus, the algorithm satisfies as many constraints as possible in the order specified by the user. In the case when constraints are dropped, the user will receive a message explaining how many constraints had to be dropped to obtain the fit. The next several arguments are related to the type of constraint and the constraint interval. The last four arguments determine the spline solution. The user chooses the spline subspace

(KORDER, XKNOT, and NCOEF), and the routine returns the B-spline coefficients in BSCOEF.

Let n_f denote the number of feasible constraints as described above. Then, the routine solves the problem.

$$\sum_{i=1}^{N} \left| f_i - \sum_{j=1}^{m} a_j B_j(x_i) \right|^2 w_i$$

$$\text{subject to} \qquad E_p \left[\sum_{j=1}^{m} a_j B_j \right] \in I_p \quad p = 1, \ldots, n_f$$

This linearly constrained least-squares problem is treated as a quadratic program and is solved by invoking the IMSL routine QPROG (page 959).

The choice of weights depends on the data uncertainty in the problem. In some cases, there is a natural choice for the weights based on the estimates of errors in the data points.

Determining feasibility of linear constraints is a numerically sensitive task. If you encounter difficulties, a quick fix would be to widen the constraint intervals I_p.

Example 1

This is a simple application of CONFT. We generate data from the function

$$\frac{x}{2} + \sin\left(\frac{x}{2}\right)$$

contaminated with random noise and fit it with cubic splines. The function is increasing so we would hope that our least-squares fit would also be increasing. This is not the case for the unconstrained least squares fit generated by BSLSQ (page 543). We then force the derivative to be greater than 0 at NXVAL = 15 equally spaced points and call CONFT. The resulting curve is monotone. We print the error for the two fits averaged over 100 equally spaced points.

```
      INTEGER    KORDER, NCOEF, NDATA, NXVAL
      PARAMETER  (KORDER=4, NCOEF=8, NDATA=15, NXVAL=15)
C
      INTEGER    I, IDER(NXVAL), ITYPE(NXVAL), NHARD, NOUT
      REAL       ABS, BL(NXVAL), BSCLSQ(NDATA), BSCNFT(NDATA), BSVAL,
     &           BU(NXVAL), ERRLSQ, ERRNFT, F1, FDATA(NDATA), FLOAT,
     &           GRDSIZ, RNUNF, SIN, WEIGHT(NDATA), X, XDATA(NDATA),
     &           XKNOT(KORDER+NDATA), XVAL(NXVAL)
      INTRINSIC  ABS, FLOAT, SIN
      EXTERNAL   BSLSQ, BSVAL, CONFT, RNSET, RNUNF, SSET, UMACH
C
      F1(X) = .5*X + SIN(.5*X)
C                                 Initialize random number generator
C                                 and get output unit number.
      CALL RNSET (234579)
      CALL UMACH (2, NOUT)
C                                 Set all weights to one.
      CALL SSET (NDATA, 1.0, WEIGHT, 1)
C                                 Compute original XDATA and FDATA
C                                 with random noise.
      GRDSIZ = 10.0
      DO 10  I=1, NDATA
         XDATA(I) = GRDSIZ*((FLOAT(I-1)/FLOAT(NDATA-1)))
         FDATA(I) = F1(XDATA(I)) + (RNUNF()-.5)
   10 CONTINUE
C                                 Compute knots
      DO 20  I=1, NCOEF - KORDER + 2
         XKNOT(I+KORDER-1) = GRDSIZ*((FLOAT(I-1)/FLOAT(NCOEF-KORDER+1))
     &                        )
   20 CONTINUE
      DO 30  I=1, KORDER - 1
         XKNOT(I) = XKNOT(KORDER)
         XKNOT(I+NCOEF+1) = XKNOT(NCOEF+1)
   30 CONTINUE
C
C                                 Compute BSLSQ fit.
      CALL BSLSQ (NDATA, XDATA, FDATA, WEIGHT, KORDER, XKNOT, NCOEF,
     &            BSCLSQ)
C                                 Construct the constraints for
C                                 CONFT.
      DO 40  I=1, NXVAL
         XVAL(I)  = GRDSIZ*FLOAT(I-1)/FLOAT(NXVAL-1)
         ITYPE(I) = 3
         IDER(I)  = 1
         BL(I)    = 0.0
   40 CONTINUE
C                                 Call CONFT
      NHARD = 0
      CALL CONFT (NDATA, XDATA, FDATA, WEIGHT, NXVAL, XVAL, NHARD,
     &            IDER, ITYPE, BL, BU, KORDER, XKNOT, NCOEF, BSCNFT)
C                                 Compute the average error
C                                 of 100 points in the interval.
      ERRLSQ = 0.0
      ERRNFT = 0.0
      DO 50  I=1, 100
         X      = GRDSIZ*FLOAT(I-1)/99.0
         ERRNFT = ERRNFT + ABS(F1(X)-BSVAL(X,KORDER,XKNOT,NCOEF,BSCNFT)
     &            )
         ERRLSQ = ERRLSQ + ABS(F1(X)-BSVAL(X,KORDER,XKNOT,NCOEF,BSCLSQ)
     &            )
   50 CONTINUE
C                                 Print results
      WRITE (NOUT,99998) ERRLSQ/100.0
      WRITE (NOUT,99999) ERRNFT/100.0
C
99998 FORMAT (’ Average error with BSLSQ fit:  ’, F8.5)
99999 FORMAT (’ Average error with CONFT fit:  ’, F8.5)
      END
```

Output

```
Average error with BSLSQ fit:   0.20250
Average error with CONFT fit:   0.14334
```

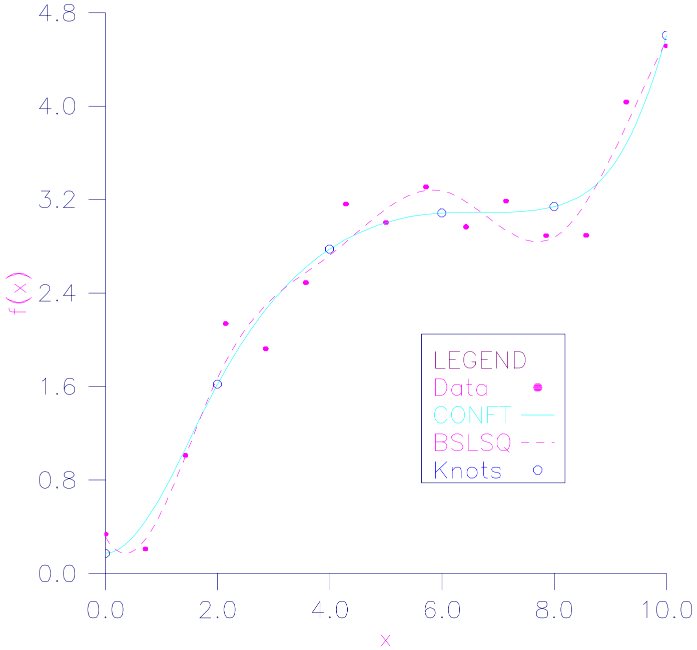


Figure 3-8   CONFT vs. BSLSQ Forcing Monotonicity

Example 2

We now try to recover the function

$$\frac{1}{1+x^4}$$

from noisy data. We first try the unconstrained least-squares fit using `BSLSQ` (page 543). Finding that fit somewhat unsatisfactory, we apply several constraints using `CONFT`. First, notice that the unconstrained fit oscillates through the true function at both ends of the interval. This is common for flat data. To remove this oscillation, we constrain the cubic spline to have zero second derivative at the first and last four knots. This forces the cubic spline to reduce to a linear polynomial on the first and last three knot intervals. In addition, we constrain the fit (which we will call *s*) as follows:

$$\begin{aligned} s(-7) &\geq 0 \\ \int_{-7}^{7} s(x)dx &\leq 2.3 \\ s(-7) &= s(7) \end{aligned}$$

Notice that the last constraint was generated using the periodic option (requiring only the zeroeth derivative to be periodic). We print the error for the two fits averaged over 100 equally spaced points.

```
      INTEGER    KORDER, NCOEF, NDATA, NXVAL
      PARAMETER  (KORDER=4, NCOEF=13, NDATA=51, NXVAL=12)
C
      INTEGER    I, IDER(NXVAL), ITYPE(NXVAL), NHARPT, NOUT
      REAL       ABS, BL(NXVAL), BSCLSQ(NDATA), BSCNFT(NDATA), BSVAL,
     &           BU(NXVAL), ERRLSQ, ERRNFT, F1, FDATA(NDATA), FLOAT,
     &           GRDSIZ, RNUNF, WEIGHT(NDATA), X, XDATA(NDATA),
     &           XKNOT(KORDER+NDATA), XVAL(NXVAL)
      INTRINSIC  ABS, FLOAT
      EXTERNAL   BSLSQ, BSVAL, CONFT, RNSET, RNUNF, SSET, UMACH
C
      F1(X) = 1.0/(1.0+X**4)
C                                 Initialize random number generator
C                                 and get output unit number.
      CALL UMACH (2, NOUT)
      CALL RNSET (234579)
C                                 Set all weights to one.
      CALL SSET (NDATA, 1.0, WEIGHT, 1)
C                                 Compute original XDATA and FDATA
C                                 with random noise.
      GRDSIZ = 14.0
      DO 10  I=1, NDATA
         XDATA(I) = GRDSIZ*((FLOAT(I-1)/FLOAT(NDATA-1))) - GRDSIZ/2.0
         FDATA(I) = F1(XDATA(I)) + 0.125*(RNUNF()-.5)
   10 CONTINUE
C                                 Compute KNOTS
      DO 20  I=1, NCOEF - KORDER + 2
         XKNOT(I+KORDER-1) = GRDSIZ*((FLOAT(I-1)/FLOAT(NCOEF-KORDER+1))
     &                       ) - GRDSIZ/2.0
   20 CONTINUE
      DO 30  I=1, KORDER - 1
         XKNOT(I) = XKNOT(KORDER)
         XKNOT(I+NCOEF+1) = XKNOT(NCOEF+1)
   30 CONTINUE
C                                 Compute BSLSQ fit
      CALL BSLSQ (NDATA, XDATA, FDATA, WEIGHT, KORDER, XKNOT, NCOEF,
     &            BSCLSQ)
C                                 Construct the constraints for
C                                 CONFT
      DO 40  I=1, 4
         XVAL(I)    = XKNOT(KORDER+I-1)
         XVAL(I+4)  = XKNOT(NCOEF-3+I)
         ITYPE(I)   = 1
         ITYPE(I+4) = 1
         IDER(I)    = 2
         IDER(I+4)  = 2
         BL(I)      = 0.0
         BL(I+4)    = 0.0
   40 CONTINUE
C
      XVAL(9)  = -7.0
      ITYPE(9) = 3
      IDER(9)  = 0
      BL(9)    = 0.0
C
```

```
      XVAL(10)  = -7.0
      ITYPE(10) = 2
      IDER(10)  = -1
      BU(10)    = 2.3
C
      XVAL(11)  = 7.0
      ITYPE(11) = 2
      IDER(11)  = -1
      BU(11)    = 2.3
C
      XVAL(12)  = -7.0
      ITYPE(12) = 10
      IDER(12)  = 0
C                                 Call CONFT
      CALL CONFT (NDATA, XDATA, FDATA, WEIGHT, NXVAL, XVAL, NHARPT,
     &            IDER, ITYPE, BL, BU, KORDER, XKNOT, NCOEF, BSCNFT)
C                                 Compute the average error
C                                 of 100 points in the interval.
      ERRLSQ = 0.0
      ERRNFT = 0.0
      DO 50  I=1, 100
         X      = GRDSIZ*FLOAT(I-1)/99.0 - GRDSIZ/2.0
         ERRNFT = ERRNFT + ABS(F1(X)-BSVAL(X,KORDER,XKNOT,NCOEF,BSCNFT)
     &            )
         ERRLSQ = ERRLSQ + ABS(F1(X)-BSVAL(X,KORDER,XKNOT,NCOEF,BSCLSQ)
     &            )
   50 CONTINUE
C                                 Print results
      WRITE (NOUT,99998) ERRLSQ/100.0
      WRITE (NOUT,99999) ERRNFT/100.0
C
99998 FORMAT (’ Average error with BSLSQ fit:  ’, F8.5)
99999 FORMAT (’ Average error with CONFT fit:  ’, F8.5)
      END
```

Output

```
Average error with BSLSQ fit:   0.01783
Average error with CONFT fit:   0.01339
```

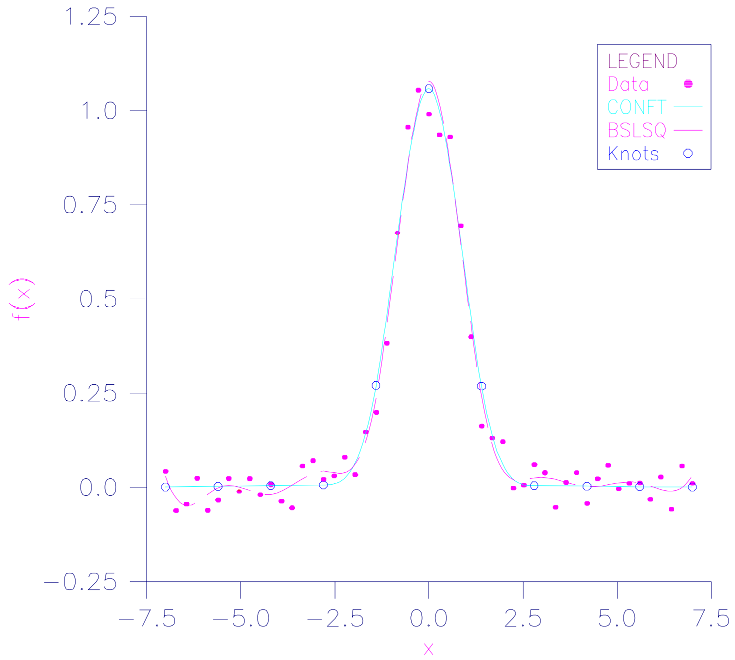


Figure 3-9   CONFT vs. BSLSQ Approximating $1/(1 + x^4)$

BSLS2/DBSLS2 (Single/Double precision)

Compute a two-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.

Usage

```
CALL BSLS2 (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF,
            KXORD, KYORD, XKNOT, YKNOT, NXCOEF, NYCOEF,
            XWEIGH, YWEIGH, BSCOEF)
```

Arguments

NXDATA — Number of data points in the `X`-direction.   (Input)

XDATA — Array of length `NXDATA` containing the data points in the `X`-direction. (Input)
`XDATA` must be nondecreasing.

NYDATA — Number of data points in the `Y`-direction.   (Input)

YDATA — Array of length `NYDATA` containing the data points in the `Y`-direction. (Input)
`YDATA` must be nondecreasing.

FDATA — Array of size NXDATA by NYDATA containing the values on the X – Y grid to be interpolated. (Input)
FDATA(I, J) contains the value at (XDATA(I), YDATA(I)).

LDF — Leading dimension of FDATA exactly as specified in the dimension statement of calling program. (Input)

KXORD — Order of the spline in the X-direction. (Input)

KYORD — Order of the spline in the Y-direction. (Input)

XKNOT — Array of length KXORD + NXCOEF containing the knots in the X-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length KYORD + NYCOEF containing the knots in the Y-direction. (Input)
YKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the X-direction. (Input)

NYCOEF — Number of B-spline coefficients in the Y-direction. (Input)

XWEIGH — Array of length NXDATA containing the positive weights of XDATA. (Input)

YWEIGH — Array of length NYDATA containing the positive weights of YDATA. (Input)

BSCOEF — Array of length NXCOEF * NYCOEF that contains the tensor product B-spline coefficients. (Output)
BSCOEF is treated internally as an array of size NXCOEF by NYCOEF.

Comments

1. Automatic workspace usage is

   BSLS2 (NXCOEF + 1) * NYDATA + KXORD * NXCOEF + KYORD * NYCOEF + 3 * MAX(KXOR, KYORD) units, or

   DBSLS2 2 * ((NXCOEF + 1) * NYDATA + KXORD * NXCOEF + KYORD * NYCOEF + 3 * MAX(KXORD, KYORD)) units.

Workspace may be explicitly provided, if desired, by use of B2LS2/DB2LS2. The reference is

```
CALL B2LS2 (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF,
            KXORD, KYORD, XKNOT, YKNOT, NXCOEF, NYCOEF,
            XWEIGH, YWEIGH, BSCOEF, WK)
```

The additional argument is

WK — Work array of length (NXCOEF + 1) * NYDATA + KXORD * NXCOEF + KYORD * NYCOEF + 3 * MAX(KXORD, KYORD).

2. Informational errors

Type	Code	
3	14	There may be less than one digit of accuracy in the least squares fit. Try using higher precision if possible.
4	5	Multiplicity of the knots cannot exceed the order of the spline.
4	6	The knots must be nondecreasing.
4	7	All weights must be greater than zero.
4	9	The data point abscissae must be nondecreasing.
4	10	The smallest element of the data point array must be greater than or equal to the `K_ORD`th knot.
4	11	The largest element of the data point array must be less than or equal to the (`N_COEF` + 1)st knot.

Algorithm

The routine `BSLS2` computes the coefficients of a tensor-product spline least-squares approximation to weighted tensor-product data. The input for this subroutine consists of data vectors to specify the tensor-product grid for the data, two vectors with the weights, the values of the surface on the grid, and the specification for the tensor-product spline. The grid is specified by the two vectors x = `XDATA` and y = `YDATA` of length n = `NXDATA` and m = `NYDATA`, respectively. A two-dimensional array f = `FDATA` contains the data values that are to be fit. The two vectors w_x = `XWEIGH` and w_y = `YWEIGH` contain the weights for the weighted least-squares problem. The information for the approximating tensor-product spline must also be provided. This information is contained in k_x = `KXORD`, $\mathbf{t}_x$ = `XKNOT`, and N = `NXCOEF` for the spline in the first variable, and in k_y = `KYORD` , $\mathbf{t}_y$ = `YKNOT` and M = `NYCOEF` for the spline in the second variable. The coefficients of the resulting tensor-product spline are returned in c = `BSCOEF`, which is an N * M array. The procedure computes coefficients by solving the normal equations in tensor-product form as discussed

in de Boor (1978, Chapter 17). The interested reader might also want to study the paper by E. Grosse (1980).

The final result produces coefficients c minimizing

$$\sum_{i=1}^{n}\sum_{j=1}^{m} w_x(i) w_y(j) \left[\sum_{k=1}^{N}\sum_{l=1}^{M} c_{kl} B_{kl}\left(x_i, y_j\right) - f_{ij} \right]^2$$

where the function B_{kl} is the tensor-product of two B-splines of order k_x and k_y. Specifically, we have

$$B_{kl}(x, y) = B_{k,k_x,\mathbf{t}_x}(x) B_{l,k_y,\mathbf{t}_y}(y)$$

The spline

$$\sum_{k=1}^{N}\sum_{l=1}^{M} c_{kl} B_{kl}$$

can be evaluated using BS2VL (page 479) and its partial derivatives can be evaluated using BS2DR (page 480).

Example

The data for this example arise from the function $e^x \sin(x + y) + \varepsilon$ on the rectangle [0, 3] × [0, 5]. Here, ε is a uniform random variable with range [−1, 1]. We sample this function on a 100 × 50 grid and then try to recover it by using cubic splines in the *x* variable and quadratic splines in the *y* variable. We print out the values of the function $e^x \sin(x + y)$ on a 3 × 5 grid and compare these values with the values of the tensor-product spline that was computed using the IMSL routine BSLS2.

```
      INTEGER    KXORD, KYORD, LDF, NXCOEF, NXDATA, NXVEC, NYCOEF,
     &           NYDATA, NYVEC
      PARAMETER  (KXORD=4, KYORD=3, NXCOEF=15, NXDATA=100, NXVEC=4,
     &           NYCOEF=7, NYDATA=50, NYVEC=6, LDF=NXDATA)
C
      INTEGER    I, J, NOUT
      REAL       BSCOEF(NXCOEF,NYCOEF), EXP, F, FDATA(NXDATA,NYDATA),
     &           FLOAT, RNOISE, RNUNF, SIN, VALUE(NXVEC,NYVEC), X,
     &           XDATA(NXDATA), XKNOT(NXCOEF+KXORD), XVEC(NXVEC),
     &           XWEIGH(NXDATA), Y, YDATA(NYDATA),
     &           YKNOT(NYCOEF+KYORD), YVEC(NYVEC), YWEIGH(NYDATA)
      INTRINSIC  EXP, FLOAT, SIN
      EXTERNAL   BS2GD, BSLS2, RNSET, RNUNF, SSET, UMACH
C                                  Define function
      F(X,Y) = EXP(X)*SIN(X+Y)
C                                  Set random number seed
      CALL RNSET (1234579)
C                                  Set up X knot sequence.
      DO 10  I=1, NXCOEF - KXORD + 2
         XKNOT(I+KXORD-1) = 3.0*(FLOAT(I-1)/FLOAT(NXCOEF-KXORD+1))
   10 CONTINUE
      XKNOT(NXCOEF+1) = XKNOT(NXCOEF+1) + 0.001
C                                  Stack knots.
      DO 20  I=1, KXORD - 1
         XKNOT(I) = XKNOT(KXORD)
         XKNOT(I+NXCOEF+1) = XKNOT(NXCOEF+1)
   20 CONTINUE
C                                  Set up Y knot sequence.
      DO 30  I=1, NYCOEF - KYORD + 2
         YKNOT(I+KYORD-1) = 5.0*(FLOAT(I-1)/FLOAT(NYCOEF-KYORD+1))
   30 CONTINUE
      YKNOT(NYCOEF+1) = YKNOT(NYCOEF+1) + 0.001
C                                  Stack knots.
      DO 40  I=1, KYORD - 1
         YKNOT(I) = YKNOT(KYORD)
         YKNOT(I+NYCOEF+1) = YKNOT(NYCOEF+1)
   40 CONTINUE
C                                  Set up X-grid.
      DO 50  I=1, NXDATA
```

```
         XDATA(I) = 3.0*(FLOAT(I-1)/FLOAT(NXDATA-1))
   50 CONTINUE
C                                 Set up Y-grid.
      DO 60  I=1, NYDATA
         YDATA(I) = 5.0*(FLOAT(I-1)/FLOAT(NYDATA-1))
   60 CONTINUE
C                                 Evaluate function on grid and
C                                 introduce random noise in [1,-1].
      DO 70  I=1, NYDATA
         DO 70  J=1, NXDATA
            RNOISE     = 2.0*RNUNF() - 1.0
            FDATA(J,I) = F(XDATA(J),YDATA(I)) + RNOISE
   70 CONTINUE
C                                 Set all weights equal to 1.
      CALL SSET (NXDATA, 1.0E0, XWEIGH, 1)
      CALL SSET (NYDATA, 1.0E0, YWEIGH, 1)
C                                 Compute least squares approximation.
      CALL BSLS2 (NXDATA, XDATA, NYDATA, YDATA, FDATA, LDF, KXORD,
     &            KYORD, XKNOT, YKNOT, NXCOEF, NYCOEF, XWEIGH, YWEIGH,
     &            BSCOEF)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Write heading
      WRITE (NOUT,99999)
C                                 Print interpolated values
C                                 on [0,3] x [0,5].
      DO 80  I=1, NXVEC
         XVEC(I) = FLOAT(I-1)
   80 CONTINUE
      DO 90  I=1, NYVEC
         YVEC(I) = FLOAT(I-1)
   90 CONTINUE
C                                 Evaluate spline
      CALL BS2GD (0, 0, NXVEC, XVEC, NYVEC, YVEC, KXORD, KYORD, XKNOT,
     &            YKNOT, NXCOEF, NYCOEF, BSCOEF, VALUE, NXVEC)
      DO 110  I=1, NXVEC
         DO 100  J=1, NYVEC
            WRITE (NOUT,’(5F15.4)’) XVEC(I), YVEC(J),
     &                             F(XVEC(I),YVEC(J)), VALUE(I,J),
     &                             (F(XVEC(I),YVEC(J))-VALUE(I,J))
  100    CONTINUE
  110 CONTINUE
99999 FORMAT (13X, ’X’, 14X, ’Y’, 10X, ’F(X,Y)’, 9X, ’S(X,Y)’, 10X,
     &       ’Error’)
      END
```

Output

```
     X              Y          F(X,Y)         S(X,Y)          Error
0.0000         0.0000          0.0000         0.2782        -0.2782
0.0000         1.0000          0.8415         0.7762         0.0653
0.0000         2.0000          0.9093         0.8203         0.0890
0.0000         3.0000          0.1411         0.1391         0.0020
0.0000         4.0000         -0.7568        -0.5705        -0.1863
0.0000         5.0000         -0.9589        -1.0290         0.0701
1.0000         0.0000          2.2874         2.2678         0.0196
1.0000         1.0000          2.4717         2.4490         0.0227
1.0000         2.0000          0.3836         0.4947        -0.1111
1.0000         3.0000         -2.0572        -2.0378        -0.0195
```

```
1.0000         4.0000        -2.6066        -2.6218         0.0151
1.0000         5.0000        -0.7595        -0.7274        -0.0321
2.0000         0.0000         6.7188         6.6923         0.0265
2.0000         1.0000         1.0427         0.8492         0.1935
2.0000         2.0000        -5.5921        -5.5885        -0.0035
2.0000         3.0000        -7.0855        -7.0955         0.0099
2.0000         4.0000        -2.0646        -2.1588         0.0942
2.0000         5.0000         4.8545         4.7339         0.1206
3.0000         0.0000         2.8345         2.5971         0.2373
3.0000         1.0000       -15.2008       -15.1079        -0.0929
3.0000         2.0000       -19.2605       -19.1698        -0.0907
3.0000         3.0000        -5.6122        -5.5820        -0.0302
3.0000         4.0000        13.1959        12.6659         0.5300
3.0000         5.0000        19.8718        20.5170        -0.6452
```

BSLS3/DBSLS3 (Single/Double precision)

Compute a three-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.

Usage

```
CALL BSLS3 (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA,
            FDATA, LDFDAT, MDFDAT, KXORD, KYORD, KZORD,
            XKNOT, YKNOT, ZKNOT, NXCOEF, NYCOEF, NZCOEF,
            XWEIGH, YWEIGH, ZWEIGH, BSCOEF)
```

Arguments

NXDATA — Number of data points in the *x*-direction. (Input)
NXDATA must be greater than or equal to NXCOEF.

XDATA — Array of length NXDATA containing the data points in the *x*-direction. (Input)
XDATA must be nondecreasing.

NYDATA — Number of data points in the *y*-direction. (Input)
NYDATA must be greater than or equal to NYCOEF.

YDATA — Array of length NYDATA containing the data points in the *y*-direction. (Input)
YDATA must be nondecreasing.

NZDATA — Number of data points in the *z*-direction. (Input)
NZDATA must be greater than or equal to NZCOEF.

ZDATA — Array of length NZDATA containing the data points in the *z*-direction. (Input)
ZDATA must be nondecreasing.

FDATA — Array of size NXDATA by NYDATA by NZDATA containing the values to be interpolated. (Input)
FDATA(I, J, K) contains the value at (XDATA(I), YDATA(J), ZDATA(K)).

LDFDAT — Leading dimension of FDATA exactly as specified in the dimension statement of the calling program. (Input)

MDFDAT — Second dimension of FDATA exactly as specified in the dimension statement of the calling program. (Input)

KXORD — Order of the spline in the *x*-direction. (Input)

KYORD — Order of the spline in the *y*-direction. (Input)

KZORD — Order of the spline in the *z*-direction. (Input)

XKNOT — Array of length KXORD + NXCOEF containing the knots in the *x*-direction. (Input)
XKNOT must be nondecreasing.

YKNOT — Array of length KYORD + NYCOEF containing the knots in the *y*-direction. (Input)
YKNOT must be nondecreasing.

ZKNOT — Array of length KZORD + NZCOEF containing the knots in the *z*-direction. (Input)
ZKNOT must be nondecreasing.

NXCOEF — Number of B-spline coefficients in the *x*-direction. (Input)

NYCOEF — Number of B-spline coefficients in the *y*-direction. (Input)

NZCOEF — Number of B-spline coefficients in the *z*-direction. (Input)

XWEIGH — Array of length NXDATA containing the positive weights of XDATA. (Input)

YWEIGH — Array of length NYDATA containing the positive weights of YDATA. (Input)

ZWEIGH — Array of length NZDATA containing the positive weights of ZDATA. (Input)

BSCOEF — Array of length NXCOEF*NYCOEF*NZCOEF that contains the tensor product B-spline coefficients. (Output)

Comments

1. Automatic workspace usage is

   BSLS3 NYCOEF * (NZDATA + KYORD + NZCOEF) + (NZDATA * (1 + NYDATA)NXCOEF * (KXORD + NYDATA * NZDATA) + KZORD * NZCOEF + 3 * MAX0(KXORD, KYORD, KZORD) units, or

   DBSLS3 2 * (NYCOEF * (NZDATA + KYORD + NZCOEF) + NZDATA * (1 + NYDATA) + NXCOEF * (KXORD + NYDATA * NZDATA) + KZORD * NZCOEF + 3 * MAX0(KXORD, KYORD, KZORD) units.

   Workspace may be explicitly provided, if desired, by use of B2LS3/DB2LS3. The reference is

```
CALL B2LS3 (NXDATA, XDATA, NYDATA, YDATA, FDATA,
            LDFDAT, KXORD, KYORD, XKNOT, YKNOT,
            NXCOEF, NYCOEF, XWEIGH, YWEIGH, BSCOEF,
            WK)
```

The additional argument is

WK — Work array of length NYCOEF * (NZDATA + KYORD + NZCOEF) + NZDATA * (1 + NYDATA) + NXCOEF * (KXORD + NYDATA * NZDATA) + KZORD * NZCOEF + 3 * MAX0(KXORD, KYORD, KZORD).

2. Informational errors

Type	Code	
3	13	There may be less than one digit of accuracy in the least squares fit. Try using higher precision if possible.
4	7	Multiplicity of knots cannot exceed the order of the spline.
4	8	The knots must be nondecreasing.
4	9	All weights must be greater than zero.
4	10	The data point abscissae must be nondecreasing.
4	11	The smallest element of the data point array must be greater than or equal to the K_ORDth knot.
4	12	The largest element of the data point array must be less than or equal to the (N_COEF + 1)st knot.

Algorithm

The routine BSLS3 computes the coefficients of a tensor-product spline least-squares approximation to weighted tensor-product data. The input for this subroutine consists of data vectors to specify the tensor-product grid for the data, three vectors with the weights, the values of the surface on the grid, and the specification for the tensor-product spline. The grid is specified by the three vectors x = XDATA, y = YDATA, and z = ZDATA of length k = NXDATA, l = NYDATA , and m = NYDATA, respectively. A three-dimensional array f = FDATA contains the data values which are to be fit. The three vectors w_x = XWEIGH, w_y = YWEIGH, and w_z = ZWEIGH contain the weights for the weighted least-squares problem. The information for the approximating tensor-product spline must also be provided. This information is contained in k_x = KXORD, $\mathbf{t}_x$ = XKNOT, and K = NXCOEF for the spline in the first variable, in k_y = KYORD, $\mathbf{t}_y$ = YKNOT and L = NYCOEF for the spline in the second variable, and in k_z = KZORD, $\mathbf{t}_z$ = ZKNOT and M = NZCOEF for the spline in the third variable.

The coefficients of the resulting tensor product spline are returned in c = BSCOEF, which is an $K \times L \times M$ array. The procedure computes coefficients by solving the normal equations in tensor-product form as discussed in de Boor (1978, Chapter 17). The interested reader might also want to study the paper by E. Grosse (1980).

The final result produces coefficients c minimizing

$$\sum_{i=l}^{k}\sum_{j=1}^{l}\sum_{p=1}^{m} w_x(i)w_y(j)w_z(p)\left[\sum_{s=1}^{K}\sum_{t=1}^{L}\sum_{u=1}^{M} c_{stu}B_{stu}\left(x_i, y_j, z_p\right) - f_{ijp}\right]^2$$

where the function B_{stu} is the tensor-product of three B-splines of order k_x, k_y, and k_z. Specifically, we have

$$B_{stu}(x, y, z) = B_{s,k_x,\mathbf{t}_x}(x)B_{t,k_y,\mathbf{t}_y}(y)B_{u,k_z,\mathbf{t}_z}(z)$$

The spline

$$\sum_{s=1}^{K}\sum_{t=1}^{L}\sum_{u=1}^{M} c_{stu}B_{stu}$$

can be evaluated at one point using `BS3VL` (page 490) and its partial derivatives can be evaluated using `BS3DR` (page 491). If the values on a grid are desired then we recommend `BS3GD` (page 495).

Example

The data for this example arise from the function $e^{(y-z)}\sin(x + y) + \varepsilon$ on the rectangle $[0, 3] \times [0, 2] \times [0, 1]$. Here, ε is a uniform random variable with range $[-.5, .5]$. We sample this function on a $4 \times 3 \times 2$ grid and then try to recover it by using tensor-product cubic splines in all variables. We print out the values of the function $e^{(y-z)}\sin(x + y)$ on a $4 \times 3 \times 2$ grid and compare these values with the values of the tensor-product spline that was computed using the IMSL routine `BSLS3`.

```
      INTEGER    KXORD, KYORD, KZORD, LDFDAT, MDFDAT, NXCOEF, NXDATA,
     &           NXVAL, NYCOEF, NYDATA, NYVAL, NZCOEF, NZDATA, NZVAL
      PARAMETER  (KXORD=4, KYORD=4, KZORD=4, NXCOEF=8, NXDATA=15,
     &           NXVAL=4, NYCOEF=8, NYDATA=15, NYVAL=3, NZCOEF=8,
     &           NZDATA=15, NZVAL=2, LDFDAT=NXDATA, MDFDAT=NYDATA)
C
      INTEGER    I, J, K, NOUT
      REAL       BSCOEF(NXCOEF,NYCOEF,NZCOEF), EXP, F,
     &           FDATA(NXDATA,NYDATA,NZDATA), FLOAT, RNOISE, RNUNF,
     &           SIN, SPXYZ(NXVAL,NYVAL,NZVAL), X, XDATA(NXDATA),
     &           XKNOT(NXCOEF+KXORD), XVAL(NXVAL), XWEIGH(NXDATA), Y,
     &           YDATA(NYDATA), YKNOT(NYCOEF+KYORD), YVAL(NYVAL),
     &           YWEIGH(NYDATA), Z, ZDATA(NZDATA),
     &           ZKNOT(NZCOEF+KZORD), ZVAL(NZVAL), ZWEIGH(NZDATA)
C    INTRINSIC  EXP,FLOAT,SIN
      INTRINSIC  EXP, FLOAT, SIN
      EXTERNAL   BS3GD, RNSET, RNUNF, SSET, UMACH, BSLS3
C                                 Define a function
      F(X,Y,Z) = EXP(Y-Z)*SIN(X+Y)
C
      CALL RNSET (1234579)
      CALL UMACH (2, NOUT)
C                                 Set up knot sequences
```

```
C                                 X-knots
      DO 10  I=1, NXCOEF - KXORD + 2
         XKNOT(I+KXORD-1) = 3.0*(FLOAT(I-1)/FLOAT(NXCOEF-KXORD+1))
   10 CONTINUE
      DO 20  I=1, KXORD - 1
         XKNOT(I) = XKNOT(KXORD)
         XKNOT(I+NXCOEF+1) = XKNOT(NXCOEF+1)
   20 CONTINUE
C                                 Y-knots
      DO 30  I=1, NYCOEF - KYORD + 2
         YKNOT(I+KYORD-1) = 2.0*(FLOAT(I-1)/FLOAT(NYCOEF-KYORD+1))
   30 CONTINUE
      DO 40  I=1, KYORD - 1
         YKNOT(I) = YKNOT(KYORD)
         YKNOT(I+NYCOEF+1) = YKNOT(NYCOEF+1)
   40 CONTINUE
C                                 Z-knots
      DO 50  I=1, NZCOEF - KZORD + 2
         ZKNOT(I+KZORD-1) = 1.0*(FLOAT(I-1)/FLOAT(NZCOEF-KZORD+1))
   50 CONTINUE
      DO 60  I=1, KZORD - 1
         ZKNOT(I) = ZKNOT(KZORD)
         ZKNOT(I+NZCOEF+1) = ZKNOT(NZCOEF+1)
   60 CONTINUE
C                                 Set up X-grid.
      DO 70  I=1, NXDATA
         XDATA(I) = 3.0*(FLOAT(I-1)/FLOAT(NXDATA-1))
   70 CONTINUE
C                                 Set up Y-grid.
      DO 80  I=1, NYDATA
         YDATA(I) = 2.0*(FLOAT(I-1)/FLOAT(NYDATA-1))
   80 CONTINUE
C                                 Set up Z-grid
      DO 90  I=1, NZDATA
         ZDATA(I) = 1.0*(FLOAT(I-1)/FLOAT(NZDATA-1))
   90 CONTINUE
C                                 Evaluate the function on the grid
C                                 and add noise.
      DO 100  I=1, NXDATA
         DO 100  J=1, NYDATA
            DO 100  K=1, NZDATA
               RNOISE = RNUNF() - 0.5
               FDATA(I,J,K) = F(XDATA(I),YDATA(J),ZDATA(K)) + RNOISE
  100 CONTINUE
C                                 Set all weights equal to 1.0
      CALL SSET (NXDATA, 1.0E0, XWEIGH, 1)
      CALL SSET (NYDATA, 1.0E0, YWEIGH, 1)
      CALL SSET (NZDATA, 1.0E0, ZWEIGH, 1)
C                                 Compute least-squares
      CALL BSLS3 (NXDATA, XDATA, NYDATA, YDATA, NZDATA, ZDATA, FDATA,
     &            LDFDAT, MDFDAT, KXORD, KYORD, KZORD, XKNOT, YKNOT,
     &            ZKNOT, NXCOEF, NYCOEF, NZCOEF, XWEIGH, YWEIGH,
     &            ZWEIGH, BSCOEF)
C                                 Set up grid for evaluation.
      DO 110  I=1, NXVAL
         XVAL(I) = FLOAT(I-1)
  110 CONTINUE
      DO 120  I=1, NYVAL
         YVAL(I) = FLOAT(I-1)
  120 CONTINUE
      DO 130  I=1, NZVAL
         ZVAL(I) = FLOAT(I-1)
  130 CONTINUE
C                                 Evaluate on the grid.
      CALL BS3GD (0, 0, 0, NXVAL, XVAL, NYVAL, YVAL, NZVAL, ZVAL,
     &            KXORD, KYORD, KZORD, XKNOT, YKNOT, ZKNOT, NXCOEF,
     &            NYCOEF, NZCOEF, BSCOEF, SPXYZ, NXVAL, NYVAL)
C                                 Print results.
      WRITE (NOUT,99998)
      DO 140  I=1, NXVAL
         DO 140  J=1, NYVAL
            DO 140  K=1, NZVAL
               WRITE (NOUT,99999) XVAL(I), YVAL(J), ZVAL(K),
     &                           F(XVAL(I),YVAL(J),ZVAL(K)),
     &                           SPXYZ(I,J,K), F(XVAL(I),YVAL(J),ZVAL(K)
     &                           ) - SPXYZ(I,J,K)
  140 CONTINUE
99998 FORMAT (8X, ’X’, 9X, ’Y’, 9X, ’Z’, 6X, ’F(X,Y,Z)’, 3X,
     &       ’S(X,Y,Z)’, 4X, ’Error’)
99999 FORMAT (’ ’, 3F10.3, 3F11.4)
      END
```

Output

```
   X         Y         Z      F(X,Y,Z)   S(X,Y,Z)    Error
0.000     0.000     0.000     0.0000     0.1987    -0.1987
0.000     0.000     1.000     0.0000     0.1447    -0.1447
0.000     1.000     0.000     2.2874     2.2854     0.0019
0.000     1.000     1.000     0.8415     1.0557    -0.2142
0.000     2.000     0.000     6.7188     6.4704     0.2484
0.000     2.000     1.000     2.4717     2.2054     0.2664
1.000     0.000     0.000     0.8415     0.8779    -0.0365
1.000     0.000     1.000     0.3096     0.2571     0.0524
1.000     1.000     0.000     2.4717     2.4015     0.0703
1.000     1.000     1.000     0.9093     0.8995     0.0098
1.000     2.000     0.000     1.0427     1.1330    -0.0902
1.000     2.000     1.000     0.3836     0.4951    -0.1115
2.000     0.000     0.000     0.9093     0.8269     0.0824
2.000     0.000     1.000     0.3345     0.3258     0.0087
2.000     1.000     0.000     0.3836     0.3564     0.0272
2.000     1.000     1.000     0.1411     0.1905    -0.0494
2.000     2.000     0.000    -5.5921    -5.5362    -0.0559
2.000     2.000     1.000    -2.0572    -1.9659    -0.0913
3.000     0.000     0.000     0.1411     0.4841    -0.3430
3.000     0.000     1.000     0.0519    -0.4257     0.4776
3.000     1.000     0.000    -2.0572    -1.9710    -0.0862
3.000     1.000     1.000    -0.7568    -0.8479     0.0911
3.000     2.000     0.000    -7.0855    -7.0957     0.0101
3.000     2.000     1.000    -2.6066    -2.1650    -0.4416
```

CSSED/DCSSED (Single/Double precision)

Smooth one-dimensional data by error detection.

Usage

```
CALL CSSED (NDATA, XDATA, FDATA, DIS, SC, MAXIT, SDATA)
```

Arguments

NDATA — Number of data points. (Input)

XDATA — Array of length NDATA containing the abscissas of the data points. (Input)

FDATA — Array of length NDATA containing the ordinates (function values) of the data points. (Input)

DIS — Proportion of the distance the ordinate in error is moved to its interpolating curve. (Input)
It must be in the range 0.0 to 1.0. A suggested value for DIS is one.

SC — Stopping criterion. (Input)
SC should be greater than or equal to zero. A suggested value for SC is zero.

MAXIT — Maximum number of iterations allowed. (Input)

SDATA — Array of length NDATA containing the smoothed data. (Output)

Comments

1. Automatic workspace usage is

   CSSED 6 * NDATA + 30 units, or
   DCSSED 10 * NDATA + 60 units.

   Workspace may be explicitly provided, if desired, by use of C2SED/DC2SED. The reference is

   ```
   CALL C2SED (NDATA, XDATA, FDATA, DIS, SC, MAXIT,
               SDATA, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 4 * NDATA + 30.

   IWK — Work array of length 2 * NDATA.

2. Informational error

Type	Code	
3	1	The maximum number of iterations allowed has been reached.

3. The arrays FDATA and SDATA may the the same.

Algorithm

The routine CSSED is designed to smooth a data set that is mildly contaminated with isolated errors. In general, the routine will not work well if more than 25% of the data points are in error. The routine CSSED is based on an algorithm of Guerra and Tapia (1974).

Setting NDATA = n, FDATA = f, SDATA = s and XDATA = x, the algorithm proceeds as follows. Although the user need not input an ordered XDATA sequence, we will assume that x is increasing for simplicity. The algorithm first sorts the XDATA values into an increasing sequence and then continues. A cubic spline interpolant is computed for each of the 6-point data sets (initially setting $s = f$)

$$(x_j, s_j) \quad j = i - 3, \ldots, i + 3\, j \neq i,$$

where $i = 4, \ldots, n - 3$ using CSAKM (page 432). For each i the interpolant, which we will call S_i, is compared with the current value of s_i, and a ‘point energy’ is computed as

$$pe_i = S_i(x_i) - s_i$$

Setting $sc = SC$, the algorithm terminates either if MAXIT iterations have taken place or if

$$\left|pe_i\right| \leq sc\left(x_{i+3} - x_{i-3}\right)/6 \qquad i = 4, \ldots, n-3$$

If the above inequality is violated for any i, then we update the i-th element of s by setting $s_i = s_i + d(pe_i)$, where d = DIS. Note that neither the first three nor the last three data points are changed. Thus, if these points are inaccurate, care must be taken to interpret the results.

The choice of the parameters d, sc and MAXIT are crucial to the successful usage of this subroutine. If the user has specific information about the extent of the contamination, then he should choose the parameters as follows: $d = 1$, $sc = 0$ and MAXIT to be the number of data points in error. On the other hand, if no such specific information is available, then choose $d = .5$, MAXIT $\leq 2n$, and

$$sc = .5\frac{\max s - \min s}{\left(x_n - x_1\right)}$$

In any case, we would encourage the user to experiment with these values.

Example

We take 91 uniform samples from the function $5 + (5 + t^2 \sin t)/t$ on the interval [1, 10]. Then, we contaminate 10 of the samples and try to recover the original function values.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=91)
C
      INTEGER    I, MAXIT, NOUT, ISUB(10)
      REAL       DIS, F, FDATA(91), SC, SDATA(91), SIN, X, XDATA(91),
     &           RNOISE(10)
      INTRINSIC  SIN
      EXTERNAL   CSSED, UMACH
C
      DATA ISUB/6, 17, 26, 34, 42, 49, 56, 62, 75, 83/
      DATA RNOISE/2.5, -3.0, -2.0, 2.5, 3.0, -2.0, -2.5, 2.0, -2.0, 3.0/
C
      F(X) = (X*X*SIN(X)+5.0)/X + 5.0
C                                  EX. #1; No specific information
C                                  available
      DIS   = 0.5
      SC    = 0.56
      MAXIT = 182
C                                  Set values for XDATA and FDATA
      XDATA(1) = 1.0
      FDATA(1) = F(XDATA(1))
      DO 10  I=2, NDATA
         XDATA(I) = XDATA(I-1) + .1
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Contaminate the data
      DO 20 I=1, 10
         FDATA(ISUB(I)) = FDATA(ISUB(I)) + RNOISE(I)
   20 CONTINUE
C                                  Smooth data
      CALL CSSED (NDATA, XDATA, FDATA, DIS, SC, MAXIT, SDATA)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99997)
C                                  Write data
      DO 30 I=1, 10
         WRITE (NOUT,99999) F(XDATA(ISUB(I))), FDATA(ISUB(I)),
     &                      SDATA(ISUB(I))
   30 CONTINUE
C                                  EX. #2; Specific information
C                                  available
      DIS   = 1.0
      SC    = 0.0
      MAXIT = 10
C                                  A warning message is produced
C                                  because the maximum number of
C                                  iterations is reached.
C
C                                  Smooth data
      CALL CSSED (NDATA, XDATA, FDATA, DIS, SC, MAXIT, SDATA)
C                                  Write heading
      WRITE (NOUT,99998)
C                                  Write data
      DO 40 I=1, 10
         WRITE (NOUT,99999) F(XDATA(ISUB(I))), FDATA(ISUB(I)),
     &                      SDATA(ISUB(I))
   40 CONTINUE
C
99997 FORMAT (’ Case A - No specific information available’, /,
     &       ’    F(X)       F(X)+NOISE          SDATA(X)’, /)
99998 FORMAT (’ Case B - Specific information available’, /,
     &       ’    F(X)       F(X)+NOISE          SDATA(X)’, /)
```

```
99999 FORMAT (' ', F7.3, 8X, F7.3, 11X, F7.3)
      END
```

Output

```
Case A - No specific information available
 F(X)       F(X)+NOISE         SDATA(X)

 9.830        12.330             9.870
 8.263         5.263             8.215
 5.201         3.201             5.168
 2.223         4.723             2.264
 1.259         4.259             1.308
 3.167         1.167             3.138
 7.167         4.667             7.131
10.880        12.880            10.909
12.774        10.774            12.708
 7.594        10.594             7.639

 *** WARNING  ERROR 1 from CSSED.  Maximum number of iterations limit MAXIT
 ***          =10 exceeded.  The best answer found is returned.
Case B - Specific information available
 F(X)       F(X)+NOISE         SDATA(X)

 9.830        12.330             9.831
 8.263         5.263             8.262
 5.201         3.201             5.199
 2.223         4.723             2.225
 1.259         4.259             1.261
 3.167         1.167             3.170
 7.167         4.667             7.170
10.880        12.880            10.878
12.774        10.774            12.770
 7.594        10.594             7.592
```

CSSMH/DCSSMH (Single/Double precision)

Compute a smooth cubic spline approximation to noisy data.

Usage

```
CALL CSSMH (NDATA, XDATA, FDATA, WEIGHT, SMPAR, BREAK,
            CSCOEF)
```

Arguments

NDATA — Number of data points. (Input)
NDATA must be at least 2.

XDATA — Array of length NDATA containing the data point abscissas. (Input)
XDATA must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

WEIGHT — Array of length NDATA containing estimates of the standard deviations of FDATA. (Input)
All elements of WEIGHT must be positive.

SMPAR — A nonnegative number which controls the smoothing. (Input)
The spline function S returned is such that the sum from I = 1 to NDATA of ((S(XDATA(I))FDATA(I)) / WEIGHT(I))**2 is less than or equal to SMPAR. It is recommended that SMPAR lie in the confidence interval of this sum, i.e., NDATA – SQRT(2 * NDATA).LE. SMPAR.LE. NDATA + SQRT(2 * NDATA).

BREAK — Array of length NDATA containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by NDATA containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   CSSMH 9 * NDATA + 5 units, or
   DCSSMH 17 * NDATA + 10 units.

   Workspace may be explicitly provided, if desired, by use of C2SMH/DC2SMH. The reference is

   ```
   CALL C2SMH (NDATA, XDATA, FDATA, WEIGHT, SMPAR,
               BREAK, CSCOEF, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 8 * NDATA + 5.

   IWK — Work array of length NDATA.

2. Informational errors

Type	Code	
3	1	The maximum number of iterations has been reached. The best approximation is returned.
4	3	All weights must be greater than zero.

3. The cubic spline can be evaluated using CSVAL (page 440); its derivative can be evaluated using CSDER (page 441).

Algorithm

The routine CSSMH is designed to produce a C^2 cubic spline approximation to a data set in which the function values are noisy. This spline is called a *smoothing spline*. It is a natural cubic spline with knots at all the data abscissas x = XDATA, but it does *not* interpolate the data (x_i, f_i). The smoothing spline S is the unique C^2 function which minimizes

$$\int_a^b S''(x)^2 \, dx$$

subject to the constraint

$$\sum_{i=1}^{N} \left| \frac{S(x_i) - f_i}{w_i} \right|^2 \le \sigma$$

where w = WEIGHT, σ = SMPAR is the smoothing parameter, and N = NDATA.

Recommended values for σ depend on the weights w. If an estimate for the standard deviation of the error in the value f_i is available, then w_i should be set to this value and the smoothing parameter σ should be chosen in the confidence interval corresponding to the left side of the above inequality. That is,

$$N - \sqrt{2N} \le \sigma \le N + \sqrt{2N}$$

The routine CSSMH is based on an algorithm of Reinsch (1967). This algorithm is also discussed in de Boor (1978, pages 235−243).

Example

In this example, function values are contaminated by adding a small “random” amount to the correct values. The routine CSSMH is used to approximate the original, uncontaminated data.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=300)
C
      INTEGER    I, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, ERROR, F,
     &           FDATA(NDATA), FLOAT, FVAL, RNUNF, SDEV, SMPAR, SQRT,
     &           SVAL, WEIGHT(NDATA), X, XDATA(NDATA), XT
      INTRINSIC  FLOAT, SQRT
      EXTERNAL   CSSMH, CSVAL, RNSET, RNUNF, SSET, UMACH
C
      F(X) = 1.0/(.1+(3.0*(X-1.0))**4)
C                                  Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = 3.0*(FLOAT(I-1)/FLOAT(NDATA-1))
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                  Set the random number seed
      CALL RNSET (1234579)
C                                  Contaminate the data
      DO 20  I=1, NDATA
         FDATA(I) = FDATA(I) + 2.0*RNUNF() - 1.0
   20 CONTINUE
C                                  Set the WEIGHT vector
      SDEV = 1.0/SQRT(3.0)
      CALL SSET (NDATA, SDEV, WEIGHT, 1)
      SMPAR = NDATA
C                                  Smooth the data
      CALL CSSMH (NDATA, XDATA, FDATA, WEIGHT, SMPAR, BREAK, CSCOEF)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Write heading
      WRITE (NOUT,99999)
```

```
C                                 Print 10 values of the function.
      DO 30  I=1, 10
         XT    = 90.0*(FLOAT(I-1)/FLOAT(NDATA-1))
C                                 Evaluate the spline
         SVAL  = CSVAL(XT,NDATA-1,BREAK,CSCOEF)
         FVAL  = F(XT)
         ERROR = SVAL - FVAL
         WRITE (NOUT,'(4F15.4)') XT, FVAL, SVAL, ERROR
   30 CONTINUE
C
99999 FORMAT (12X, 'X', 9X, 'Function', 7X, 'Smoothed', 10X,
     &       'Error')
      END
```

Output

```
     X         Function       Smoothed          Error
 0.0000         0.0123         0.1118         0.0995
 0.3010         0.0514         0.0646         0.0131
 0.6020         0.4690         0.2972        -0.1718
 0.9030         9.3312         8.7022        -0.6289
 1.2040         4.1611         4.7887         0.6276
 1.5050         0.1863         0.2718         0.0856
 1.8060         0.0292         0.1408         0.1116
 2.1070         0.0082         0.0826         0.0743
 2.4080         0.0031         0.0076         0.0045
 2.7090         0.0014        -0.1789        -0.1803
```

CSSCV/DCSSCV (Single/Double precision)

Compute a smooth cubic spline approximation to noisy data using cross-validation to estimate the smoothing parameter.

Usage

CALL CSSCV (NDATA, XDATA, FDATA, IEQUAL, BREAK, CSCOEF)

Arguments

NDATA — Number of data points. (Input)
NDATA must be at least 3.

XDATA — Array of length NDATA containing the data point abscissas. (Input)
XDATA must be distinct.

FDATA — Array of length NDATA containing the data point ordinates. (Input)

IEQUAL — A flag alerting the subroutine that the data is equally spaced. (Input)

BREAK — Array of length NDATA containing the breakpoints for the piecewise cubic representation. (Output)

CSCOEF — Matrix of size 4 by NDATA containing the local coefficients of the cubic pieces. (Output)

Comments

1. Automatic workspace usage is

   CSSCV 10 * NDATA + 14 units, or
   DCSSCV 19 * NDATA + 28 units.

   Workspace may be explicitly provided, if desired, by use of C2SCV/DC2SCV. The reference is

   ```
   CALL C2SCV (NDATA, XDATA, FDATA, IEQUAL, BREAK,
               CSCOEF, WK, SDWK, IPVT)
   ```

   The additional arguments are as follows:

   WK — Work array of length 7 * (NDATA + 2).

   SDWK — Work array of length 2 * NDATA.

   IPVT — Work array of length NDATA.

2. Informational error

Type	Code	
4	2	Points in the data point abscissas array, XDATA, must be distinct.

Algorithm

The routine CSSCV is designed to produce a C^2 cubic spline approximation to a data set in which the function values are noisy. This spline is called a *smoothing spline*. It is a natural cubic spline with knots at all the data abscissas x = XDATA, but it does *not* interpolate the data (x_i, f_i). The smoothing spline S_σ is the unique C^2 function that minimizes

$$\int_a^b S''_\sigma(x)^2 \; dx$$

subject to the constraint

$$\sum_{i=1}^{N} \left| S_\sigma(x_i) - f_i \right|^2 \le \sigma$$

where σ is the smoothing parameter and N = NDATA. The reader should consult Reinsch (1967) for more information concerning smoothing splines. The IMSL subroutine CSSMH (page 575) solves the above problem when the user provides the smoothing parameter σ. This routine attempts to find the ‘optimal’ smoothing parameter using the statistical technique known as cross-validation. This means that (in a very rough sense) one chooses the value of σ so that the smoothing spline (S_σ) best approximates the value of the data at x_i, if it is computed using all the data *except* the i-th; this is true for all i = 1, …, N. For more information on this topic, we refer the reader to Craven and Wahba (1979).

Example

In this example, function values are computed and are contaminated by adding a small “random” amount. The routine CSSCV is used to try to reproduce the original, uncontaminated data.

```
      INTEGER    NDATA
      PARAMETER  (NDATA=300)
C
      INTEGER    I, IEQUAL, NOUT
      REAL       BREAK(NDATA), CSCOEF(4,NDATA), CSVAL, ERROR, F,
     &           FDATA(NDATA), FLOAT, FVAL, RNUNF, SVAL, X,
     &           XDATA(NDATA), XT
      INTRINSIC  FLOAT
      EXTERNAL   CSSCV, CSVAL, RNSET, RNUNF, UMACH
C
      F(X) = 1.0/(.1+(3.0*(X-1.0))**4)
C
      CALL UMACH (2, NOUT)
C                                 Set up a grid
      DO 10  I=1, NDATA
         XDATA(I) = 3.0*(FLOAT(I-1)/FLOAT(NDATA-1))
         FDATA(I) = F(XDATA(I))
   10 CONTINUE
C                                 Introduce noise on [-.5,.5]
C                                 Contaminate the data
      CALL RNSET (1234579)
      DO 20  I=1, NDATA
         FDATA(I) = FDATA(I) + 2.0*RNUNF() - 1.0
   20 CONTINUE
C
C                                 Set IEQUAL=1 for equally spaced data
      IEQUAL = 1
C                                 Smooth data
      CALL CSSCV (NDATA, XDATA, FDATA, IEQUAL, BREAK, CSCOEF)
C                                 Print results
      WRITE (NOUT,99999)
      DO 30  I=1, 10
         XT    = 90.0*(FLOAT(I-1)/FLOAT(NDATA-1))
         SVAL  = CSVAL(XT,NDATA-1,BREAK,CSCOEF)
         FVAL  = F(XT)
         ERROR = SVAL - FVAL
         WRITE (NOUT,’(4F15.4)’) XT, FVAL, SVAL, ERROR
   30 CONTINUE
99999 FORMAT (12X, ’X’, 9X, ’Function’, 7X, ’Smoothed’, 10X,
     &       ’Error’)
      END
```

Output

```
    X         Function       Smoothed          Error
 0.0000         0.0123         0.2528         0.2405
 0.3010         0.0514         0.1054         0.0540
 0.6020         0.4690         0.3117        -0.1572
 0.9030         9.3312         8.9461        -0.3850
 1.2040         4.1611         4.6847         0.5235
 1.5050         0.1863         0.3819         0.1956
 1.8060         0.0292         0.1168         0.0877
 2.1070         0.0082         0.0658         0.0575
 2.4080          0.0031           0.0395           0.0364
 2.7090          0.0014          -0.2155          -0.2169
```

RATCH/DRATCH (Single/Double precision)

Compute a rational weighted Chebyshev approximation to a continuous function on an interval.

Usage

```
CALL RATCH (F, PHI, WEIGHT, A, B, N, M, P, Q, ERROR)
```

Arguments

F — User-supplied `FUNCTION` to be approximated. The form is `F(X)`, where
    `X` – Independent variable. (Input)
    `F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

PHI — User-supplied `FUNCTION` to supply the variable transformation which must be continuous and monotonic. The form is `PHI(X)`, where
    `X` – Independent variable. (Input)
    `PHI` – The function value. (Output)

`PHI` must be declared `EXTERNAL` in the calling program.
WEIGHT — User-supplied `FUNCTION` to scale the maximum error. It must be continuous and nonvanishing on the closed interval (`A`, `B`). The form is `WEIGHT(X)`, where
    `X` – Independent variable. (Input)
    `WEIGHT` – The function value. (Output)

`WEIGHT` must be declared `EXTERNAL` in the calling program.

A — Lower end of the interval on which the approximation is desired. (Input)

B — Upper end of the interval on which the approximation is desired. (Input)

N — The degree of the numerator. (Input)

M — The degree of the denominator. (Input)

P — Vector of length `N` + 1 containing the coefficients of the numerator polynomial. (Output)

Q — Vector of length `M` + 1 containing the coefficients of the denominator polynomial. (Output)

ERROR — Min-max error of approximation. (Output)

Comments

1. Automatic workspace usage is

   `RATCH` (`N` + `M` + 8) `*` (`N` + `M` + 2) + `N` + `M` + 2 units, or
   `DRATCH` 2 `*` (`N` + `M` + 8) `*` (`N` + `M` + 2) + `N` + `M` + 2 units.

Workspace may be explicitly provided, if desired, by use of `R2TCH/DR2TCH`. The reference is

```
CALL R2TCH (F, PHI, WEIGHT, A, B, N, M, P, Q, ERROR,
            ITMAX, IWK, WK)
```

The additional arguments are as follows:

ITMAX — Maximum number of iterations. (Input)
The default value is 20.

IWK — Workspace vector of length (`N` + `M` + 2). (Workspace)

WK — Workspace vector of length (`N` + `M` + 8) `*` (`N` + `M` + 2). (Workspace)

2. Informational errors

Type	Code	
3	1	The maximum number of iterations has been reached. The routine `R2TCH` may be called directly to set a larger value for `ITMAX`.
3	2	The error was reduced as far as numerically possible. A good approximation is returned in `P` and `Q`, but this does not necessarily give the Chebyshev approximation.
4	3	The linear system that defines `P` and `Q` was found to be algorithmically singular. This indicates the possibility of a degenerate approximation.
4	4	A sequence of critical points that was not monotonic generated. This indicates the possibility of a degenerate approximation.
4	5	The value of the error curve at some critical point is too large. This indicates the possibility of poles in the rational function.
4	6	The weight function cannot be zero on the closed interval (`A`, `B`).

Algorithm

The routine `RATCH` is designed to compute the best weighted L_∞ (Chebyshev) approximant to a given function. Specifically, given a weight function w = `WEIGHT`, a monotone function ϕ = `PHI`, and a function f to be approximated on the interval $[a, b]$, the subroutine `RATCH` returns the coefficients (in P and Q) for a rational approximation to f on $[a, b]$. The user must supply the degree of the numerator N and the degree of the denominator M of the rational function

$$R_M^N$$

The goal is to produce coefficients which minimize the expression

$$\left\| \frac{f - R_M^N}{w} \right\| := \max_{x \in [a,b]} \frac{\left| f(x) - \dfrac{\sum_{i=1}^{N+1} P_i \phi^{i-1}(x)}{\sum_{i=1}^{M+1} Q_i \phi^{i-1}(x)} \right|}{w(x)}$$

Notice that setting $\phi(x) = x$ yields ordinary rational approximation. A typical use of the function ϕ occurs when one wants to approximate an even function on a symmetric interval, say $[-a, a]$ using ordinary rational functions. In this case, it is known that the answer must be an even function. Hence, one can set $\phi(x) = x^2$, only approximate on $[0, a]$, and decrease by one half the degrees in the numerator and denominator.

The algorithm implemented in this subroutine is designed for fast execution. It assumes that the best approximant has precisely $N + M + 2$ equi-oscillations. That is, that there exist $N + M + 2$ points $\mathbf{t}_1 < \ldots < \mathbf{t}_{N+M+2}$ satisfying

$$e(\mathbf{t}_i) = -e(\mathbf{t}_{i+1}) = \pm \left\| \frac{f - R_M^N}{w} \right\|$$

Such points are called alternants. Unfortunately, there are many instances in which the best rational approximant to the given function has either fewer alternants or more alternants. In this case, it is not expected that this subroutine will perform well. For more information on rational Chebyshev approximation, the reader can consult Cheney (1966). The subroutine is based on work of Cody, Fraser, and Hart (1968).

Example

In this example, we compute the best rational approximation to the gamma function, Γ, on the interval [2, 3] with weight function $w = 1$ and $N = M = 2$. We display the maximum error and the coefficients. This problem is taken from the paper of Cody, Fraser, and Hart (1968). We compute in double precision due to the conditioning of this problem.

```
      INTEGER    M, N
      PARAMETER  (M=2, N=2)
C
      INTEGER    NOUT
      DOUBLE PRECISION  A, B, ERROR, F, P(N+1), PHI, Q(M+1), WEIGHT
      EXTERNAL   F, PHI, RATCH, UMACH, WEIGHT
C
      A = 2.0D0
      B = 3.0D0
C                                 Compute double precision rational
C                                 approximation
      CALL DRATCH (F, PHI, WEIGHT, A, B, N, M, P, Q, ERROR)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print P, Q and min-max error
      WRITE (NOUT,’(1X,A)’) ’In double precision we have:’
      WRITE (NOUT,99999) ’P      = ’, P
      WRITE (NOUT,99999) ’Q      = ’, Q
      WRITE (NOUT,99999) ’ERROR  = ’, ERROR
99999 FORMAT (’ ’, A, 5X, 3F20.12, /)
      END
C-----------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION F (X)
      DOUBLE PRECISION X
C
      DOUBLE PRECISION DGAMMA
      EXTERNAL   DGAMMA
C
      F = DGAMMA(X)
      RETURN
      END
C-----------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION PHI (X)
      DOUBLE PRECISION X
C
      PHI = X
      RETURN
      END
C-----------------------------------------------------------------------
C
      DOUBLE PRECISION FUNCTION WEIGHT (X)
      DOUBLE PRECISION X
C
      DOUBLE PRECISION DGAMMA
      EXTERNAL   DGAMMA
C
      WEIGHT = DGAMMA(X)
      RETURN
      END
```

Output

```
In double precision we have:
P      =           1.265583562487     -0.650585004466      0.197868699191

Q      =           1.000000000000     -0.064342721236     -0.028851461855

ERROR  =          -0.000026934190
```

Chapter 4: Integration and Differentiation

Routines

Usage Notes

Univariate Quadrature

The first nine routines described in this chapter are designed to compute approximations to integrals of the form

$$\int_a^b f(x)w(x)dx$$

The weight function *w* is used to incorporate known singularities (either algebraic or logarithmic), to incorporate oscillations, or to indicate that a Cauchy principal value is desired. For general purpose integration, we recommend the use of `QDAGS` (page 589) (even if no endpoint singularities are present). If more efficiency is desired, then the use of `QDAG` (page 591) (or `QDAG*`) should be considered. These routines are organized as follows:

- $w = 1$

  – `QDAGS`

  – `QDAG`

  – `QDAGP`

  – `QDAGI`

  – `QDNG`

- $w(x) = \sin \omega x$ or $w(x) = \cos \omega x$

  – `QDAWO` (for a finite interval)

  – `QDAWF` (for an infinite interval)

- $w(x) = (x - a)^{\alpha}(b - x)^{\beta} \ln(x - a) \ln(b - x)$, where the ln factors are optional

  – `QDAWS`

- $w(x) = 1/(x - c)$ Cauchy principal value

  – `QDAWC`

The calling sequences for these routines are very similar. The function to be integrated is always `F`; the lower and upper limits are, respectively, `A` and `B`. The requested absolute error ε is `ERRABS`, while the requested relative error ρ is `ERRREL`. These quadrature routines return two numbers of interest, namely, `RESULT` and `ERREST`, which are the approximate integral *R* and the error estimate *E*, respectively. These numbers are related as follows:

$$\left|\int_a^b f(x)w(x)\,dx - R\right| \le E \le \max\left\{\varepsilon, \rho\left|\int_a^b f(x)w(x)\,dx\right|\right\}$$

One situation that occasionally arises in univariate quadrature concerns the approximation of integrals when only tabular data are given. The routines described above do not directly address this question. However, the standard method for handling this problem is first to interpolate the data and then to integrate the interpolant. This can be accomplished by using the IMSL spline interpolation routines described in Chapter 3 with one of the Chapter 3 integration routines `CSINT` (page 423), `BSINT` (page 450), or `PPITG` (page 512).

Multivariate Quadrature

Two routines are described in this chapter that are of use in approximating certain multivariate integrals. In particular, the routine `TWODQ` returns an approximation to an iterated two-dimensional integral of the form

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y)\, dy\, dx$$

The second routine, `QAND`, returns an approximation to the integral of a function of *n* variables over a hyper-rectangle

$$\int_{a_1}^{b_1} \cdots \int_{a_n}^{b_n} f\left(x_1, \ldots, x_n\right) dx_n \ldots dx_1$$

If one has two- or three-dimensional tensor-product tabular data, use the IMSL spline interpolation routines `BS2IN` (page 459) or `BS3IN` (page 464), followed by the IMSL spline integration routines `BS2IG` (page 487) or `BS3IG` (page 500) that are described in Chapter 3.

Gauss rules and three-term recurrences

The routines described in this section deal with the constellation of problems encountered in Gauss quadrature. These problems arise when quadrature formulas, which integrate polynomials of the highest degree possible, are computed. Once a member of a family of seven weight functions is specified, the routine `GQRUL` (page 621) produces the points $\{x_i\}$ and weights $\{w_i\}$ for $i = 1, \ldots,$ N that satisfy

$$\int_a^b f(x) w(x)\, dx = \sum_{i=1}^{N} f\left(x_i\right) w_i$$

for all functions *f* that are polynomials of degree less than 2*N*. The weight functions *w* may be selected from the following table:

$w(x)$	**Interval**	**Name**
1	$(-1,1)$	Legendre
$1/\sqrt{1-x^2}$	$(-1,1)$	Chebyshev 1st kind
$\sqrt{1-x^2}$	$(-1,1)$	Chebyshev 2nd kind
e^{-x^2}	$(-\infty,\infty)$	Hermite
$(1+x)^{\alpha}(1-x)^{\beta}$	$(-1,1)$	Jacobi
$e^{-x}x^{\alpha}$	$(0,\infty)$	Generalized Laguerre
$1/\cosh(x)$	$(-\infty\ \infty)$	Hyperbolic cosine

Where permissible, `GQRUL` will also compute Gauss-Radau and Gauss-Lobatto quadrature rules. The routine `RECCF` (page 628) produces the three-term recurrence relation for the monic orthogonal polynomials with respect to the above weight functions.

Another routine, `GQRCF` (page 625), produces the Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule from the three-term recurrence relation. This means Gauss rules for general weight functions may be obtained if the three-term recursion for the orthogonal polynomials is known. The routine `RECQR` (page 630) is an inverse to `GQRCF` in the sense that it produces the recurrence coefficients given the Gauss quadrature formula.

The last routine described in this section, `FQRUL` (page 632), generates the Fejér quadrature rules for the following family of weights:

$$\begin{aligned}
w(x) &= 1 \\
w(x) &= 1/(x-\alpha) \\
w(x) &= (b-x)^{\alpha}(x-a)^{\beta} \\
w(x) &= (b-x)^{\alpha}(x-a)^{\beta}\ \ln(x-a) \\
w(x) &= (b-x)^{\alpha}(x-a)^{\beta}\ \ln(b-x)
\end{aligned}$$

Numerical differentiation

We provide one routine, `DERIV` (page 636), for numerical differentiation. This routine provides an estimate for the first, second, or third derivative of a user-supplied function.

QDAGS/DQDAGS (Single/Double precision)

Integrate a function (which may have endpoint singularities).

Usage

```
CALL QDAGS (F, A, B, ERRABS, ERRREL, RESULT, ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F`(`X`), where
`X` – Independent variable. (Input)
`F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `QDAGS` 2500 units, or
   `DQDAGS` 4500 units.

   Workspace may be explicitly provided, if desired, by use of `Q2AGS`/`DQ2AGS`. The reference is

   ```
   CALL Q2AGS (F, A, B, ERRABS, ERRREL, RESULT, ERREST
               MAXSUB, NEVAL, NSUBIN, ALIST, BLIST,
               RLIST, ELIST, IORD)
   ```

   The additional arguments are as follows:

   MAXSUB — Number of subintervals allowed. (Input)
   A value of 500 is used by `QDAGS`.

   NEVAL — Number of evaluations of `F`. (Output)

   NSUBIN — Number of subintervals generated. (Output)

   ALIST — Array of length `MAXSUB` containing a list of the `NSUBIN` left endpoints. (Output)

   BLIST — Array of length `MAXSUB` containing a list of the `NSUBIN` right endpoints. (Output)

RLIST — Array of length MAXSUB containing approximations to the NSUBIN integrals over the intervals defined by ALIST, BLIST. (Output)

ELIST — Array of length MAXSUB containing the error estimates of the NSUBIN values in RLIST. (Output)

IORD — Array of length MAXSUB. (Output)
Let *k* be
NSUBIN if NSUBIN ≤ (MAXSUB/2 + 2);
MAXSUB + 1 − NSUBIN otherwise.
The first *k* locations contain pointers to the error estimates over the subintervals such that ELIST(IORD(1)), …, ELIST(IORD(*k*)) form a decreasing sequence.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.
3	4	Roundoff error in the extrapolation table, preventing the requested tolerance from being achieved, has been detected.
4	5	Integral is probably divergent or slowly convergent.

3. If EXACT is the exact value, QDAGS attempts to find RESULT such that |EXACT − RESULT| ≤ max(ERRABS, ERRREL * |EXACT|). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine QDAGS is a general-purpose integrator that uses a globally adaptive scheme to reduce the absolute error. It subdivides the interval [*A*, *B*] and uses a 21-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the 10-point Gauss quadrature rule. This routine is designed to handle functions with endpoint singularities. However, the performance on functions, which are well-behaved at the endpoints, is quite good also. In addition to the general strategy described in QDAG (page 591), this routine uses an extrapolation procedure known as the ε-algorithm. The routine QDAGS is an implementation of the routine QAGS, which is fully documented by Piessens et al. (1983). Should QDAGS fail to produce acceptable results, then either IMSL routines QDAG or QDAG* may be appropriate. These routines are documented in this chapter.

Example

The value of

$$\int_0^1 \ln(x)x^{-1/2}dx = -4$$

is estimated. The values of the actual and estimated error are machine dependent.

```
      INTEGER    NOUT
      REAL       A, ABS, B, ERRABS, ERREST, ERROR, ERRREL, EXACT, F,
     &           RESULT
      INTRINSIC  ABS
      EXTERNAL   F, QDAGS, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      A = 0.0
      B = 1.0
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDAGS (F, A, B, ERRABS, ERRREL, RESULT, ERREST)
C                                 Print results
      EXACT = -4.0
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       ALOG, SQRT
      INTRINSIC  ALOG, SQRT
      F = ALOG(X)/SQRT(X)
      RETURN
      END
```

Output

```
Computed =  -4.000              Exact =  -4.000

Error estimate = 2.782E-04      Error = 4.292E-06
```

QDAG/DQDAG (Single/Double precision)

Integrate a function using a globally adaptive scheme based on Gauss-Kronrod rules.

Usage

```
CALL QDAG (F, A, B, ERRABS, ERRREL, IRULE, RESULT, ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F`(`X`), where

`X` – Independent variable. (Input)

`F` – The function value. (Output)

`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

IRULE — Choice of quadrature rule. (Input)
The Gauss-Kronrod rule is used with the following points:

IRULE	**Points**
1	7-15
2	10-21
3	15-31
4	20-41
5	25-51
6	30-61

`IRULE` = 2 is recommended for most functions. If the function has a peak singularity, use `IRULE` = 1. If the function is oscillatory, use `IRULE` = 6.

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

`QDAG`	2500 units, or
`DQDAG`	4500 units.

Workspace may be explicitly provided, if desired, by use of `Q2AG`/`DQ2AG`. The reference is

```
CALL Q2AG (F, A, B, ERRABS, ERRREL, IRULE, RESULT,
           ERREST, MAXSUB, NEVAL, NSUBIN, ALIST,
           BLIST, RLIST, ELIST, IORD)
```

The additional arguments are as follows:

MAXSUB — Number of subintervals allowed. (Input)
A value of 500 is used by `QDAG`.

NEVAL — Number of evaluations of `F`. (Output)

NSUBIN — Number of subintervals generated. (Output)

ALIST — Array of length MAXSUB containing a list of the NSUBIN left endpoints. (Output)

BLIST — Array of length MAXSUB containing a list of the NSUBIN right endpoints. (Output)

RLIST — Array of length MAXSUB containing approximations to the NSUBIN integrals over the intervals defined by ALIST, BLIST. (Output)

ELIST — Array of length MAXSUB containing the error estimates of the NSUBIN values in RLIST. (Output)

IORD — Array of length MAXSUB. (Output)
Let K be NSUBIN if NSUBIN.LE.(MAXSUB/2 + 2), MAXSUB + 1 – NSUBIN otherwise. The first K locations contain pointers to the error estimates over the corresponding subintervals, such that ELIST(IORD(1)), …, ELIST(IORD(K)) form a decreasing sequence.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.

3. If EXACT is the exact value, QDAG attempts to find RESULT such that ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT)). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine QDAG is a general-purpose integrator that uses a globally adaptive scheme in order to reduce the absolute error. It subdivides the interval [A, B] and uses a ($2k$ + 1)-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the k-point Gauss quadrature rule. The subinterval with the largest estimated error is then bisected and the same procedure is applied to both halves. The bisection process is continued until either the error criterion is satisfied, roundoff error is detected, the subintervals become too small, or the maximum number of subintervals allowed is reached. The routine QDAG is based on the subroutine QAG by Piessens et al. (1983).

Should QDAG fail to produce acceptable results, then one of the IMSL routines QDAG* may be appropriate. These routines are documented in this chapter.

Example

The value of

$$\int_0^2 xe^x \, dx = e^2 + 1$$

is estimated. Since the integrand is not oscillatory, `IRULE` = 1 is used. The values of the actual and estimated error are machine dependent.

```
      INTEGER    IRULE, NOUT
      REAL       A, ABS, B, ERRABS, ERREST, ERROR, ERRREL, EXACT, EXP,
     &           F, RESULT
      INTRINSIC  ABS, EXP
      EXTERNAL   F, QDAG, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Set limits of integration
      A = 0.0
      B = 2.0
C                                  Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
C                                  Parameter for non-oscillatory
C                                  function
      IRULE = 1
      CALL QDAG (F, A, B, ERRABS, ERRREL, IRULE, RESULT, ERREST)
C                                  Print results
      EXACT = 1.0 + EXP(2.0)
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       EXP
      INTRINSIC  EXP
      F = X*EXP(X)
      RETURN
      END
```

Output

```
Computed =   8.389              Exact =   8.389

Error estimate = 5.000E-05      Error = 9.537E-07
```

QDAGP/DQDAGP (Single/Double precision)

Integrate a function with singularity points given.

Usage

```
CALL QDAGP (F, A, B, NPTS, POINTS, ERRABS, ERRREL, RESULT,
            ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F(X)`, where
`X` – Independent variable. (Input)
`F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

NPTS — Number of break points given. (Input)

POINTS — Array of length `NPTS` containing breakpoints in the range of integration. (Input)
Usually these are points where the integrand has singularities.

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `QDAGP` 2704 + 2 `*` `NPTS` units, or
   `DQDAGP` 4506 + 3 `*` `NPTS` units.

   Workspace may be explicitly provided, if desired, by use of `Q2AGP`/`DQ2AGP`. The reference is

   ```
   CALL Q2AGP (F, A, B, NPTS, POINTS, ERRABS, ERRREL,
               RESULT, ERREST, MAXSUB, NEVAL, NSUBIN,
               ALIST, BLIST, RLIST, ELIST, IORD, LEVEL,
               WK, IWK)
   ```

   The additional arguments are as follows:

   MAXSUB — Number of subintervals allowed. (Input)
   A value of 450 is used by `QDAGP`.

   NEVAL — Number of evaluations of `F`. (Output)

   NSUBIN — Number of subintervals generated. (Output)

   ALIST — Array of length `MAXSUB` containing a list of the `NSUBIN` left endpoints. (Output)

   BLIST — Array of length `MAXSUB` containing a list of the `NSUBIN` right endpoints. (Output)

   RLIST — Array of length `MAXSUB` containing approximations to the `NSUBIN` integrals over the intervals defined by `ALIST`, `BLIST`. (Output)

ELIST — Array of length MAXSUB containing the error estimates of the NSUBIN values in RLIST. (Output)

IORD — Array of length MAXSUB. (Output)
Let K be NSUBIN if NSUBIN.LE.(MAXSUB/2 + 2), MAXSUB + 1 – NSUBIN otherwise. The first K locations contain pointers to the error estimates over the subintervals, such that ELIST(IORD(1)), …, ELIST(IORD(K)) form a decreasing sequence.

LEVEL — Array of length MAXSUB, containing the subdivision levels of the subinterval. (Output)
That is, if (AA, BB) is a subinterval of (P1, P2) where P1 as well as P2 is a user-provided break point or integration limit, then (AA, BB) has level L if ABS(BB – AA) = ABS(P2 – P1) * 2**(–L).

WK — Work array of length NPTS + 2.

IWK — Work array of length NPTS + 2.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.
3	4	Roundoff error in the extrapolation table, preventing the requested tolerance from being achieved, has been detected.
4	5	Integral is probably divergent or slowly convergent.

3. If EXACT is the exact value, QDAGP attempts to find RESULT such that ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT)). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine QDAGP uses a globally adaptive scheme in order to reduce the absolute error. It initially subdivides the interval [A, B] into NPTS + 1 user-supplied subintervals and uses a 21-point Gauss-Kronrod rule to estimate the integral over each subinterval. The error for each subinterval is estimated by comparison with the 10-point Gauss quadrature rule. This routine is designed to handle endpoint as well as interior singularities. In addition to the general strategy described in the IMSL routine QDAG (page 591), this routine employs an extrapolation procedure known as the ε-algorithm. The routine QDAGP is an implementation of the subroutine QAGP, which is fully documented by Piessens et al. (1983).

Example

The value of

$$\int_0^3 x^3 \ln\left|\left(x^2-1\right)\left(x^2-2\right)\right| dx = 61\ln 2 + \frac{77}{4}\ln 7 - 27$$

is estimated. The values of the actual and estimated error are machine dependent. Note that this subroutine never evaluates the user-supplied function at the user-supplied breakpoints.

```
      INTEGER    NOUT, NPTS
      REAL       A, ABS, ALOG, B, ERRABS, ERREST, ERROR, ERRREL,
     &           EXACT, F, POINTS(2), RESULT, SQRT
      INTRINSIC  ABS, ALOG, SQRT
      EXTERNAL   F, QDAGP, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      A = 0.0
      B = 3.0
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.01
C                                 Set singularity parameters
      NPTS      = 2
      POINTS(1) = 1.0
      POINTS(2) = SQRT(2.0)
      CALL QDAGP (F, A, B, NPTS, POINTS, ERRABS, ERRREL, RESULT,
     &            ERREST)
C                                 Print results
      EXACT = 61.0*ALOG(2.0) + 77.0/4.0*ALOG(7.0) - 27.0
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
C
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       ABS, ALOG
      INTRINSIC  ABS, ALOG
      F = X**3*ALOG(ABS((X*X-1.0)*(X*X-2.0)))
      RETURN
      END
```

Output

```
Computed =  52.741             Exact =  52.741

Error estimate = 5.062E-01      Error = 6.218E-04
```

QDAGI/DQDAGI (Single/Double precision)

Integrate a function over an infinite or semi-infinite interval.

Usage

```
CALL QDAGI (F, BOUND, INTERV, ERRABS, ERRREL, RESULT,
            ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is
`F(X)`, where

`X` – Independent variable. (Input)

`F` – The function value. (Output)

`F` must be declared `EXTERNAL` in the calling program.

BOUND — Finite bound of the integration range. (Input)
Ignored if `INTERV` = 2.

INTERV — Flag indicating integration interval. (Input)

INTERV	**Interval**
–1	(–∞, `BOUND`)
1	(`BOUND`, + ∞)
2	(–∞, + ∞)

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `QDAGI` 2500 units, or
   `DQDAGI` 4500 units.

   Workspace may be explicitly provided, if desired, by use of `Q2AGI/DQ2AGI`. The reference is

```
CALL Q2AGI (F, BOUND, INTERV, ERRABS, ERRREL,
            RESULT, ERREST, MAXSUB, NEVAL, NSUBIN,
            ALIST, BLIST, RLIST, ELIST, IORD)
```

   The additional arguments are as follows:

   MAXSUB — Number of subintervals allowed. (Input)
   A value of 500 is used by `QDAGI`.

   NEVAL — Number of evaluations of `F`. (Output)

NSUBIN — Number of subintervals generated. (Output)

ALIST — Array of length `MAXSUB` containing a list of the `NSUBIN` left endpoints. (Output)

BLIST — Array of length `MAXSUB` containing a list of the `NSUBIN` right endpoints. (Output)

RLIST — Array of length `MAXSUB` containing approximations to the `NSUBIN` integrals over the intervals defined by `ALIST`, `BLIST`. (Output)

ELIST — Array of length `MAXSUB` containing the error estimates of the `NSUBIN` values in `RLIST`. (Output)

IORD — Array of length `MAXSUB`. (Output)
Let `K` be `NSUBIN` if `NSUBIN .LE.`(`MAXSUB`/2 + 2), `MAXSUB` + 1 – `NSUBIN` otherwise. The first `K` locations contain pointers to the error estimates over the subintervals, such that `ELIST`(`IORD`(1)), …, `ELIST`(`IORD`(`K`)) form a decreasing sequence.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.
3	4	Roundoff error in the extrapolation table, preventing the requested tolerance from being achieved, has been detected.
4	5	Integral is divergent or slowly convergent.

3. If `EXACT` is the exact value, `QDAGI` attempts to find `RESULT` such that `ABS`(`EXACT` – `RESULT`)`.LE.MAX`(`ERRABS`, `ERRREL * ABS`(`EXACT`)). To specify only a relative error, set `ERRABS` to zero. Similarly, to specify only an absolute error, set `ERRREL` to zero.

Algorithm

The routine `QDAGI` uses a globally adaptive scheme in an attempt to reduce the absolute error. It initially transforms an infinite or semi-infinite interval into the finite interval [0, 1]. Then, `QDAGI` uses a 21-point Gauss-Kronrod rule to estimate the integral and the error. It bisects any interval with an unacceptable error estimate and continues this process until termination. This routine is designed to handle endpoint singularities. In addition to the general strategy described in `QDAG` (page 591), this subroutine employs an extrapolation procedure known as the ε-algorithm. The routine `QDAGI` is an implementation of the subroutine `QAGI`, which is fully documented by Piessens et al. (1983).

Example

The value of

$$\int_0^\infty \frac{\ln(x)}{1+(10x)^2}\,dx = \frac{-\pi\ln(10)}{20}$$

is estimated. The values of the actual and estimated error are machine dependent. Note that we have requested an absolute error of 0 and a relative error of .001. The effect of these requests, as documented in Comment 3 above, is to ignore the absolute error requirement.

```
      INTEGER    INTERV, NOUT
      REAL       ABS, ALOG, ATAN, BOUND, ERRABS, ERREST, ERROR,
     &           ERRREL, EXACT, F, PI, RESULT, CONST
      INTRINSIC  ABS, ALOG
      EXTERNAL   F, QDAGI, UMACH, CONST
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      BOUND  = 0.0
      INTERV = 1
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDAGI (F, BOUND, INTERV, ERRABS, ERRREL, RESULT, ERREST)
C                                 Print results
      PI    = CONST(’PI’)
      EXACT = -PI*ALOG(10.)/20.
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3//’ Error ’,
     &       ’estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       ALOG
      INTRINSIC  ALOG
      F = ALOG(X)/(1.+(10.*X)**2)
      RETURN
      END
```

Output

```
Computed =  -0.362             Exact =  -0.362

Error estimate = 2.652E-06      Error = 5.960E-08
```

QDAWO/DQDAWO (Single/Double precision)

Integrate a function containing a sine or a cosine.

Usage

```
CALL QDAWO (F, A, B, IWEIGH, OMEGA, ERRABS, ERRREL, RESULT,
            ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F(X)`, where

`X` – Independent variable. (Input)

`F` – The function value. (Output)

`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

IWEIGH — Type of weight function used. (Input)

`IWEIGH`	**Weight**
1	`COS(OMEGA * X)`
2	`SIN(OMEGA * X)`

OMEGA — Parameter in the weight function. (Input)

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to `B` of `F * WEIGHT`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `QDAWO` 2865 units, or
   `DQDAWO` 4950 units.

   Workspace may be explicitly provided, if desired, by use of `Q2AWO/DQ2AWO`. The reference is

   ```
   CALL Q2AWO (F, A, B, IWEIGH, OMEGA, ERRABS, ERRREL,
               RESULT, ERREST, MAXSUB, MAXCBY, NEVAL,
               NSUBIN, ALIST, BLIST, RLIST, ELIST,
               IORD, NNLOG, WK)
   ```

   The additional arguments are as follows:

   MAXSUB — Maximum number of subintervals allowed. (Input)
   A value of 390 is used by `QDAWO`.

MAXCBY — Upper bound on the number of Chebyshev moments which can be stored. That is, for the intervals of lengths `ABS(B – A) * 2**(–L)`, `L` = 0, 1, …, `MAXCBY` – 2, `MAXCBY.GE.1`. The routine `QDAWO` uses 21. (Input)

NEVAL — Number of evaluations of `F`. (Output)

NSUBIN — Number of subintervals generated. (Output)

ALIST — Array of length `MAXSUB` containing a list of the `NSUBIN` left endpoints. (Output)

BLIST — Array of length `MAXSUB` containing a list of the `NSUBIN` right endpoints. (Output)

RLIST — Array of length `MAXSUB` containing approximations to the `NSUBIN` integrals over the intervals defined by `ALIST`, `BLIST`. (Output)

ELIST — Array of length `MAXSUB` containing the error estimates of the `NSUBIN` values in `RLIST`. (Output)

IORD — Array of length `MAXSUB`. Let `K` be `NSUBIN` if `NSUBIN.LE.` (`MAXSUB`/2 + 2), `MAXSUB` + 1 – `NSUBIN` otherwise. The first `K` locations contain pointers to the error estimates over the subintervals, such that `ELIST(IORD`(1)), …, `ELIST(IORD(K))` form a decreasing sequence. (Output)

NNLOG — Array of length `MAXSUB` containing the subdivision levels of the subintervals, i.e. `NNLOG(I)` = `L` means that the subinterval numbered `I` is of length `ABS(B – A) *` (1– `L`). (Output)

WK — Array of length 25 `* MAXCBY`. (Workspace)

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.
3	4	Roundoff error in the extrapolation table, preventing the requested tolerances from being achieved, has been detected.
4	5	Integral is probably divergent or slowly convergent.

3. If `EXACT` is the exact value, `QDAWO` attempts to find `RESULT` such that `ABS(EXACT – RESULT) .LE. MAX(ERRABS, ERRREL * ABS(EXACT))`. To specify only a relative error, set `ERRABS` to zero. Similarly, to specify only an absolute error, set `ERRREL` to zero.

Algorithm

The routine QDAWO uses a globally adaptive scheme in an attempt to reduce the absolute error. This routine computes integrals whose integrands have the special form $w(x)\, f(x)$, where $w(x)$ is either cos ωx or sin ωx. Depending on the length of the subinterval in relation to the size of ω, either a modified Clenshaw-Curtis procedure or a Gauss-Kronrod 7/15 rule is employed to approximate the integral on a subinterval. In addition to the general strategy described for the IMSL routine QDAG (page 591), this subroutine uses an extrapolation procedure known as the ε-algorithm. The routine QDAWO is an implementation of the subroutine QAWO, which is fully documented by Piessens et al. (1983).

Example

The value of

$$\int_0^1 \ln(x)\sin(10\pi x)\, dx$$

is estimated. The values of the actual and estimated error are machine dependent. Notice that the log function is coded to protect for the singularity at zero.

```
      INTEGER    IWEIGH, NOUT
      REAL       A, ABS, B, CONST, ERRABS, ERREST, ERROR, ERRREL,
     &           EXACT, F, OMEGA, PI, RESULT
      INTRINSIC  ABS
      EXTERNAL   CONST, F, QDAWO, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      A = 0.0
      B = 1.0
C                                 Weight function = sin(10.*pi*x)
      IWEIGH = 2
      PI     = CONST('PI')
      OMEGA  = 10.*PI
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDAWO (F, A, B, IWEIGH, OMEGA, ERRABS, ERRREL, RESULT,
     &            ERREST)
C                                 Print results
      EXACT = -0.1281316
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (' Computed =', F8.3, 13X, ' Exact =', F8.3, /, /,
     &       ' Error estimate =', 1PE10.3, 6X, 'Error =', 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       ALOG
      INTRINSIC  ALOG
      IF (X .EQ. 0.) THEN
         F = 0.0
      ELSE
         F = ALOG(X)
      END IF
      RETURN
      END
```

Output

```
Computed =  -0.128            Exact =  -0.128

Error estimate = 7.504E-05      Error = 5.260E-06
```

QDAWF/DQDAWF (Single/Double precision)

Compute a Fourier integral.

Usage

`CALL QDAWF (F, A, IWEIGH, OMEGA, ERRABS, RESULT, ERREST)`

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F`(`X`), where
`X` – Independent variable. (Input)
`F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

IWEIGH — Type of weight function used. (Input)

`IWEIGH`	**Weight**
1	`COS(OMEGA * X)`
2	`SIN(OMEGA * X)`

OMEGA — Parameter in the weight function. (Input)

ERRABS — Absolute accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to infinity of `F * WEIGHT`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `QDAWF` 2865 units, or
   `DQDAWF` 4950 units.

   Workspace may be explicitly provided, if desired, by use of `Q2AWF`/`DQ2AWF`. The reference is

```
CALL Q2AWF (F, A, IWEIGH, OMEGA, ERRABS, RESULT,
            ERREST, MAXCYL, MAXSUB, MAXCBY, NEVAL,
            NCYCLE, RSLIST, ERLIST, IERLST, NSUBIN,
            WK, IWK)
```

The additional arguments are as follows:

MAXSUB — Maximum number of subintervals allowed. (Input)
A value of 365 is used by `QDAWF`.

MAXCYL — Maximum number of cycles allowed. (Input)
`MAXCYL` must be at least 3. `QDAWF` uses 50.

MAXCBY — Maximum number of Chebyshev moments allowed. (Input)
`QDAWF` uses 21.

NEVAL — Number of evaluations of `F`. (Output)

NCYCLE — Number of cycles used. (Output)

RSLIST — Array of length `MAXCYL` containing the contributions to the integral over the interval (`A` + (*k* – 1) `* C`, `A` + *k* `* C`), for *k* = 1, …, `NCYCLE`. (Output)
`C` = (2 `* INT`(`ABS`(`OMEGA`)) + 1) `* PI`/`ABS`(`OMEGA`).

ERLIST — Array of length `MAXCYL` containing the error estimates for the intervals defined in `RSLIST`. (Output)

IERLST — Array of length `MAXCYL` containing error flags for the intervals defined in `RSLIST`. (Output)

IERLST(K)	**Meaning**
1	The maximum number of subdivisions (`MAXSUB`) has been achieved on the `K`-th cycle.
2	Roundoff error prevents the desired accuracy from being achieved on the `K`-th cycle.
3	Extremely bad integrand behavior occurs at some points of the `K`-th cycle.
4	Integration procedure does not converge (to the desired accuracy) due to roundoff in the extrapolation procedure on the `K`-th cycle. It is assumed that the result on this interval is the best that can be obtained.
5	Integral over the `K`-th cycle is divergent or slowly convergent.

NSUBIN — Number of subintervals generated. (Output)

WK — Work array of length 4 `* MAXSUB` + 25 `* MAXCBY`.

IWK — Work array of length 2 `* MAXSUB`.

2. Informational errors

Type	Code	
3	1	Bad integrand behavior occurred in one or more cycles.
4	2	Maximum number of cycles allowed has been reached.
3	3	Extrapolation table constructed for convergence acceleration of the series formed by the integral contributions of the cycles does not converge to the requested accuracy.

3. If `EXACT` is the exact value, `QDAWF` attempts to find `RESULT` such that `ABS(EXACT – RESULT) .LE. ERRABS`.

Algorithm

The routine `QDAWF` uses a globally adaptive scheme in an attempt to reduce the absolute error. This routine computes integrals whose integrands have the special form $w(x)\,f(x)$, where $w(x)$ is either cos ωx or sin ωx. The integration interval is always semi-infinite of the form $[A, \infty]$. These Fourier integrals are approximated by repeated calls to the IMSL routine `QDAWO` (page 601) followed by extrapolation. The routine `QDAWF` is an implementation of the subroutine `QAWF`, which is fully documented by Piessens et al. (1983).

Example

The value of

$$\int_0^\infty x^{-1/2}\cos(\pi x/2)\,dx = 1$$

is estimated. The values of the actual and estimated error are machine dependent. Notice that *F* is coded to protect for the singularity at zero.

```
      INTEGER    IWEIGH, NOUT
      REAL       A, ABS, CONST, ERRABS, ERREST, ERROR, EXACT, F,
     &           OMEGA, PI, RESULT
      INTRINSIC  ABS
      EXTERNAL   CONST, F, QDAWF, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set lower limit of integration
      A = 0.0
C                                 Select weight W(X) = COS(PI*X/2)
      IWEIGH = 1
      PI     = CONST('PI')
      OMEGA  = PI/2.0
C                                 Set error tolerance
      ERRABS = 0.001
      CALL QDAWF (F, A, IWEIGH, OMEGA, ERRABS, RESULT, ERREST)
C                                 Print results
      EXACT = 1.0
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
```

```
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       SQRT
      INTRINSIC  SQRT
      IF (X .GT. 0.0) THEN
         F = 1.0/SQRT(X)
      ELSE
         F = 0.0
      END IF
      RETURN
      END
```

Output

```
Computed =   1.000              Exact =   1.000

Error estimate = 6.267E-04      Error = 2.205E-06
```

QDAWS/DQDAWS (Single/Double precision)

Integrate a function with algebraic-logarithmic singularities.

Usage

```
CALL QDAWS (F, A, B, IWEIGH, ALPHA, BETA, ERRABS, ERRREL,
            RESULT, ERREST)
```

Arguments

F — User-supplied FUNCTION to be integrated. The form is F(X), where
X – Independent variable. (Input)
F – The function value. (Output)
F must be declared EXTERNAL in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)
B must be greater than A

IWEIGH — Type of weight function used. (Input)

IWEIGH	**Weight**
1	(X – A)**ALPHA * (B – X)**BETA
2	(X – A)**ALPHA * (B – X)**BETA * LOG(X – A)
3	(X – A)**ALPHA * (B – X)**BETA * LOG(B – X)
4	(X – A)**ALPHA * (B – X)**BETA * LOG (X – A) * LOG (B – X)

ALPHA — Parameter in the weight function. (Input)
ALPHA must be greater than –1.0.

BETA — Parameter in the weight function. (Input)
BETA must be greater than −1.0.

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from A to B of F * WEIGHT. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   QDAWS 2500 units, or
   DQDAWS 4500 units.

   Workspace may be explicitly provided, if desired, by use of Q2AWS/DQ2AWS. The reference is

```
CALL Q2AWS (F, A, B, IWEIGH, ALPHA, BETA, ERRABS,
            ERRREL, RESULT, ERREST, MAXSUB, NEVAL,
            NSUBIN, ALIST, BLIST, RLIST, ELIST,
            IORD)
```

   The additional arguments are as follows:

   MAXSUB — Maximum number of subintervals allowed. (Input)
   A value of 500 is used by QDAWS.

   NEVAL — Number of evaluations of *F*. (Output)

   NSUBIN — Number of subintervals generated. (Output)

   ALIST — Array of length MAXSUB containing a list of the NSUBIN left endpoints. (Output)

   BLIST — Array of length MAXSUB containing a list of the NSUBIN right endpoints. (Output)

   RLIST — Array of length MAXSUB containing approximations to the NSUBIN integrals over the intervals defined by ALIST, BLIST. (Output)

   ELIST — Array of length MAXSUB containing the error estimates of the NSUBIN values in RLIST. (Output)

   IORD — Array of length MAXSUB. Let K be NSUBIN if NSUBIN.LE.(MAXSUB/2 + 2), MAXSUB + 1 − NSUBIN otherwise. The first K locations contain pointers to the error estimates over the subintervals, such that ELIST(IORD(1)), …, ELIST(IORD(K)) form a decreasing sequence. (Output)

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.

3. If `EXACT` is the exact value, `QDAWS` attempts to find `RESULT` such that `ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT))`. To specify only a relative error, set `ERRABS` to zero. Similarly, to specify only an absolute error, set `ERRREL` to zero.

Algorithm

The routine `QDAWS` uses a globally adaptive scheme in an attempt to reduce the absolute error. This routine computes integrals whose integrands have the special form $w(x)\,f(x)$, where $w(x)$ is a weight function described above. A combination of modified Clenshaw-Curtis and Gauss-Kronrod formulas is employed. In addition to the general strategy described for the IMSL routine `QDAG` (page 591), this routine uses an extrapolation procedure known as the ε-algorithm. The routine `QDAWS` is an implementation of the routine `QAWS`, which is fully documented by Piessens et al. (1983).

Example

The value of

$$\int_0^1 \left[(1+x)(1-x)\right]^{1/2} x\ln(x)\,dx = \frac{3\ln(2)-4}{9}$$

is estimated. The values of the actual and estimated error are machine dependent.

```
      INTEGER    IWEIGH, NOUT
      REAL       A, ABS, ALOG, ALPHA, B, BETA, ERRABS, ERREST, ERROR,
     &           ERRREL, EXACT, F, RESULT
      INTRINSIC  ABS, ALOG
      EXTERNAL   F, QDAWS, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Set limits of integration
      A = 0.0
      B = 1.0
C                                  Select weight
      ALPHA  = 1.0
      BETA   = 0.5
      IWEIGH = 2
C                                  Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDAWS (F, A, B, IWEIGH, ALPHA, BETA, ERRABS, ERRREL,
     &            RESULT, ERREST)
C                                  Print results
```

```
      EXACT = (3.*ALOG(2.)-4.)/9.
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (' Computed =', F8.3, 13X, ' Exact =', F8.3, /, /,
     &       ' Error estimate =', 1PE10.3, 6X, 'Error =', 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       SQRT
      INTRINSIC  SQRT
      F = SQRT(1.0+X)
      RETURN
      END
```

Output

```
Computed =  -0.213             Exact =  -0.213

Error estimate = 1.261E-08      Error = 2.980E-08
```

QDAWC/DQDAWC (Single/Double precision)

Integrate a function F(X)/(X – C) in the Cauchy principal value sense.

Usage

CALL QDAWC (F, A, B, C, ERRABS, ERRREL, RESULT, ERREST)

Arguments

F — User-supplied FUNCTION to be integrated. The form is F(X), where
X – Independent variable. (Input)
F – The function value. (Output)
F must be declared EXTERNAL in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

C — Singular point. (Input)
C must not equal A or B.

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from A to B of F(X)/(X – C). (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   QDAWC 2500 units, or
   DQDAWC 4500 units.

   Workspace may be explicitly provided, if desired, by use of Q2AWC/DQ2AWC. The reference is

   ```
   CALL Q2AWC (F, A, B, C, ERRABS, ERRREL, RESULT,
               ERREST, MAXSUB, NEVAL, NSUBIN, ALIST,
               BLIST, RLIST, ELIST, IORD)
   ```

   The additional arguments are as follows:

   MAXSUB — Number of subintervals allowed. (Input)
   A value of 500 is used by QDAWC.

   NEVAL — Number of evaluations of F. (Output)

   NSUBIN — Number of subintervals generated. (Output)

   ALIST — Array of length MAXSUB containing a list of the NSUBIN left endpoints. (Output)

   BLIST — Array of length MAXSUB containing a list of the NSUBIN right endpoints. (Output)

   RLIST — Array of length MAXSUB containing approximations to the NSUBIN integrals over the intervals defined by ALIST, BLIST. (Output)

   ELIST — Array of length MAXSUB containing the error estimates of the NSUBIN values in RLIST. (Output)

   IORD — Array of length MAXSUB. (Output)
   Let K be NSUBIN if NSUBIN.LE.(MAXSUB/2 + 2), MAXSUB + 1 – NSUBIN otherwise. The first K locations contain pointers to the error estimates over the subintervals, such that ELIST(IORD(1)), …, ELIST(IORD(K)) form a decreasing sequence.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.
3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.

3. If EXACT is the exact value, QDAWC attempts to find RESULT such that ABS(EXACT – RESULT) .LE. MAX(ERRABS, ERRREL * ABS(EXACT)). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine QDAWC uses a globally adaptive scheme in an attempt to reduce the absolute error. This routine computes integrals whose integrands have the special form $w(x)\,f(x)$, where $w(x) = 1/(x - c)$. If c lies in the interval of integration, then the integral is interpreted as a Cauchy principal value. A combination of modified Clenshaw-Curtis and Gauss-Kronrod formulas are employed. In addition to the general strategy described for the IMSL routine QDAG (page 591), this routine uses an extrapolation procedure known as the ε-algorithm. The routine QDAWC is an implementation of the subroutine QAWC, which is fully documented by Piessens et al. (1983).

Arguments

The Cauchy principal value of

$$\int_{-1}^{5} \frac{1}{x\left(5x^3+6\right)}\,dx = \frac{\ln(125/631)}{18}$$

is estimated. The values of the actual and estimated error are machine dependent.

```
      INTEGER    NOUT
      REAL       A, ABS, ALOG, B, C, ERRABS, ERREST, ERROR, ERRREL,
     &           EXACT, F, RESULT
      INTRINSIC  ABS, ALOG
      EXTERNAL   F, QDAWC, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration and C
      A = -1.0
      B = 5.0
      C = 0.0
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDAWC (F, A, B, C, ERRABS, ERRREL, RESULT, ERREST)
C                                 Print results
      EXACT = ALOG(125./631.)/18.
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      F = 1.0/(5.*X**3+6.0)
      RETURN
      END
```

Output

```
Computed =  -0.090              Exact =  -0.090

Error estimate = 2.235E-06      Error = 2.980E-08
```

QDNG/DQDNG (Single/Double precision)

Integrate a smooth function using a nonadaptive rule.

Usage

```
CALL QDNG (F, A, B, ERRABS, ERRREL, RESULT, ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F(X)`, where
`X` – Independent variable. (Input)
`F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of integration. (Input)

B — Upper limit of integration. (Input)

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Informational error

Type	Code	
4	1	The maximum number of steps allowed have been taken. The integral is too difficult for `QDNG`.

2. If `EXACT` is the exact value, `QDNG` attempts to find `RESULT` such that `ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT))`. To specify only a relative error, set `ERRABS` to zero. Similarly, to specify only an absolute error, set `ERRREL` to zero.

3. This routine is designed for efficiency, not robustness. If the above error is encountered, try `QDAGS`.

Algorithm

The routine `QDNG` is designed to integrate smooth functions. This routine implements a nonadaptive quadrature procedure based on nested Paterson rules of order 10, 21, 43, and 87. These rules are positive quadrature rules with degree of accuracy 19, 31, 64, and 130, respectively. The routine `QDNG` applies these rules successively, estimating the error, until either the error estimate satisfies the user-supplied constraints or the last rule is applied. The routine `QDNG` is based on the routine `QNG` by Piessens et al. (1983).

This routine is not very robust, but for certain smooth functions it can be efficient. If QDNG should not perform well, we recommend the use of the IMSL routine QDAGS (page 589).

Example

The value of

$$\int_0^2 xe^x dx = e^2 + 1$$

is estimated. The values of the actual and estimated error are machine dependent.

```
      INTEGER    NOUT
      REAL       A, ABS, B, ERRABS, ERREST, ERROR, ERRREL, EXACT, EXP,
     &           F, RESULT
      INTRINSIC  ABS, EXP
      EXTERNAL   F, QDNG, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      A = 0.0
      B = 2.0
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.001
      CALL QDNG (F, A, B, ERRABS, ERRREL, RESULT, ERREST)
C                                 Print results
      EXACT = 1.0 + EXP(2.0)
      ERROR = ABS(RESULT-EXACT)
      WRITE (NOUT,99999) RESULT, EXACT, ERREST, ERROR
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Exact =’, F8.3, /, /,
     &       ’ Error estimate =’, 1PE10.3, 6X, ’Error =’, 1PE10.3)
      END
C
      REAL FUNCTION F (X)
      REAL       X
      REAL       EXP
      INTRINSIC  EXP
      F = X*EXP(X)
      RETURN
      END
```

Output

```
Computed =   8.389             Exact =   8.389

Error estimate = 5.000E-05      Error = 9.537E-07
```

TWODQ/DTWODQ (Single/Double precision)

Compute a two-dimensional iterated integral.

Usage

```
CALL TWODQ (F, A, B, G, H, ERRABS, ERRREL, IRULE, RESULT,
            ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F(X, Y)`, where
`X` – First argument of `F`. (Input)
`Y` – Second argument of `F`. (Input)
`F` – The function value. (Output)
`F` must be declared `EXTERNAL` in the calling program.

A — Lower limit of outer integral. (Input)

B — Upper limit of outer integral. (Input)

G — User-supplied `FUNCTION` to evaluate the lower limits of the inner integral. The form is `G(X)`, where
`X` – Only argument of `G`. (Input)
`G` – The function value. (Output)
`G` must be declared `EXTERNAL` in the calling program.

H — User-supplied `FUNCTION` to evaluate the upper limits of the inner integral. The form is `H(X)`, where
`X` – Only argument of `H`. (Input)
`H` – The function value. (Output)
`H` must be declared `EXTERNAL` in the calling program.

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

IRULE — Choice of quadrature rule. (Input)
The Gauss-Kronrod rule is used with the following points:

IRULE	**Points**
1	7-15
2	10-21
3	15-31
4	20-41
5	25-51
6	30-61

If the function has a peak singularity, use `IRULE` = 1. If the function is oscillatory, use `IRULE` = 6.

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)

ERREST — Estimate of the absolute value of the error. (Output)

Comments

1. Automatic workspace usage is

   `TWODQ` 2500 units, or
   `DTWODQ` 4500 units.

   Workspace may be explicitly provided, if desired, by use of `T2ODQ`/`DT2ODQ`. The reference is

   ```
   CALL T2ODQ (F, A, B, G, H, ERRABS, ERRREL, IRULE,
               RESULT, ERREST, MAXSUB, NEVAL, NSUBIN,
               ALIST, BLIST, RLIST, ELIST, IORD, WK,
               IWK)
   ```

   The additional arguments are as follows:

   MAXSUB — Number of subintervals allowed. (Input)
   A value of 250 is used by `TWODQ`.

   NEVAL — Number of evaluations of `F`. (Output)

   NSUBIN — Number of subintervals generated in the outer integral. (Output)

   ALIST — Array of length `MAXSUB` containing a list of the `NSUBIN` left endpoints for the outer integral. (Output)

   BLIST — Array of length `MAXSUB` containing a list of the `NSUBIN` right endpoints for the outer integral. (Output)

   RLIST — Array of length `MAXSUB` containing approximations to the `NSUBIN` integrals over the intervals defined by `ALIST`, `BLIST`, pertaining only to the outer integral. (Output)

   ELIST — Array of length `MAXSUB` containing the error estimates of the `NSUBIN` values in `RLIST`. (Output)

   IORD — Array of length `MAXSUB`. (Output)
   Let `K` be `NSUBIN` if `NSUBIN.LE.`(`MAXSUB`/2 + 2), `MAXSUB` + 1 – `NSUBIN` otherwise. Then the first `K` locations contain pointers to the error estimates over the corresponding subintervals, such that `ELIST`(`IORD`(1)), …, `ELIST`(`IORD`(`K`)) form a decreasing sequence.

   WK — Work array of length 4 `* MAXSUB`, needed to evaluate the inner integral.

   IWK — Work array of length `MAXSUB`, needed to evaluate the inner integral.

2. Informational errors

Type	Code	
4	1	The maximum number of subintervals allowed has been reached.

3	2	Roundoff error, preventing the requested tolerance from being achieved, has been detected.
3	3	A degradation in precision has been detected.

3. If EXACT is the exact value, TWODQ attempts to find RESULT such that ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT)). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine TWODQ approximates the two-dimensional iterated integral

$$\int_a^b \int_{g(x)}^{h(x)} f(x, y)\, dy\, dx$$

with the approximation returned in RESULT. An estimate of the error is returned in ERREST. The approximation is achieved by iterated calls to QDAG (page 591). Thus, this algorithm will share many of the characteristics of the routine QDAG. As in QDAG, several options are available. The absolute and relative error must be specified, and in addition, the Gauss-Kronrod pair must be specified (IRULE). The lower-numbered rules are used for less smooth integrands while the higher-order rules are more efficient for smooth (oscillatory) integrands.

Example 1

In this example, we approximate the integral

$$\int_0^1 \int_1^3 y \cos\left(x + y^2\right) dy\, dx$$

The value of the error estimate is machine dependent.

```
      INTEGER    IRULE, NOUT
      REAL       A, B, ERRABS, ERREST, ERRREL, F, G, H, RESULT
      EXTERNAL   F, G, H, TWODQ, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Set limits of integration
      A = 0.0
      B = 1.0
C                                 Set error tolerances
      ERRABS = 0.0
      ERRREL = 0.01
C                                 Parameter for oscillatory function
      IRULE = 6
      CALL TWODQ (F, A, B, G, H, ERRABS, ERRREL, IRULE, RESULT, ERREST)
C                                 Print results
      WRITE (NOUT,99999) RESULT, ERREST
99999 FORMAT (’ Result =’, F8.3, 13X, ’ Error estimate = ’, 1PE9.3)
      END
C
      REAL FUNCTION F (X, Y)
      REAL       X, Y
      REAL       COS
      INTRINSIC  COS
      F = Y*COS(X+Y*Y)
      RETURN
      END
C
      REAL FUNCTION G (X)
      REAL       X
      G = 1.0
      RETURN
      END
C
      REAL FUNCTION H (X)
      REAL       X
      H = 3.0
      RETURN
      END
```

Output

```
Result =  -0.514              Error estimate = 3.065E-06
```

Example 2

We modify the above example by assuming that the limits for the inner integral depend on *x* and, in particular, are *g*(*x*) = –2*x* and *h*(*x*) = 5*x*. The integral now becomes

$$\int_0^1 \int_{-2x}^{5x} y \cos\left(x + y^2\right) dy\, dx$$

The value of the error estimate is machine dependent.

```
C                                Declare F, G, H
      INTEGER    IRULE, NOUT
      REAL       A, B, ERRABS, ERREST, ERRREL, F, G, H, RESULT
      EXTERNAL   F, G, H, TWODQ, UMACH
C
      CALL UMACH (2, NOUT)
C                                  Set limits of integration
      A = 0.0
      B = 1.0
C                                  Set error tolerances
      ERRABS = 0.001
      ERRREL = 0.0
C                                  Parameter for oscillatory function
      IRULE = 6
      CALL TWODQ (F, A, B, G, H, ERRABS, ERRREL, IRULE, RESULT, ERREST)
C                                  Print results
      WRITE (NOUT,99999) RESULT, ERREST
99999 FORMAT (’ Computed =’, F8.3, 13X, ’ Error estimate = ’, 1PE9.3)
      END
      REAL FUNCTION F (X, Y)
      REAL       X, Y
C
      REAL       COS
      INTRINSIC  COS
C
      F = Y*COS(X+Y*Y)
      RETURN
      END
      REAL FUNCTION G (X)
      REAL       X
C
      G = -2.0*X
      RETURN
      END
      REAL FUNCTION H (X)
      REAL       X
C
      H = 5.0*X
      RETURN
      END
```

Output

```
Computed =  -0.083              Error estimate = 2.095E-06
```

QAND/DQAND (Single/Double precision)

Integrate a function on a hyper-rectangle.

Usage

```
CALL QAND (F, N, A, B, ERRABS, ERRREL, MAXFCN, RESULT,
           ERREST)
```

Arguments

F — User-supplied `FUNCTION` to be integrated. The form is `F(N, X)`, where

`N` – The dimension of the hyper-rectangle. (Input)
`X` – The independent variable of dimension `N`. (Input)
`F` – The value of the integrand at `X`. (Output)

`F` must be declared `EXTERNAL` in the calling program.

N — The dimension of the hyper-rectangle. (Input)
`N` must be less than or equal to 20.

A — Vector of length `N`. (Input)
Lower limits of integration.

B — Vector of length `N`. (Input)
Upper limits of integration.

ERRABS — Absolute accuracy desired. (Input)

ERRREL — Relative accuracy desired. (Input)

MAXFCN — Approximate maximum number of function evaluations to be permitted. (Input)

`MAXFCN` cannot be greater than 256^N.

RESULT — Estimate of the integral from `A` to `B` of `F`. (Output)
The integral of `F` is approximated over the `N`-dimensional hyper-rectangle `A.LE.X.LE.B`.

ERREST — Estimate of the absolute value of the error.   (Output)

Comments

1. Informational errors

Type	Code	
3	1	MAXFCN was set greater than 256^N.
4	2	The maximum number of function evaluations has been reached, and convergence has not been attained.

2. If EXACT is the exact value, QAND attempts to find RESULT such that ABS(EXACT – RESULT).LE.MAX(ERRABS, ERRREL * ABS(EXACT)). To specify only a relative error, set ERRABS to zero. Similarly, to specify only an absolute error, set ERRREL to zero.

Algorithm

The routine QAND approximates the *n*-dimensional iterated integral

$$\int_{a_1}^{b_1} \ldots \int_{a_n}^{b_n} f\left(x_1, \ldots, x_n\right) dx_n \ldots dx_1$$

with the approximation returned in RESULT. An estimate of the error is returned in ERREST. The approximation is achieved by iterated applications of product Gauss formulas. The integral is first estimated by a two-point tensor product formula in each direction. Then for *i* = 1, …, *n* the routine calculates a new estimate by doubling the number of points in the *i*-th direction, but halving the number immediately afterwards if the new estimate does not change appreciably. This process is repeated until either one complete sweep results in no increase in the number of sample points in any dimension, or the number of Gauss points in one direction exceeds 256, or the number of function evaluations needed to complete a sweep would exceed MAXFCN.

Example 1

In this example, we approximate the integral of

$$e^{-\left(x_1^2 + x_2^2 + x_3^2\right)}$$

on an expanding cube. The values of the error estimates are machine dependent. The exact integral over

$$\mathbf{R}^3 \text{is } \pi^{3/2}$$

```
      INTEGER    I, J, MAXFCN, N, NOUT
      REAL       A(3), B(3), CONST, ERRABS, ERREST, ERRREL, F, RESULT
      EXTERNAL   F, QAND, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      N      = 3
      MAXFCN = 100000
C                                 Set error tolerances
```

```
      ERRABS = 0.0001
      ERRREL = 0.001
C
      DO 20  I=1, 6
         CONST = I/2.0
C                                Set limits of integration
C                                As CONST approaches infinity, the
C                                answer approaches PI**1.5
         DO 10  J=1, 3
            A(J) = -CONST
            B(J) = CONST
   10  CONTINUE
         CALL QAND (F, N, A, B, ERRABS, ERRREL, MAXFCN, RESULT, ERREST)
         WRITE (NOUT,99999) CONST, RESULT, ERREST
   20 CONTINUE
99999 FORMAT (1X, ’For CONST = ’, F4.1, ’, result = ’, F7.3, ’ with ’,
     &        ’error estimate ’, 1PE10.3)
      RETURN
      END
C
      REAL FUNCTION F (N, X)
      INTEGER    N
      REAL       X(N)
      REAL       EXP
      INTRINSIC  EXP
      F = EXP(-(X(1)*X(1)+X(2)*X(2)+X(3)*X(3)))
      RETURN
      END
```

Output

```
For CONST =  0.5, result =   0.785 with error estimate  3.934E-06
For CONST =  1.0, result =   3.332 with error estimate  2.100E-03
For CONST =  1.5, result =   5.021 with error estimate  1.192E-05
For CONST =  2.0, result =   5.491 with error estimate  2.413E-04
For CONST =  2.5, result =   5.561 with error estimate  4.232E-03
For CONST =  3.0, result =   5.568 with error estimate  2.580E-04
```

GQRUL/DGQRUL (Single/Double precision)

Compute a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.

Usage

```
CALL GQRUL (N, IWEIGH, ALPHA, BETA, NFIX, QXFIX, QX, QW)
```

Arguments

N — Number of quadrature points.   (Input)

IWEIGH — Index of the weight function.   (Input)

IWEIGH	WT(X)	Interval	Name
1	1	$(-1, +1)$	Legendre
2	$1/\sqrt{1-X^2}$	$(-1, +1)$	Chebyshev 1st kind
3	$\sqrt{1-X^2}$	$(-1, +1)$	Chebyshev 2nd kind
4	e^{-X^2}	$(-\infty, +\infty)$	Hermite
5	$(1-X)^{\alpha}(1+X)^{\beta}$	$(-1, +1)$	Jacobi
6	$e^{-X}X^{\alpha}$	$(0, +\infty)$	Generalized Laguerre
7	$1/\cosh(X)$	$(-\infty, +\infty)$	COSH

ALPHA — Parameter used in the weight function with some values of IWEIGH, otherwise it is ignored. (Input)

BETA — Parameter used in the weight function with some values of IWEIGH, otherwise it is ignored. (Input)

NFIX — Number of fixed quadrature points. (Input)
NFIX = 0, 1 or 2. For the usual Gauss quadrature rules, NFIX = 0.

QXFIX — Array of length NFIX (ignored if NFIX = 0) containing the preset quadrature point(s). (Input)

QX — Array of length N containing quadrature points. (Output)

QW — Array of length N containing quadrature weights. (Output)

Comments

1. Automatic workspace usage is

   GQRUL N units, or
   DGQRUL 2 * N units.

   Workspace may be explicitly provided, if desired, by use of G2RUL/DG2RUL. The reference is

   ```
   CALL G2RUL (N, IWEIGH, ALPHA, BETA, NFIX, QXFIX, QX,
               QW, WK)
   ```

   The additional argument is

   WK — Work array of length N.

2. If IWEIGH specifies the weight WT(X) and the interval (*a*, *b*), then approximately

$$\int_a^b F(X) * WT(X) dX = \sum_{I=1}^{N} F\big(QX(I)\big) * QW(I)$$

3. Gaussian quadrature is always the method of choice when the function F(X) behaves like a polynomial. Gaussian quadrature is also useful on infinite intervals (with appropriate weight functions), because other techniques often fail.

4. The weight function 1/cosh(X) behaves like a polynomial near zero and like $e^{|X|}$ far from zero.

Algorithm

The routine GQRUL produces the points and weights for the Gauss, Gauss-Radau, or Gauss-Lobatto quadrature formulas for some of the most popular weights. In fact, it is slightly more general than this suggests because the extra one or two points that may be specified do not have to lie at the endpoints of the interval. This routine is a modification of the subroutine GAUSSQUADRULE (Golub and Welsch 1969).

In the simple case when NFIX = 0, the routine returns points in x = QX and weights in w = QW so that

$$\int_a^b f(x) w(x)\, dx = \sum_{i=1}^{N} f(x_i) w_i$$

for all functions f that are polynomials of degree less than 2N.

If NFIX = 1, then one of the above x_i equals the first component of QXFIX. Similarly, if NFIX = 2, then two of the components of x will equal the first two components of QXFIX. In general, the accuracy of the above quadrature formula degrades when NFIX increases. The quadrature rule will integrate all functions f that are polynomials of degree less than 2N – NFIX.

Example 1

In this example, we obtain the classical Gauss-Legendre quadrature formula, which is accurate for polynomials of degree less than 2N, and apply this when N = 6 to the function x^8 on the interval [–1, 1]. This quadrature rule is accurate for polynomials of degree less than 12.

```
      PARAMETER (N=6)
      INTEGER    I, IWEIGH, NFIX, NOUT
      REAL       ALPHA, ANSWER, BETA, QW(N), QX(N), QXFIX(2), SUM
      EXTERNAL   GQRUL, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      IWEIGH = 1
      ALPHA  = 0.0
      BETA   = 0.0
      NFIX   = 0
C                                 Get points and weights from GQRUL
      CALL GQRUL (N, IWEIGH, ALPHA, BETA, NFIX, QXFIX, QX, QW)
C                                 Write results from GQRUL
      WRITE (NOUT,99998) (I,QX(I),I,QW(I),I=1,N)
99998 FORMAT (6(6X,'QX(',I1,') = ',F8.4,7X,'QW(',I1,') = ',F8.5,/))
C                                 Evaluate the integral from these
C                                 points and weights
      SUM = 0.0
      DO 10  I=1, N
         SUM = SUM + QX(I)**8*QW(I)
   10 CONTINUE
      ANSWER = SUM
      WRITE (NOUT,99999) ANSWER
99999 FORMAT (/, ' The quadrature result making use of these ',
     &       'points and weights is ', 1PE10.4, '.')
      END
```

Output

```
QX(1) =  -0.9325       QW(1) =  0.17132
QX(2) =  -0.6612       QW(2) =  0.36076
QX(3) =  -0.2386       QW(3) =  0.46791
QX(4) =   0.2386       QW(4) =  0.46791
QX(5) =   0.6612       QW(5) =  0.36076
QX(6) =   0.9325       QW(6) =  0.17132

The quadrature result making use of these points and weights is 2.2222E-01.
```

Example 2

We modify Example 1 by requiring that both endpoints be included in the quadrature formulas and again apply the new formulas to the function x^8 on the interval [−1, 1]. This quadrature rule is accurate for polynomials of degree less than 10.

```
      PARAMETER (N=6)
      INTEGER    I, IWEIGH, NFIX, NOUT
      REAL       ALPHA, ANSWER, BETA, QW(N), QX(N), QXFIX(2), SUM
      EXTERNAL   GQRUL, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      IWEIGH   = 1
      ALPHA    = 0.0
      BETA     = 0.0
      NFIX     = 2
      QXFIX(1) = -1.0
      QXFIX(2) = 1.0
C                                 Get points and weights from GQRUL
      CALL GQRUL (N, IWEIGH, ALPHA, BETA, NFIX, QXFIX, QX, QW)
C                                 Write results from GQRUL
      WRITE (NOUT,99998) (I,QX(I),I,QW(I),I=1,N)
99998 FORMAT (6(6X,'QX(',I1,') = ',F8.4,7X,'QW(',I1,') = ',F8.5,/))
C                                 Evaluate the integral from these
C                                 points and weights
      SUM = 0.0
      DO 10  I=1, N
```

```
         SUM = SUM + QX(I)**8*QW(I)
   10 CONTINUE
      ANSWER = SUM
      WRITE (NOUT,99999) ANSWER
99999 FORMAT (/, ' The quadrature result making use of these ',
     &       'points and weights is ', 1PE10.4, '.')
      END
```

Output

```
QX(1) =  -1.0000       QW(1) =  0.06667
QX(2) =  -0.7651       QW(2) =  0.37847
QX(3) =  -0.2852       QW(3) =  0.55486
QX(4) =   0.2852       QW(4) =  0.55486
QX(5) =   0.7651       QW(5) =  0.37847
QX(6) =   1.0000       QW(6) =  0.06667

The quadrature result making use of these points and weights is 2.2222E-01.
```

GQRCF/DGQRCF (Single/Double precision)

Compute a Gauss, Gauss-Radau or Gauss-Lobatto quadrature rule given the recurrence coefficients for the monic polynomials orthogonal with respect to the weight function.

Usage

`CALL GQRCF (N, B, C, NFIX, QXFIX, QX, QW)`

Arguments

N — Number of quadrature points. (Input)

B — Array of length `N` containing the recurrence coefficients. (Input)
See Comments for definitions.

C — Array of length `N` containing the recurrence coefficients. (Input)
See Comments for definitions.

NFIX — Number of fixed quadrature points. (Input)
`NFIX` = 0, 1 or 2. For the usual Gauss quadrature rules `NFIX` = 0.

QXFIX — Array of length `NFIX` (ignored if `NFIX` = 0) containing the preset quadrature point(s). (Input)

QX — Array of length `N` containing quadrature points. (Output)

QW — Array of length `N` containing quadrature weights. (Output)

Comments

1. Automatic workspace usage is

   `GQRCF` `N` units, or
   `DGQRCF` 2 * `N` units.

Workspace may be explicitly provided, if desired, by use of G2RCF/DG2RCF. The reference is

CALL G2RCF (N, B, C, NFIX, QXFIX, QX, QW, WK)

The additional argument is

WK — Work array of length N.

2. Informational error

Type	Code	
4	1	No convergence in 100 iterations.

3. The recurrence coefficients B(I) and C(I) define the monic polynomials via the relation P(I) = (X – B(I + 1)) * P(I – 1) – C(I + 1) * P(I – 2). C(1) contains the zero-th moment

$$\int WT(X)\,dX$$

of the weight function. Each element of C must be greater than zero.

4. If *WT*(*X*) is the weight specified by the coefficients and the interval is (*a*, *b*), then approximately

$$\int_a^b F(X) * WT(X)dX = \sum_{I=1}^{N} F\big(QX(I)\big) * QW(I)$$

5. Gaussian quadrature is always the method of choice when the function F(X) behaves like a polynomial. Gaussian quadrature is also useful on infinite intervals (with appropriate weight functions) because other techniques often fail.

Algorithm

The routine GQRCF produces the points and weights for the Gauss, Gauss-Radau, or Gauss-Lobatto quadrature formulas given the three-term recurrence relation for the orthogonal polynomials. In particular, it is assumed that the orthogonal polynomials are monic, and hence, the three-term recursion may be written as

$$p_i(x) = \left(x - b_i\right)p_{i-1}(x) - c_i p_{i-2}(x) \qquad \text{for } i = 1, \ldots, N$$

where p_0 = 1 and p_{-1} = 0. It is obvious from this representation that the degree of p_i is i and that p_i is monic. In order for the recurrence to give rise to a sequence of orthogonal polynomials (with respect to a nonnegative measure), it is necessary and sufficient that $c_i > 0$. This routine is a modification of the subroutine GAUSSQUADRULE (Golub and Welsch 1969). In the simple case when NFIX = 0, the routine returns points in x = QX and weights in w = QW so that

$$\int_a^b f(x)w(x)\,dx = \sum_{i=1}^{N} f\left(x_i\right)w_i$$

for all functions *f* that are polynomials of degree less than 2*N*. Here, *w* is any weight function for which the above recurrence produces the orthogonal polynomials p_i on the interval [*a*, *b*] and *w* is normalized by

$$\int_a^b w(x)\,dx = c_1$$

If NFIX = 1, then one of the above x_i equals the first component of QXFIX. Similarly, if NFIX = 2, then two of the components of *x* will equal the first two components of QXFIX. In general, the accuracy of the above quadrature formula degrades when NFIX increases. The quadrature rule will integrate all functions *f* that are polynomials of degree less than 2*N* – NFIX.

Example

We compute the Gauss quadrature rule (with *N* = 6) for the Chebyshev weight, (1 + $x^2)^{(-1/2)}$, from the recurrence coefficients. These coefficients are obtained by a call to the IMSL routine RECCF (page 628).

```
      PARAMETER (N=6)
      INTEGER    I, NFIX, NOUT
      REAL       B(N), C(N), QW(N), QX(N), QXFIX(2)
      EXTERNAL   GQRCF, RECCF, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Recursion coefficients will come from
C                                 routine RECCF.
C                                 The call to RECCF finds recurrence
C                                 coefficients for Chebyshev
C                                 polynomials of the 1st kind.
      IWEIGH = 1
      ALPHA  = 0.0
      BETA   = 0.0
      CALL RECCF (N, IWEIGH, ALPHA, BETA, B, C)
C
      NFIX = 0
C                                 The call to GQRCF will compute the
C                                 quadrature rule from the recurrence
C                                 coefficients determined above.
      CALL GQRCF (N, B, C, NFIX, QXFIX, QX, QW)
      WRITE (NOUT,99999) (I,QX(I),I,QW(I),I=1,N)
99999 FORMAT (6(6X,'QX(',I1,') = ',F8.4,7X,'QW(',I1,') = ',F8.5,/))
C
      END
```

Output

```
QX(1) =  -0.9325       QW(1) =  0.17132
QX(2) =  -0.6612       QW(2) =  0.36076
QX(3) =  -0.2386       QW(3) =  0.46791
QX(4) =   0.2386       QW(4) =  0.46791
QX(5) =   0.6612       QW(5) =  0.36076
QX(6) =   0.9325       QW(6) =  0.17132
```

RECCF/DRECCF (Single/Double precision)

Compute recurrence coefficients for various monic polynomials.

Usage

```
CALL RECCF (N, IWEIGH, ALPHA, BETA, B, C)
```

Arguments

N — Number of recurrence coefficients. (Input)

IWEIGH — Index of the weight function. (Input)

IWEIGH	WT(X)	Interval	Name
1	1	$(-1, +1)$	Legendre
2	$1/\sqrt{1-X^2}$	$(-1, +1)$	Chebyshev 1st kind
3	$\sqrt{1-X^2}$	$(-1, +1)$	Chebyshev 2nd kind
4	e^{-X^2}	$(-\infty, +\infty)$	Hermite
5	$(1-X)^{\alpha}(1+X)^{\beta}$	$(-1, +1)$	Jacobi
6	$e^{-X}X^{\alpha}$	$(0, +\infty)$	Generalized Laguerre
7	$1/\cosh(X)$	$(-\infty, +\infty)$	COSH

ALPHA — Parameter used in the weight function with some values of `IWEIGH`, otherwise it is ignored. (Input)

BETA — Parameter used in the weight function with some values of `IWEIGH`, otherwise it is ignored. (Input)

B — Array of length `N` containing recurrence coefficients. (Output)

C — Array of length `N` containing recurrence coefficients. (Output)

Comments

The recurrence coefficients `B(I)` and `C(I)` define the monic polynomials via the relation `P(I)` = (`X` – `B(I` + 1)) `*` `P(I` – 1) – `C(I` + 1) `*` `P(I` – 2). The zero-th moment

$$\left(\int WT(X)\,dX\right)$$

of the weight function is returned in `C`(1).

Algorithm

The routine RECCF produces the recurrence coefficients for the orthogonal polynomials for some of the most important weights. It is assumed that the orthogonal polynomials are monic; hence, the three-term recursion may be written as

$$p_i(x) = (x - b_i)\, p_{i-1}(x) - c_i\, p_{i-2}(x) \qquad \text{for } i = 1, \ \ldots, \ N$$

where $p_0 = 1$ and $p_{-1} = 0$. It is obvious from this representation that the degree of p_i is i and that p_i is monic. In order for the recurrence to give rise to a sequence of orthogonal polynomials (with respect to a nonnegative measure), it is necessary and sufficient that $c_i > 0$.

Example

Here, we obtain the well-known recurrence relations for the first six *monic* Legendre polynomials, Chebyshev polynomials of the first kind, and Laguerre polynomials.

```
      PARAMETER (N=6)
      INTEGER    I, IWEIGH, NOUT
      REAL       ALPHA, B(N), BETA, C(N)
      EXTERNAL   RECCF, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      ALPHA = 0.0
      BETA  = 0.0
C
      IWEIGH = 1
      CALL RECCF (N, IWEIGH, ALPHA, BETA, B, C)
      WRITE (NOUT,99996)
      WRITE (NOUT,99999) (I,B(I),I,C(I),I=1,N)
C
      IWEIGH = 2
      CALL RECCF (N, IWEIGH, ALPHA, BETA, B, C)
      WRITE (NOUT,99997)
      WRITE (NOUT,99999) (I,B(I),I,C(I),I=1,N)
C
      IWEIGH = 6
      CALL RECCF (N, IWEIGH, ALPHA, BETA, B, C)
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (I,B(I),I,C(I),I=1,N)
C
99996 FORMAT (1X, ’Legendre’)
99997 FORMAT (/, 1X, ’Chebyshev, first kind’)
99998 FORMAT (/, 1X, ’Laguerre’)
99999 FORMAT (6(6X,’B(’,I1,’) = ’,F8.4,7X,’C(’,I1,’) = ’,F8.5,/))
      END
```

Output

```
Legendre
B(1) =   0.0000       C(1) =  2.00000
B(2) =   0.0000       C(2) =  0.33333
B(3) =   0.0000       C(3) =  0.26667
```

```
B(4) =   0.0000       C(4) =  0.25714
B(5) =   0.0000       C(5) =  0.25397
B(6) =   0.0000       C(6) =  0.25253

Chebyshev, first kind
B(1) =   0.0000       C(1) =  3.14159
B(2) =   0.0000       C(2) =  0.50000
B(3) =   0.0000       C(3) =  0.25000
B(4) =   0.0000       C(4) =  0.25000
B(5) =   0.0000       C(5) =  0.25000
B(6) =   0.0000       C(6) =  0.25000

Laguerre
B(1) =   1.0000       C(1) =  1.00000
B(2) =   3.0000       C(2) =  1.00000
B(3) =   5.0000       C(3) =  4.00000
B(4) =   7.0000       C(4) =  9.00000
B(5) =   9.0000       C(5) = 16.00000
B(6) =  11.0000       C(6) = 25.00000
```

RECQR/DRECQR (Single/Double precision)

Compute recurrence coefficients for monic polynomials given a quadrature rule.

Usage

```
CALL RECQR (N, QX, QW, NTERM, B, C)
```

Arguments

N — Number of quadrature points. (Input)

QX — Array of length N containing the quadrature points. (Input)

QW — Array of length N containing the quadrature weights. (Input)

NTERM — Number of recurrence coefficients. (Input)
NTERM must be less than or equal to N.

B — Array of length NTERM containing recurrence coefficients. (Output)

C — Array of length NTERM containing recurrence coefficients. (Output)

Comments

1. Automatic workspace usage is

   RECQR 2 * N units, or
   DRECQR 4 * N units.

   Workspace may be explicitly provided, if desired, by use of R2CQR/DR2CQR. The reference is

   ```
   CALL R2CQR (N, QX, QW, NTERM, B, C, WK)
   ```

   The additional argument is

WK — Work array of length 2 * N.

2. The recurrence coefficients B(I) and C(I) define the monic polynomials via the relation P(I) = (X – B(I + 1)) * P(I – 1) – C(I + 1) * P(I – 2). The zero-th moment

$$\left(\int WT(X)\,dX\right)$$

of the weight function is returned in C(1).

Algorithm

The routine RECQR produces the recurrence coefficients for the orthogonal polynomials given the points and weights for the Gauss quadrature formula. It is assumed that the orthogonal polynomials are monic; hence the three-term recursion may be written

$$p_i(x) = \left(x - b_i\right)p_{i-1}(x) - c_i\,p_{i-2}(x) \qquad \text{for } i = 1,\ \ldots,\ N$$

where $p_0 = 1$ and $p_{-1} = 0$. It is obvious from this representation that the degree of p_i is i and that p_i is monic. In order for the recurrence to give rise to a sequence of orthogonal polynomials (with respect to a nonnegative measure), it is necessary and sufficient that $c_i > 0$.

This routine is an inverse routine to GQRCF (page 625). Given the recurrence coefficients, the routine GQRCF produces the corresponding Gauss quadrature formula, whereas the routine RECQR produces the recurrence coefficients given the quadrature formula.

Example

To illustrate the use of RECQR, we will input a simple choice of recurrence coefficients, call GQRCF for the quadrature formula, put this information into RECQR, and recover the recurrence coefficients.

```
      PARAMETER (N=5)
      INTEGER    I, J, NFIX, NOUT, NTERM
      REAL       B(N), C(N), FLOAT, QW(N), QX(N), QXFIX(2)
      INTRINSIC  FLOAT
      EXTERNAL   GQRCF, RECQR, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
      NFIX = 0
C                                 Set arrays B and C of recurrence
C                                 coefficients
      DO 10  J=1, N
         B(J) = FLOAT(J)
         C(J) = FLOAT(J)/2.0
   10 CONTINUE
      WRITE (NOUT,99995)
99995 FORMAT (1X, ’Original recurrence coefficients’)
      WRITE (NOUT,99996) (I,B(I),I,C(I),I=1,N)
99996 FORMAT (5(6X,’B(’,I1,’) = ’,F8.4,7X,’C(’,I1,’) = ’,F8.5,/))
C
```

```
C                                 The call to GQRCF will compute the
C                                 quadrature rule from the recurrence
C                                 coefficients given above.
C
      CALL GQRCF (N, B, C, NFIX, QXFIX, QX, QW)
      WRITE (NOUT,99997)
99997 FORMAT (/, 1X, ’Quadrature rule from the recurrence coefficients’
     &       )
      WRITE (NOUT,99998) (I,QX(I),I,QW(I),I=1,N)
99998 FORMAT (5(6X,’QX(’,I1,’) = ’,F8.4,7X,’QW(’,I1,’) = ’,F8.5,/))
C
C                                 Call RECQR to recover the original
C                                 recurrence coefficients
      NTERM = N
      CALL RECQR (N, QX, QW, NTERM, B, C)
      WRITE (NOUT,99999)
99999 FORMAT (/, 1X, ’Recurrence coefficients determined by RECQR’)
      WRITE (NOUT,99996) (I,B(I),I,C(I),I=1,N)
C
      END
```

Output

```
Original recurrence coefficients
B(1) =   1.0000       C(1) =  0.50000
B(2) =   2.0000       C(2) =  1.00000
B(3) =   3.0000       C(3) =  1.50000
B(4) =   4.0000       C(4) =  2.00000
B(5) =   5.0000       C(5) =  2.50000

Quadrature rule from the recurrence coefficients
QX(1) =   0.1525       QW(1) =  0.25328
QX(2) =   1.4237       QW(2) =  0.17172
QX(3) =   2.7211       QW(3) =  0.06698
QX(4) =   4.2856       QW(4) =  0.00790
QX(5) =   6.4171       QW(5) =  0.00012

Recurrence coefficients determined by RECQR
B(1) =   1.0000       C(1) =  0.50000
B(2) =   2.0000       C(2) =  1.00000
B(3) =   3.0000       C(3) =  1.50000
B(4) =   4.0000       C(4) =  2.00000
B(5) =   5.0000       C(5) =  2.50000
```

FQRUL/DFQRUL (Single/Double precision)

Compute a Fejér quadrature rule with various classical weight functions.

Usage

```
CALL FQRUL (N, A, B, IWEIGH, ALPHA, BETA, QX, QW)
```

Arguments

N — Number of quadrature points.   (Input)

A — Lower limit of integration.   (Input)

B — Upper limit of integration.   (Input)
B must be greater than A.

IWEIGH — Index of the weight function.   (Input)

IWEIGH	WT(X)
1	1
2	1/(X – ALPHA)
3	$(B - X)^{\alpha} (X - A)^{\beta}$
4	$(B - X)^{\alpha} (X - A)^{\beta} \log(X - A)$
5	$(B - X)^{\alpha} (X - A)^{\beta} \log(B - X)$

ALPHA — Parameter used in the weight function (except if IWEIGH = 1, it is ignored).   (Input)
If IWEIGH = 2, then it must satisfy A.LT.ALPHA.LT.B. If IWEIGH = 3, 4, or 5, then ALPHA must be greater than –1.

BETA — Parameter used in the weight function (ignored if IWEIGH = 1 or 2). (Input)
BETA must be greater than –1.0.

QX — Array of length N containing quadrature points.   (Output)

QW — Array of length N containing quadrature weights.   (Output)

Comments

1. Automatic workspace usage is

   FQRUL   3 * N + 15 units, or
   DFQRUL 6 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of F2RUL/DF2RUL. The reference is

   ```
   CALL F2RUL (N, A, B, IWEIGH, ALPHA, BETA, QX, QW,
               WK)
   ```

   The additional argument is

   WK — Work array of length 3 * N + 15.

2. If IWEIGH specifies the weight WT(X) and the interval (A, B), then approximately

   $$\int_A^B F(X) * WT(X) dX = \sum_{I=1}^{N} F\left(QX(I)\right) * QW(I)$$

3. The routine FQRUL uses an FFT, so it is most efficient when N is the product of small primes.

Algorithm

The routine FQRUL produces the weights and points for the Fejér quadrature rule. Since this computation is based on a quarter-wave cosine transform, the computations are most efficient when N, the number of points, is a product of small primes. These quadrature formulas may be an intermediate step in a more complicated situation, see for instance Gautschi and Milovanofic (1985).

The Fejér quadrature rules are based on polynomial interpolation. First, choose classical abscissas (in our case, the Gauss points for the Chebyshev weight function $(1 - x^2)^{-1/2}$), then derive the quadrature rule for a different weight. In order to keep the presentation simple, we will describe the case where the interval of integration is [–1, 1] even though FQRUL allows rescaling to an arbitrary interval $[a, b]$.

We are looking for quadrature rules of the form

$$Q(f) := \sum_{j=1}^{N} w_j f\left(x_j\right)$$

where the

$$\{x_j\}_{j=1}^{N}$$

are the zeros of the N-th Chebyshev polynomial (of the first kind) $T_N(x) = \cos(N \arccos x)$. The weights in the quadrature rule Q are chosen so that, for all polynomials p of degree less than N,

$$Q(p) = \sum_{j=1}^{N} w_j p\left(x_j\right) = \int_{-1}^{1} p(x) w(x)\, dx$$

for some weight function w. In FQRUL, the user has the option of choosing w from five families of functions with various algebraic and logarithmic endpoint singularities.

These Fejér rules are important because they can be computed using specialized FFT quarter-wave transform routines. This means that rules with a large number of abscissas may be computed efficiently. If we insert T_l for p in the above formula, we obtain

$$Q(T_l) = \sum_{j=1}^{N} w_j T_l\left(x_j\right) = \int_{-1}^{1} T_l(x) w(x)\, dx$$

for l = 0, …, N – 1. This is a system of linear equations for the unknown weights w_j that can be simplified by noting that

$$x_j = \cos\frac{(2j-1)\pi}{2N} \qquad j = 1, \ldots, N$$

and hence,

$$\begin{aligned}\int_{-1}^{1} T_l(x)w(x)\,dx &= \sum_{j=1}^{N} w_j T_l\left(x_j\right)\\ &= \sum_{j=1}^{N} w_j \cos\frac{l(2j-1)\pi}{2N}\end{aligned}$$

The last expression is the cosine quarter-wave forward transform for the sequence

$$\{w_j\}_{j=1}^{N}$$

that is implemented in Chapter 6 under the name QCOSF (page 791). More importantly, QCOSF has an inverse QCOSB (page 793). It follows that if the integrals on the left in the last expression can be computed, then the Fejér rule can be derived efficiently for highly composite integers N utilizing QCOSB. For more information on this topic, consult Davis and Rabinowitz (1984, pages 84−86) and Gautschi (1968, page 259).

Example

Here, we obtain the Fejér quadrature rules using 10, 100, and 200 points. With these rules, we get successively better approximations to the integral

$$\int_0^1 x\sin\left(41\pi x^2\right)dx = \frac{1}{41\pi}$$

```
      PARAMETER (NMAX=200)
      INTEGER    I, IWEIGH, K, N, NFIX, NOUT
      REAL       A, ALPHA, ANSWER, B, BETA, CONST, F, QW(NMAX),
     &           QX(NMAX), SIN, SUM, X, PI, ERROR
      INTRINSIC  SIN, ABS
      EXTERNAL   CONST, FQRUL, UMACH
C
      F(X) = X*SIN(41.0*PI*X**2)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C
      PI = CONST('PI')
      DO 20  K=1, 3
         IF (K .EQ. 1) N = 10
         IF (K .EQ. 2) N = 100
         IF (K .EQ. 3) N = 200
         A      = 0.0
         B      = 1.0
         IWEIGH = 1
         ALPHA  = 0.0
         BETA   = 0.0
         NFIX   = 0
C                                  Get points and weights from FQRUL
         CALL FQRUL (N, A, B, IWEIGH, ALPHA, BETA, QX, QW)
C                                  Evaluate the integral from these
C                                  points and weights
         SUM = 0.0
         DO 10  I=1, N
            SUM = SUM + F(QX(I))*QW(I)
   10  CONTINUE
         ANSWER = SUM
         ERROR = ABS(ANSWER - 1.0/(41.0*PI))
         WRITE (NOUT,99999) N, ANSWER, ERROR
   20 CONTINUE
C
99999 FORMAT (/, 1X, ’When N = ’, I3, ’, the quadrature result making ’
     &       , ’use of these points ’, /, ’ and weights is ’, 1PE11.4,
     &       ’, with error ’, 1PE9.2, ’.’)
      END
```

Output

```
When N =  10, the quadrature result making use of these points and weights
is -1.6523E-01, with error  1.73E-01.

When N = 100, the quadrature result making use of these points and weights
is  7.7637E-03, with error  2.79E-08.

When N = 200, the quadrature result making use of these points and weights
is  7.7636E-03, with error  1.40E-08.
```

DERIV/DDERIV (Single/Double precision)

Compute the first, second or third derivative of a user-supplied function.

Usage

`DERIV(FCN, KORDER, X, BGSTEP, TOL)`

Arguments

FCN — User-supplied `FUNCTION` whose derivative at `X` will be computed. The form is `FCN(X)`, where

`X` – Independent variable. (Input)
`FCN` – The function value. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

KORDER — Order of the derivative desired (1, 2 or 3). (Input)

X — Point at which the derivative is to be evaluated. (Input)

BGSTEP — Beginning value used to compute the size of the interval used in computing the derivative. (Input)
The interval used is the closed interval (`X` – 4 `*` `BGSTEP`, `X` + 4 `*` `BGSTEP`). `BGSTEP` must be positive.

TOL — Relative error desired in the derivative estimate. (Input)

DERIV — Estimate of the first (`KORDER` = 1), second (`KORDER` = 2) or third (`KORDER` = 3) derivative of `FCN` at `X`. (Output)

Comments

1. Informational errors

Type	Code	
3	2	Roundoff error became dominant before estimates converged. Increase precision and/or increase `BGSTEP`.
4	1	Unable to achieve desired tolerance in derivative estimation. Increase precision, increase `TOL` and/or change `BGSTEP`. If this error continues, the function may not have a derivative at `X`.

2. Convergence is assumed when

$$\frac{2}{3}\left|\mathrm{D2} - \mathrm{D1}\right| < \mathrm{TOL}$$

for two successive derivative estimates `D1` and `D2`.

3. The initial step size, `BGSTEP`, must be chosen small enough that `FCN` is defined and reasonably smooth in the interval (`X` – 4 `*` `BGSTEP`, `X` + 4 `*` `BGSTEP`), yet large enough to avoid roundoff problems.

Algorithm

`DERIV` produces an estimate to the first, second, or third derivative of a function. The estimate originates from first computing a spline interpolant to the input function using values within the interval (`X` – 4.0 `*` `BGSTEP`, `X` + 4.0 `*` `BGSTEP`), then differentiating the spline at `X`.

Example 1

In this example, we obtain the approximate first derivative of the function

$$f(x) = -2\sin(3x/2)$$

at the point $x = 2$.

```
      INTEGER    KORDER, NCOUNT, NOUT
      REAL       BGSTEP, DERIV, DERV, FCN, TOL, X
      EXTERNAL   DERIV, FCN, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      X      = 2.0
      BGSTEP = 0.2
      TOL    = 0.01
      KORDER = 1
      NCOUNT = 1
      DERV   = DERIV(FCN,KORDER,X,BGSTEP,TOL)
      WRITE (NOUT,99999) DERV
99999 FORMAT (/, 1X, ’First derivative of FCN is ’, 1PE10.3)
      END
C
      REAL FUNCTION FCN (X)
      REAL       X
      REAL       SIN
      INTRINSIC  SIN
      FCN = -2.0*SIN(1.5*X)
      RETURN
      END
```

Output

```
First derivative of FCN is  2.970E+00
```

Example 2

In this example, we attempt to approximate in single precision the third derivative of the function

$$f(x) = 2x^4 + 3x$$

at the point $x = 0.75$. Although the function is well-behaved near $x = 0.75$, finding derivatives is often computationally difficult on 32-bit machines. The difficulty is overcome in double precision.

```
      INTEGER    KORDER, NOUT
      REAL       BGSTEP, DERIV, DERV, FCN, TOL, X
      DOUBLE PRECISION DBGSTE, DDERIV, DDERV, DFCN, DTOL, DX
      EXTERNAL   DDERIV, DERIV, DFCN, ERSET, FCN, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Turn off stopping due to error
C                                 condition
      CALL ERSET (0, -1, 0)
C
      X      = 0.75
      BGSTEP = 0.1
      TOL    = 0.01
      KORDER = 3
C                                 In single precision, on a 32-bit
C                                 machine, the following attempt
C                                 produces an error message
      DERV = DERIV(FCN,KORDER,X,BGSTEP,TOL)
C                                 In double precision, we get good
C                                 results
      DX     = 0.75D0
      DBGSTE = 0.1D0
      DTOL   = 0.01D0
      KORDER = 3
      DDERV  = DDERIV(DFCN,KORDER,DX,DBGSTE,DTOL)
      WRITE (NOUT,99999) DDERV
99999 FORMAT (/, 1X, ’The third derivative of DFCN is ’, 1PD10.4)
      END
C
      REAL FUNCTION FCN (X)
      REAL       X
      FCN = 2.0*X**4 + 3.0*X
      RETURN
      END
C
      DOUBLE PRECISION FUNCTION DFCN (X)
      DOUBLE PRECISION X
      DFCN = 2.0D0*X**4 + 3.0D0*X
      RETURN
      END
```

Output

```
*** FATAL    ERROR 1 from DERIV.  Unable to achieve desired tolerance.
***          Increase precision, increase TOL = 1.000000E-02 and/or change
***          BGSTEP = 1.000000E-01.  If this error continues the function
***          may not have a derivative at X = 7.500000E-01

The third derivative of DFCN is 3.6000D+01
```

Chapter 5: Differential Equations

Routines

Usage Notes

A *differential equation* is an equation involving one or more dependent variables (called y_i or u_i), their derivatives, and one or more independent variables (called t, x, and y). Users will typically need to relabel their own model variables so that they correspond to the variables used in the solvers described here. A differential equation with one independent variable is called an *ordinary differential equation* (ODE). A system of equations involving derivatives in one independent variable and other dependent variables is called a *differential-*

algebraic system. A differential equation with more than one independent variable is called a *partial differential equation* (PDE). The *order* of a differential equation is the highest order of any of the derivatives in the equation. When the user’s model contains derivatives of order higher than the value one, it is usually necessary to substitute for dependent variables with higher order derivatives in order to use most of the software in this chapter.

Ordinary differential equations

It is convenient to use the vector notation below. We denote the number of equations as the value *N*. The problem statement is abbreviated by writing it as a *system* of first-order ODEs

$$y(t) = \left[y_1(t), \ldots, y_N(t)\right]^T, \ f(t, y) = \left[\, f_1(t, y), \ldots, \ f_N(t, y)\right]^T$$

The problem becomes

$$y' = \frac{dy(t)}{dt} = f(t, y)$$

with initial values $y\,(t_0)$. Values of $y(t)$ for $t > t_0$ or $t < t_0$ are required. The routines `IVPRK`, page 645, `IVMRK`, page 652, and `IVPAG`, page 646, solve the IVP for systems of ODEs of the form $y' = f\,(t, y)$ with $y(t = t_0)$ specified. Here, *f* is a user supplied function that must be evaluated at any set of values $(t, y_1, \ldots, y_N)$; $i = 1, \ldots, N$. The routines `IVPAG` and `DASPG`, page 696, will also solve implicit systems of the form $Ay' = f\,(t, y)$ where *A* is a user supplied matrix. For `IVPAG`, the matrix *A* must be nonsingular.

The system $y' = f\,(t, y)$ is said to be *stiff* if some of the eigenvalues of the Jacobian matrix $\{\partial f_i/\partial y_j\}$ are large, with negative real parts. This is often the case for differential equations representing the behavior of physical systems such as chemical reactions proceeding to equilibrium where subspecies effectively complete their reaction in different epochs. An alternate model concerns discharging capacitors such that different parts of the system have widely varying decay rates (or *time constants*). This definition of stiffness, based on the eigenvalues of the Jacobian matrix, is not satisfactory. Users typically identify stiff systems by the fact that numerical differential equation solvers such as `IVPRK`, page 645, are inefficient, or else they fail. The most common inefficiency is that a large number of evaluations of the functions f_i are required. In such cases, use routine `IVPAG`, page 661, or `DASPG`, page 696. For more about stiff systems, see Gear (1971, Chapter 11) or Shampine and Gear (1979).

In the *boundary value problem* (BVP) for ODEs, constraints on the dependent variables are given at the endpoints of the interval of interest, [*a*, *b*]. The routines `BVPFD`, page 678, and `BVPMS`, page 689, solve the BVP for systems of the form $y'(t) = f\,(t, y)$, subject to the conditions

$$h_i(y_1(a), \ldots, y_N(a), y_1(b), \ldots, y_N(b)) = 0 \quad i = 1, \ldots, N$$

Here, *f* and $h = [h_1, \ldots, h_N]^T$ are user-supplied functions.

Differential-algebraic equations

Frequently, it is not possible or not convenient to express the model of a dynamical system as a set of ODEs. Rather, an implicit equation is available in the form

$$g_i\left(t, y, \ldots, y_N, y_1', \ldots, y_N'\right) = 0 \quad i = 1, \ldots, N$$

The g_i are user-supplied functions. The system is abbreviated as

$$g(t, y, y') = \left[g_1(t, y, y'), \ldots, g_N(t, y, y')\right]^T = 0$$

Initial conditions for this problem include both $y(t_0)$ and $y'(t_0)$. Any system of ODEs can be trivially written as a differential-algebraic system by defining

$$g(t, y, y') = f(t, y) - y'$$

The routine `DASPG`, page 696, solves differential-algebraic systems of index 1 or index 0. For a definition of *index* of a differential-algebraic system, see (Brenan et al. 1989). Also, see Gear and Petzold (1984) for an outline of the computing methods used.

Partial differential equations

The routine `MOLCH`, page 717, solves the IVP problem for systems of the form

$$\frac{\partial u_i}{\partial t} = f_i\left(x, t, u_1, \ldots, u_N, \frac{\partial u_1}{\partial x}, \ldots, \frac{\partial u_N}{\partial x}, \frac{\partial^2 u_1}{\partial x^2}, \ldots, \frac{\partial^2 u_N}{\partial x^2}\right)$$

subject to the boundary conditions

$$\begin{aligned} \alpha_1^{(i)} u_i(a) + \beta_1^{(i)} \frac{\partial u_i}{\partial x}(a) &= \gamma_1(t) \\ \alpha_2^{(i)} u_i(b) + \beta_2^{(i)} \frac{\partial u_i}{\partial x}(b) &= \gamma_2(t) \end{aligned}$$

and subject to the initial conditions

$$u_i(x, t = t_0) = g_i(x)$$

for $i = 1, \ldots, N$. Here, f_i, g_i,

$$\alpha_j^{(i)}, \text{and } \beta_j^{(i)}$$

are user-supplied, $j = 1, 2$.

The routines `FPS2H`, page 734, and `FPS3H`, page 739, solve Laplace’s, Poisson’s, or Helmholtz’s equation in two or three dimensions. `FPS2H` uses a fast Poisson method to solve a PDE of the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + cu = f(x, y)$$

over a rectangle, subject to boundary conditions on each of the four sides. The scalar constant c and the function f are user specified. FPS3H solves the three-dimensional analogue of this problem.

Summary

The following table summarizes the types of problems handled by the routines in this chapter. With the exception of FPS2H and FPS3H, the routines can handle more than one differential equation.

Problem	Consideration	Routine
$Ay' = f(t, y)$ $y(t_0) = y_0$	A is a general, symmetric positive definite, band or symmetric positive definite band matrix.	IVPAG page 646
	Stiff or expensive to evaluate $f(t, y)$, banded Jacobian or finely space output needed.	IVPAG page 646
$y' = f(t, y)$, $y(t_0) = y_0$	High accuracy needed and not stiff. (Use Adams methods)	IVPAG page 646
	Moderate accuracy needed and not stiff.	IVPRK page 645
$y' = f(t, y)$ $h(y(a), y(b)) = 0$	BVP solver using finite differences	BVPFD page 678
	BVP solver using multiple shooting	BVPMS page 689
$g(t, y, y') = 0$ $y(t_0)$, $y'(t_0)$ given	Stiff, differential-algebraic solver for systems of index 1 or 0.	DASPG page 696
$u_t = f(x, t, u, u_x, u_{xx})$ $\alpha_1 u(a) + \beta_1 u_x(a) = \gamma_1(t)$ $\alpha_2 u(b) + \beta_2 u_x(b) = \gamma_2(t)$	Method of lines using cubic splines and ODEs	MOLCH page 717
$u_{xx} + u_{yy} + cu = f(x, y)$ on a rectangle, given u or u_n on each edge.	Fast Poisson solver	FPS2H page 734
$u_{xx} + u_{yy} + u_{zz} + cu = f(x, y, z)$ on a box, given u or u_n on each face	Fast Poisson solver	FPS3H page 739

Problem	Consideration	Routine
$-(pu')' + qu = \lambda ru,$ $\alpha_1 u(a) - \alpha_2 (pu'(a))$ $= \lambda(\alpha'_1 u(a) - \alpha'_2 (pu'(a)))$ $\beta_1 u(b) + \beta_2 (pu'(b)) = 0$	Sturm-Liouville problems	`SLEIG` page 745

IVPRK/DIVPRK (Single/Double precision)

Solve an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.

Usage

```
CALL IVPRK (IDO, N, FCN, T, TEND, TOL, PARAM, Y)
```

Arguments

IDO — Flag indicating the state of the computation. (Input/Output)

IDO	**State**
1	Initial entry
2	Normal re-entry
3	Final call to release workspace
4	Return because of interrupt 1
5	Return because of interrupt 2 with step accepted
6	Return because of interrupt 2 with step rejected

Normally, the initial call is made with `IDO` = 1. The routine then sets `IDO` = 2, and this value is used for all but the last call that is made with `IDO` = 3. This final call is used to release workspace, which was automatically allocated by the initial call with `IDO` = 1. No integration is performed on this final call. See Comment 3 for a description of the other interrupts.

N — Number of differential equations. (Input)

FCN — User-supplied `SUBROUTINE` to evaluate functions. The usage is `CALL FCN (N, T, Y, YPRIME)`, where

`N` – Number of equations. (Input)
`T` – Independent variable, *t*. (Input)
`Y` – Array of size `N` containing the dependent variable values, *y*. (Input)
`YPRIME` – Array of size `N` containing the values of the vector *y′* evaluated at (*t*, y). (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

T — Independent variable. (Input/Output)
On input, `T` contains the initial value. On output, `T` is replaced by `TEND` unless error conditions have occurred. See `IDO` for details.

TEND — Value of *t* where the solution is required. (Input)
The value TEND may be less than the initial value of *t*.

TOL — Tolerance for error control. (Input)
An attempt is made to control the norm of the local error such that the global error is proportional to TOL.

PARAM — A *floating-point* array of size 50 containing optional parameters. (Input/ Output)
If a parameter is zero, then a default value is used. These default values are given below. Parameters that concern values of step size are applied in the direction of integration. The following parameters must be set by the user:

	PARAM	**Meaning**
1	HINIT	Initial value of the step size. Default: 10.0 * MAX (AMACH (1), AMACH(4) * MAX(ABS(TEND), ABS(T)))
2	HMIN	Minimum value of the step size. Default: 0.0
3	HMAX	Maximum value of the step size. Default: 2.0
4	MXSTEP	Maximum number of steps allowed. Default: 500
5	MXFCN	Maximum number of function evaluations allowed. Default: No enforced limit.
6		Not used.
7	INTRP1	If nonzero, then return with IDO = 4 before each step. See Comment 3. Default: 0.
8	INTRP2	If nonzero, then return with IDO = 5 after every successful step and with IDO = 6 after every unsuccessful step. See Comment 3. Default: 0.
9	SCALE	A measure of the scale of the problem, such as an approximation to the average value of a norm of the Jacobian matrix along the solution. Default: 1.0

10	INORM	Switch determining error norm. In the following, e_i is the absolute value of an estimate of the error in $y_i(t)$. Default: 0. 0 – min(absolute error, relative error) = max(e_i/w_i); i = 1, … , N, where w_i = max($\|y_i(t)\|$, 1.0). 1 – absolute error = max(e_i), i = 1 …, N. 2– max(e_i/w_i), i = 1 …, N where w_i = max($\|y_i(t)\|$, FLOOR), and FLOOR is PARAM(11). 3 – Scaled Euclidean norm defined as $\mathrm{YMAX} = \sqrt{\sum_{i=1}^{N} e_i^2 / w_i^2}$ where w_i = max($\|y_i(t)\|$, 1.0). Other definitions of YMAX can be specified by the user, as explained in Comment 1.
11	FLOOR	Used in the norm computation associated with parameter INORM. Default: 1.0.
12–30		Not used.

The following entries in PARAM are set by the program.

	PARAM	**Meaning**
31	HTRIAL	Current trial step size.
32	HMINC	Computed minimum step size allowed.
33	HMAXC	Computed maximum step size allowed.
34	NSTEP	Number of steps taken.
35	NFCN	Number of function evaluations used.
36–50		Not used.

Y — Array of size N of dependent variables. (Input/Output)
On input, Y contains the initial values. On output, Y contains the approximate solution.

Comments

1. Automatic workspace usage is

IVPRK 10N units, or
DIVPRK 20N units.

Workspace may be explicitly provided, if desired, by use of I2PRK/DI2PRK. The reference is

```
CALL I2PRK (IDO, N, FCN, T, TEND, TOL, PARAM, Y,
            VNORM, WK)
```

The additional arguments are as follows:

VNORM — A Fortran SUBROUTINE to compute the norm of the error. (Input)
The routine may be provided by the user, or the IMSL routine I3PRK/DI3PRK may be used. In either case, the name must be declared in a Fortran EXTERNAL statement. If usage of the IMSL routine is intended, then the name I3PRK/DI3PRK should be used. The usage of the error norm routine is CALL VNORM (N, V, Y, YMAX, ENORM), where

Arg	**Definition**
N	Number of equations. (Input)
V	Array of size N containing the vector whose norm is to be computed. (Input)
Y	Array of size N containing the values of the dependent variable. (Input)
YMAX	Array of size N containing the maximum values of $|y(t)|$. (Input)
ENORM	Norm of the vector V. (Output)

VNORM must be declared EXTERNAL in the calling program.

WK — Work array of size 10N using the working precision. The contents of WK must not be changed from the first call with IDO = 1 until after the final call with IDO = 3.

2. Informational errors

Type	Code	
4	1	Cannot satisfy error condition. The value of TOL may be too small.
4	2	Too many function evaluations needed.
4	3	Too many steps needed. The problem may be stiff.

3. If PARAM(7) is nonzero, the subroutine returns with IDO = 4 and will resume calculation at the point of interruption if re-entered with IDO = 4. If PARAM(8) is nonzero, the subroutine will interrupt the calculations immediately after it decides whether or not to accept the result of the most recent trial step. The values used are IDO = 5 if the routine plans to accept, or IDO = 6 if it plans to reject the step. The values of IDO may be changed by the user (by changing IDO from 6 to 5) in order to force acceptance of a step that would otherwise be rejected. Some parameters the user might want to examine after return from an interrupt are IDO, HTRIAL, NSTEP, NFCN, T, and Y. The array Y contains the newly computed trial value for $y(t)$, accepted or not.

Algorithm

Routine IVPRK finds an approximation to the solution of a system of first-order differential equations of the form $y_0 = f(t, y)$ with given initial data. The routine attempts to keep the global error proportional to a user-specified tolerance. This routine is efficient for nonstiff systems where the derivative evaluations are not expensive.

The routine IVPRK is based on a code designed by Hull, Enright and Jackson (1976, 1977). It uses Runge-Kutta formulas of order five and six developed by J. H. Verner.

Example 1

Consider a predator-prey problem with rabbits and foxes. Let *r* be the density of rabbits and let *f* be the density of foxes. In the absence of any predator-prey interaction, the rabbits would increase at a rate proportional to their number, and the foxes would die of starvation at a rate proportional to their number. Mathematically,

$$r' = 2r$$

$$f' = -f$$

The rate at which the rabbits are eaten by the foxes is $2r\,f$, and the rate at which the foxes increase, because they are eating the rabbits, is $r\,f$. So, the model to be solved is

$$r' = 2r - 2r\,f$$

$$f' = -f + r\,f$$

The initial conditions are $r(0) = 1$ and $f(0) = 3$ over the interval $0 \le t \le 10$.

In the program Y(1) = *r* and Y(2) = *f*. Note that the parameter vector PARAM is first set to zero with IMSL routine SSET (page 1037). Then, absolute error control is selected by setting PARAM(10) = 1.0.

The last call to IVPRK with IDO = 3 deallocates IMSL workspace allocated on the first call to IVPRK. It is not necessary to release the workspace in this example because the program ends after solving a single problem. The call to release workspace is made as a model of what would be needed if the program included further calls to IMSL routines.

```
      INTEGER    MXPARM, N
      PARAMETER  (MXPARM=50, N=2)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, ISTEP, NOUT
      REAL       PARAM(MXPARM), T, TEND, TOL, Y(N)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPRK, SSET, UMACH, FCN
C
      CALL UMACH (2, NOUT)
C                                  Set initial conditions
      T = 0.0
      Y(1) = 1.0
      Y(2) = 3.0
C                                  Set error tolerance
      TOL = 0.0005
C                                  Set PARAM to default
      CALL SSET (MXPARM, 0.0, PARAM, 1)
C                                  Select absolute error control
      PARAM(10) = 1.0
C                                  Print header
```

```
      WRITE (NOUT,99999)
      IDO = 1
      ISTEP = 0
   10 CONTINUE
      ISTEP = ISTEP + 1
      TEND = ISTEP
      CALL IVPRK (IDO, N, FCN, T, TEND, TOL, PARAM, Y)
      IF (ISTEP .LE. 10) THEN
         WRITE (NOUT,'(I6,3F12.3)') ISTEP, T, Y
C                                  Final call to release workspace
         IF (ISTEP .EQ. 10) IDO = 3
         GO TO 10
      END IF
99999 FORMAT (4X, 'ISTEP', 5X, 'Time', 9X, 'Y1', 11X, 'Y2')
      END
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), YPRIME(N)
C
      YPRIME(1) = 2.0*Y(1) - 2.0*Y(1)*Y(2)
      YPRIME(2) = -Y(2) + Y(1)*Y(2)
      RETURN
      END
```

Output

```
 ISTEP     Time          Y1          Y2
 1      1.000       0.078       1.465
 2      2.000       0.085       0.578
 3      3.000       0.292       0.250
 4      4.000       1.449       0.187
 5      5.000       4.046       1.444
 6      6.000       0.176       2.256
 7      7.000       0.066       0.908
 8      8.000       0.148       0.367
 9      9.000       0.655       0.188
10     10.000       3.157       0.352
```

Example 2

This is a mildly stiff problem (F2) from the test set of Enright and Pryce (1987). It is included here because it illustrates the inefficiency of requiring more function evaluations with a nonstiff solver, for a requested accuracy, than would be required using a stiff solver. Also, see IVPAG, page 646, Example 2, where the problem is solved using a BDF method. The number of function evaluations may vary, depending on the accuracy and other arithmetic characteristics of the computer. The test problem has $n = 2$ equations:

$$\begin{aligned}
y_1' &= -y_1 - y_1 y_2 + k_1 y_2 \\
y_2' &= -k_2 y_2 + k_3 (1 - y_2) y_1 \\
y_1(0) &= 1 \\
y_2(0) &= 0 \\
k_1 &= 294 \\
k_2 &= 3 \\
k_3 &= 0.01020408 \\
tend &= 240
\end{aligned}$$

```
      INTEGER    MXPARM, N
      PARAMETER  (MXPARM=50, N=2)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, ISTEP, NOUT
      REAL       PARAM(MXPARM), T, TEND, TOL, Y(N)
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPRK, SSET, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN
C
      CALL UMACH (2, NOUT)
C                                 Set initial conditions
      T = 0.0
      Y(1) = 1.0
      Y(2) = 0.0
C                                 Set error tolerance
      TOL = 0.001
C                                 Set PARAM to default
      CALL SSET (MXPARM, 0.0, PARAM, 1)
C                                 Select absolute error control
      PARAM(10) = 1.0
C                                 Print header
      WRITE (NOUT,99998)
      IDO = 1
      ISTEP = 0
   10 CONTINUE
      ISTEP = ISTEP + 24
      TEND = ISTEP
      CALL IVPRK (IDO, N, FCN, T, TEND, TOL, PARAM, Y)
      IF (ISTEP .LE. 240) THEN
         WRITE (NOUT,'(I6,3F12.3)') ISTEP/24, T, Y
C                                 Final call to release workspace
         IF (ISTEP .EQ. 240) IDO = 3
         GO TO 10
      END IF
C                                 Show number of function calls.
      WRITE (NOUT,99999) PARAM(35)
99998 FORMAT (4X, 'ISTEP', 5X, 'Time', 9X, 'Y1', 11X, 'Y2')
```

```
99999 FORMAT (4X, 'Number of fcn calls with IVPRK =', F6.0)
      END
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), YPRIME(N)
C                                  SPECIFICATIONS FOR DATA VARIABLES
      REAL       AK1, AK2, AK3
C
      DATA AK1, AK2, AK3/294.0E0, 3.0E0, 0.01020408E0/
C
      YPRIME(1) = -Y(1) - Y(1)*Y(2) + AK1*Y(2)
      YPRIME(2) = -AK2*Y(2) + AK3*(1.0E0-Y(2))*Y(1)
      RETURN
      END
```

Output

```
ISTEP     Time         Y1           Y2
 1      24.000       0.688       0.002
 2      48.000       0.634       0.002
 3      72.000       0.589       0.002
 4      96.000       0.549       0.002
 5     120.000       0.514       0.002
 6     144.000       0.484       0.002
 7     168.000       0.457       0.002
 8     192.000       0.433       0.001
 9     216.000       0.411       0.001
10     240.000       0.391       0.001
Number of fcn calls with IVPRK = 2153.
```

IVMRK/DIVMRK (Single/Double precision)

Solve an initial-value problem $y' = f(t, y)$ for ordinary differential equations using Runge-Kutta pairs of various orders.

Usage

CALL IVMRK (IDO, N, FCN, T, TEND, Y, YPRIME)

Arguments

IDO — Flag indicating the state of the computation. (Input/Output)

IDO	**State**
1	Initial entry
2	Normal re-entry
3	Final call to release workspace
4	Return after a step
5	Return for function evaluation (reverse communication)

Normally, the initial call is made with IDO = 1. The routine then sets IDO = 2, and this value is used for all but the last call that is made with IDO = 3. This final call is used to release workspace, which was automatically allocated by the initial call with IDO = 1.

N — Number of differential equations.   (Input)

FCN — User-supplied `SUBROUTINE` to evaluate functions. The usage is
`CALL FCN (N, T, Y, YPRIME)`, where
`N` — Number of equations.   (Input)
`T` — Independent variable.   (Input)
`Y` — Array of size `N` containing the dependent variable values, *y*.   (Input)
`YPRIME` — Array of size `N` containing the values of the vector *y*′ evaluated at (*t*, *y*).   (Output)
`FCN` must be declared `EXTERNAL` in the calling program.

T — Independent variable.   (Input/Output)
On input, `T` contains the initial value. On output, `T` is replaced by `TEND` unless error conditions have occurred.

TEND — Value of *t* where the solution is required.   (Input)
The value of `TEND` may be less than the initial value of *t*.

Y — Array of size `N` of dependent variables.   (Input/Output)
On input, `Y` contains the initial values. On output, `Y` contains the approximate solution.

YPRIME — Array of size `N` containing the values of the vector *y'* evaluated at (*t*, *y*).   (Output)

Comments

1. Automatic workspace is

   `IVMRK`   42`N` + 50 units, or
   `DIVMRK` 84`N` + 100 units.

   Workspace may be explicitly provided, if desired, by use of `I2MRK`/`DI2MRK`. The reference is

   ```
   CALL I2MRK (IDO, N, FCN, T, TEND, TOL, THRES, PARAM,
               Y, YPRIME, TOL, THRES, PARAM, YMAX,
               RMSERR, WORK)
   ```

   The additional arguments are as follows:

   TOL — Tolerance for error control.   (Input)

   THRES — Array of size `N`.   (Input)
   `THRES(I)` is a threshold for solution component `Y(I)`. It is chosen so that the value of `Y(L)` is not important when `Y(L)` is smaller in magnitude than `THRES(L)`. `THRES(L)` must be greater than or equal to `sqrt(amach(4))`.

   PARAM — A floating-point array of size 50 containing optional parameters.   (Input/Output)
   If a parameter is zero, then a default value is used. These default values are given below. The following parameters must be set by the user:

`PARAM`	**Meaning**
1 `HINIT`	Initial value of the step size. Must be chosen such that 0.01 ≥ `HINIT` ≥ 10.0 `amach`(4). Default: automatic selection of stepsize.
2 `METHOD`	Specify which Runge-Kutta pair is to be used. 1 - use the (2, 3) pair 2 - use the (4, 5) pair 3 - use the (7, 8) pair. Default: `METHOD` = 1 if $10^{-2} \ge$ `tol` $> 10^{-4}$ `METHOD` = 2 if $10^{-4} \ge$ `tol` $> 10^{-6}$ `METHOD` = 3 if $10^{-6} \ge$ `tol`
3 `ERREST`	`ERREST` = 1 attempts to assess the true error, the difference between the numerical solution and the true solution. The cost of this is roughly twice the cost of the integration itself with `METHOD` = 2 or `METHOD` = 3, and three times with `METHOD` = 1. Default: `ERREST` = 0.
4 `INTRP`	If nonzero, then return the `IDO` = 4 before each step. See Comment 3. Default: 0
5 `RCSTAT`	If nonzero, then reverse communication is used to get derivative information. See Comment 4. Default: 0.
6 - 30	Not used

The following entries are set by the program:

31 `HTRIAL`	Current trial step size.
32 `NSTEP`	Number of steps taken.
33 `NFCN`	Number of function evaluations.
34 `ERRMAX`	The maximum approximate weighted true error taken over all solution components and all steps from `T` through the current integration point.
35 `TERRMX`	First value of the independent variable where an approximate true error attains the maximum value `ERRMAX`.

YMAX Array of size `N`, where `YMAX(L)` is the largest value of `ABS(Y(L))` computed at any step in the integration so far.

RMSERR — Array of size `N` where `RMSERR(L)` approximates the `RMS` average of the true error of the numerical solution for the `L`-th solution component, `L` = 1,..., *N*. The average is taken over all steps from `T` through the current integration point. `RMSERR` is accessed and set only if `PARAM(3)` = 1.

WORK — Floating point work array of size 39`N` using the working precision. The contents of `WORK` must not be changed from the first call with `IDO` = 1 until after the final call with `IDO` = 3.

2. Informational errors

Type	Code	
4	1	It does not appear possible to achieve the accuracy specified by TOL and THRES(*) using the current precision and METHOD. A larger value for METHOD, if possible, will permit greater accuracy with this precision. The integration must be restarted.
4	2	The global error assessment may not be reliable beyond the current integration point T. This may occur because either too little or too much accuracy has been requested or because *f*(*t*, *y*) is not smooth enough for values of *t* just past TEND and current values of the solution *y*. This return does not mean that you cannot integrate past TEND, rather that you cannot do it with PARAM(3) = 1.

3 If PARAM(4) is nonzero, the subroutine returns with IDO = 4 and will resume calculation at the point of interruption if re-entered with IDO = 4. Some parameters the user might want to examine are IDO, HTRIAL, NSTEP, NFCN, T, and Y. The array Y contains the newly computed trial value for *y*(*t*), accepted or not.

4 If PARAM(5) is nonzero, the subroutine will return with IDO = 5. At this time, evaluate the derivatives at T, place the result in YPRIME, and call IVMRK again. The dummy function I40RK/DI40RK may be used in place of FCN.

Algorithm

Routine IVMRK finds an approximation to the solution of a system of first-order differential equations of the form $y' = f(t, y)$ with given initial data. Relative local error is controlled according to a user-supplied tolerance. For added efficiency, three Runge-Kutta formula pairs, of orders 3, 5, and 8, are available.

Optionally, the values of the vector y' can be passed to IVMRK by reverse communication, avoiding the user-supplied subroutine FCN. Reverse communication is especially useful in applications that have complicated algorithmic requirement for the evaluations of *f*(*t*, *y*). Another option allows assessment of the global error in the integration.

The routine IVMRK is based on the codes contained in RKSUITE, developed by R. W. Brankin, I. Gladwell, and L. F. Shampine (1991).

Example 1

This example integrates the small system (A.2.B2) from the test set of Enright and Pryce (1987):

$$
\begin{aligned}
y_1' &= -y_1 + y_2 \\
y_2' &= y_1 - 2y_2 + y_3 \\
y_3' &= y_2 - y_3 \\
y_1(0) &= 2 \\
y_2(0) &= 0 \\
y_3(0) &= 1
\end{aligned}
$$

```
      INTEGER    N
      PARAMETER  (N=3)
C                                  Specifications for local variables
      INTEGER    IDO
      REAL       T, TEND, Y(N), YPRIME(N)
      EXTERNAL FCN
C                                  Set initial conditions
      T = 0.0
      TEND = 20.0
      Y(1) = 2.0
      Y(2) = 0.0
      Y(3) = 1.0
      IDO = 1
      CALL IVMRK (IDO, N, FCN, T, TEND, Y, YPRIME)
C
C                                  Final call to release workspace
      IDO = 3
      CALL IVMRK (IDO, N, FCN, T, TEND, Y, YPRIME)
C
      CALL WRRRN (’Y’, N, 1, Y, N, 0)
      END
C
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                  Specifications for arguments
      INTEGER    N
      REAL       T, Y(*), YPRIME(*)
C
      YPRIME(1) = -Y(1) + Y(2)
      YPRIME(2) = Y(1) - 2.0*Y(2) + Y(3)
      YPRIME(3) = Y(2) - Y(3)
      RETURN
      END
```

Output

```
        Y
1   1.000
2   1.000
3   1.000
```

Example 2

This problem is the same mildly stiff problem (A.1.F2) from the test set of Enright and Pryce as Example 2 for `IVPRK`, page 645.

$$
\begin{aligned}
y_1' &= -y_1 - y_1 y_2 + k_1 y_2 \\
y_2' &= -k_2 y_2 + k_3 \left(1 - y_2\right) y_1 \\
y_1(0) &= 1 \\
y_2(0) &= 0 \\
k_1 &= 294 \\
k_2 &= 3 \\
k_3 &= 0.01020408 \\
tend &= 240
\end{aligned}
$$

Although not a stiff solver, one notes the greater efficiency of IVMRK over IVPRK, in terms of derivative evaluations. Reverse communication is also used in this example. Users will find this feature particularly helpful if their derivative evaluation scheme is difficult to isolate in a separate subroutine.

```
      INTEGER    N
      PARAMETER  (N=2)
c                                  Specifications for local variables
      INTEGER    IDO, ISTEP, LWORK, NOUT
      REAL       PARAM(50), RMSERR(N), T, TEND, THRES(N), TOL,
     &           WORK(1000), Y(N), YMAX(N), YPRIME(N)
      REAL       AK1, AK2, AK3
      SAVE       AK1, AK2, AK3
c                                  Specifications for intrinsics
      INTRINSIC  SQRT
      REAL       SQRT
c                                  Specifications for subroutines
      EXTERNAL   I2MRK, I40RK, SSET
c                                  Specifications for functions
      EXTERNAL   AMACH
      REAL       AMACH
c
      DATA AK1, AK2, AK3/294.0, 3.0, 0.01020408/
c
      CALL UMACH (2, NOUT)
c                                  Set initial conditions
      T    = 0.0
      Y(1) = 1.0
      Y(2) = 0.0
c                                  Set tolerance for error control,
c                                  threshold vector and parameter
c                                  vector
      TOL = .001
      CALL SSET (N, SQRT(AMACH(4)), THRES, 1)
      CALL SSET (N, 0.0, PARAM, 1)
      LWORK = 1000
c                                  Turn on derivative evaluation by
c                                  reverse communication
      PARAM(5) = 1
      IDO      = 1
      ISTEP    = 24
c                                  Print header
      WRITE (NOUT,99998)
   10 CONTINUE
      TEND = ISTEP
      CALL I2MRK (IDO, N, I40RK, T, TEND, Y, YPRIME, TOL, THRES, PARAM,
     &            YMAX, RMSERR, WORK, LWORK)
      IF (IDO .EQ. 5) THEN
C                                  Evaluate derivatives
C
         YPRIME(1) = -Y(1) - Y(1)*Y(2) + AK1*Y(2)
         YPRIME(2) = -AK2*Y(2) + AK3*(1.0-Y(2))*Y(1)
         GO TO 10
      ELSE IF (ISTEP .LE. 240) THEN
C
C                                  Integrate to 10 equally spaced points
C
         WRITE (NOUT,'(I6,3F12.3)') ISTEP/24, T, Y
         IF (ISTEP .EQ. 240) IDO = 3
         ISTEP = ISTEP + 24
         GO TO 10
      END IF
C                                  Show number of derivative evaluations
C
      WRITE (NOUT,99999) PARAM(33)
99998 FORMAT (3X, 'ISTEP', 5X, 'TIME', 9X, 'Y1', 10X, 'Y2')
99999 FORMAT (/, 4X, 'NUMBER OF DERIVATIVE EVALUATIONS WITH IVMRK =',

     &       F6.0)
      END
```

Output

```
ISTEP     TIME          Y1          Y2
1       24.000       0.688       0.002
2       48.000       0.634       0.002
3       72.000       0.589       0.002
4       96.000       0.549       0.002
5      120.000       0.514       0.002
6      144.000       0.484       0.002
7      168.000       0.457       0.002
8      192.000       0.433       0.001
9      216.000       0.411       0.001
10     240.000       0.391       0.001
NUMBER OF DERIVATIVE EVALUATIONS WITH IVMRK = 1375.
```

Example 3

This example demonstrates how exceptions may be handled. The problem is from Enright and Pryce (A.2.F1), and has discontinuities. We choose this problem to force a failure in the global error estimation scheme, which requires some smoothness in *y*. We also request an initial relative error tolerance which happens to be unsuitably small in this precision.

If the integration fails because of problems in global error assessment, the assessment option is turned off, and the integration is restarted. If the integration fails because the requested accuracy is not achievable, the tolerance is increased, and global error assessment is requested. The reason error assessment is turned on is that prior assessment failures may have been due more in part to an overly stringent tolerance than lack of smoothness in the derivatives.

When the integration is successful, the example prints the final relative error tolerance, and indicates whether or not global error estimation was possible.

$$y_1' = y_2$$

$$y_2' = \begin{cases} 2ay_2 - \left(\pi^2 + a^2\right)y_1 + 1, \lfloor x \rfloor \text{even} \\ 2ay_2 - \left(\pi^2 + a^2\right)y_1 - 1, \lfloor x \rfloor \text{odd} \end{cases}$$

$$y_1(0) = 0$$

$$y_2(0) = 0$$

$$a = 0.1$$

$$\lfloor x \rfloor = \text{largest integer } \le x$$

```
      INTEGER    N
      PARAMETER  (N=2)
c                                  Specifications for local variables
      INTEGER    IDO, LWORK, NOUT
      REAL       PARAM(50), RMSERR(N), T, TEND, THRES(N), TOL,
     &           WORK(100), Y(N), YMAX(N), YPRIME(N)
c
c                                  Specifications for intrinsics
      INTRINSIC  SQRT
      REAL       SQRT
c                                  Specifications for subroutines
c
      EXTERNAL   ERSET, I2MRK, SSET, UMACH, WRRRN
c
c                                  Specifications for functions
      EXTERNAL   AMACH, FCN, IERCD
      INTEGER    IERCD
      REAL       AMACH
c
c
      CALL UMACH (2, NOUT)
c                                  Turn off stopping for FATAL errors
      CALL ERSET (4, -1, 0)
c                                  Initialize input, turn on global
c                                  error assessment
      LWORK = 100
      TOL   = SQRT(AMACH(4))
      CALL SSET (50, 0.0E0, PARAM, 1)
      CALL SSET (N, SQRT(AMACH(4)), THRES, 1)
      TEND     = 20.0E0
      PARAM(3) = 1
c
   10 CONTINUE
c                                  Set initial values
      T    = 0.0E0
      Y(1) = 0.0E0
      Y(2) = 0.0E0
      IDO  = 1
      CALL I2MRK (IDO, N, FCN, T, TEND, Y, YPRIME, TOL, THRES, PARAM,
     &            YMAX, RMSERR, WORK, LWORK)
      IF (IERCD() .EQ. 32) THEN
c                                  Unable to achieve requested
```

```
c                                  accuracy, so increase tolerance.
c                                  Activate global error assessment
         TOL      = 10.0*TOL
         PARAM(3) = 1
         WRITE (NOUT,99995) TOL
         GO TO 10
      ELSE IF (IERCD() .EQ. 34) THEN
c                                  Global error assessment has failed,
c                                  cannot continue from this point,
c                                  so restart integration
         WRITE (NOUT,99996)
         PARAM(3) = 0
         GO TO 10
      END IF
c
c                                  Final call to release workspace
      IDO = 3
      CALL I2MRK (IDO, N, FCN, T, TEND, Y, YPRIME, TOL, THRES, PARAM,
     &            YMAX, RMSERR, WORK, LWORK)
c
c                                  Summarize status
      WRITE (NOUT,99997) TOL
      IF (PARAM(3) .EQ. 1) THEN
         WRITE (NOUT,99998)
      ELSE
         WRITE (NOUT,99999)
      END IF
      CALL WRRRN (’Y’, N, 1, Y, N, 0)
c
99995 FORMAT (/, ’CHANGING TOLERANCE TO ’, E9.3, ’ AND RESTARTING ...’
     &       , /, ’ALSO (RE)ENABLING GLOBAL ERROR ASSESSMENT’, /)
99996 FORMAT (/, ’DISABLING GLOBAL ERROR ASSESSMENT AND RESTARTING ...’
     &       , /)
99997 FORMAT (/, 72(’-’), //, ’SOLUTION OBTAINED WITH TOLERANCE = ’,
     &       E9.3)
99998 FORMAT (’GLOBAL ERROR ASSESSMENT IS AVAILABLE’)
99999 FORMAT (’GLOBAL ERROR ASSESSMENT IS NOT AVAILABLE’)
c
      END
c
      SUBROUTINE FCN (N, T, Y, YPRIME)
c                                  Specifications for arguments
      INTEGER    N
      REAL       T, Y(*), YPRIME(*)
c                                  Specifications for local variables
      REAL       A
      REAL       PI
      LOGICAL    FIRST
      SAVE       FIRST, PI
c                                  Specifications for intrinsics
      INTRINSIC  INT, MOD
      INTEGER    INT, MOD
c                                  Specifications for functions
      EXTERNAL   CONST
      REAL       CONST
c
      DATA FIRST/.TRUE./
c
      IF (FIRST) THEN
```

```
         PI    = CONST('PI')
         FIRST = .FALSE.
      END IF
C
      A         = 0.1E0
      YPRIME(1) = Y(2)
      IF (MOD(INT(T),2) .EQ. 0) THEN
         YPRIME(2) = 2.0E0*A*Y(2) - (PI*PI+A*A)*Y(1) + 1.0E0
      ELSE
         YPRIME(2) = 2.0E0*A*Y(2) - (PI*PI+A*A)*Y(1) - 1.0E0
      END IF
      RETURN
      END
```

Output

```
 *** FATAL    ERROR 34 from i2mrk.  The global error assessment may not
 ***          be reliable for T past 9.994749E-01.  The integration is
 ***          being terminated.


DISABLING GLOBAL ERROR ASSESSMENT AND RESTARTING ...


 *** FATAL    ERROR 32 from i2mrk.  In order to satisfy the error
 ***          requirement I6MRK would have to use a step size of
 ***          3.647129E- 06 at TNOW = 9.999932E-01.  This is too small
 ***          for the current precision.


CHANGING TOLERANCE TO 0.345E-02 AND RESTARTING ...
ALSO (RE)ENABLING GLOBAL ERROR ASSESSMENT


 *** FATAL    ERROR 34 from i2mrk.  The global error assessment may
 ***          not be reliable for T past 9.986024E-01.  The integration
 ***          is being terminated.

DISABLING GLOBAL ERROR ASSESSMENT AND RESTARTING ...


------------------------------------------------------------------------

SOLUTION OBTAINED WITH TOLERANCE = 0.345E-02
GLOBAL ERROR ASSESSMENT IS NOT AVAILABLE

     Y
 1  -12.30
 2    0.95
```

IVPAG/DIVPAG (Single/Double precision)

Solve an initial-value problem for ordinary differential equations using either Adams-Moulton’s or Gear’s BDF method.

Usage

```
CALL IVPAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL, PARAM, Y)
```

Arguments

IDO — Flag indicating the state of the computation. (Input/Output)

`IDO`	**State**
1	Initial entry
2	Normal re-entry
3	Final call to release workspace
4	Return because of interrupt 1
5	Return because of interrupt 2 with step accepted
6	Return because of interrupt 2 with step rejected
7	Return for new value of matrix `A`.

Normally, the initial call is made with `IDO` = 1. The routine then sets `IDO` = 2, and this value is then used for all but the last call that is made with `IDO` = 3. This final call is only used to release workspace, which was automatically allocated by the initial call with `IDO` = 1. See Comment 5 for a description of the interrupts.

When `IDO` = 7, the matrix A at t must be recomputed and `IVPAG/DIVPAG` called again. No other argument (including `IDO`) should be changed. This value of `IDO` is returned only if `PARAM`(19) = 2.

N — Number of differential equations. (Input)

FCN — User-supplied `SUBROUTINE` to evaluate functions. The usage is
`CALL FCN (N, T, Y, YPRIME)`, where

`N` – Number of equations. (Input)
`T` – Independent variable, t. (Input)
`Y` – Array of size `N` containing the dependent variable values, y. (Input)
`YPRIME` – Array of size `N` containing the values of the vector y' evaluated at (t, y). (Output)
See Comment 3.

`FCN` must be declared `EXTERNAL` in the calling program.

FCNJ — User-supplied `SUBROUTINE` to compute the Jacobian. The usage is
`CALL FCNJ (N, T, Y, DYPDY)` where

`N` – Number of equations. (Input)
`T` – Independent variable, t. (Input)
`Y` – Array of size `N` containing the dependent variable values, $y(t)$. (Input)

DYPDY – An array, with data structure and type determined by PARAM(14) = MTYPE, containing the required partial derivatives $\partial f_i / \partial y_j$. (Output)
These derivatives are to be evaluated at the current values of (t, y). When the Jacobian is dense, MTYPE = 0 or = 2, the leading dimension of DYPDY has the value N. When the Jacobian matrix is banded, MTYPE = 1, and the leading dimension of DYPDY has the value 2 * NLC + NUC + 1. If the matrix is banded positive definite symmetric, MTYPE = 3, and the leading dimension of DYPDY has the value NUC + 1.

FCNJ must be declared EXTERNAL in the calling program. If PARAM(19) = IATYPE is nonzero, then FCNJ should compute the Jacobian of the righthand side of the equation $Ay' = f(t, y)$. The subroutine FCNJ is used only if PARAM(13) = MITER = 1.

A — Matrix structure used when the system is implicit. (Input)
The matrix A is referenced only if PARAM(19) = IATYPE is nonzero. Its data structure is determined by PARAM(14) = MTYPE. The matrix A must be nonsingular and MITER must be 1 or 2. See Comment 3.

T — Independent variable, t. (Input/Output)
On input, T contains the initial independent variable value. On output, T is replaced by TEND unless error or other normal conditions arise. See IDO for details.

TEND — Value of $t = tend$ where the solution is required. (Input)
The value *tend* may be less than the initial value of t.

TOL — Tolerance for error control. (Input)
An attempt is made to control the norm of the local error such that the global error is proportional to TOL.

PARAM — A *floating-point* array of size 50 containing optional parameters. (Input/Output)
If a parameter is zero, then the default value is used. These default values are given below. Parameters that concern values of the step size are applied in the direction of integration. The following parameters must be set by the user:

	PARAM	**Meaning**
1	HINIT	Initial value of the step size H. Always nonnegative. Default: $0.001\|tend - t_0\|$.
2	HMIN	Minimum value of the step size H. Default: 0.0.
3	HMAX	Maximum value of the step size H. Default: No limit, beyond the machine scale, is imposed on the step size.
4	MXSTEP	Maximum number of steps allowed. Default: 500.
5	MXFCN	Maximum number of function evaluations allowed. Default: No enforced limit.

	PARAM	Meaning
6	MAXORD	Maximum order of the method. Default: If Adams-Moulton method is used, then 12. If Gear's or BDF method is used, then 5. The defaults are the maximum values allowed.
7	INTRP1	If this value is set nonzero, the subroutine will return before every step with IDO = 4. See Comment 5. Default: 0.
8	INTRP2	If this value is nonzero, the subroutine will return after every successful step with IDO = 5 and return with IDO = 6 after every unsuccessful step. See Comment 5. Default: 0
9	SCALE	A measure of the scale of the problem, such as an approximation to the average value of a norm of the Jacobian along the solution. Default: 1.0
10	INORM	Switch determining error norm. In the following, e_i is the absolute value of an estimate of the error in $y_i(t)$. Default: 0. 0 — min(absolute error, relative error) = max(e_i/w_i); $i = 1, \ldots, N$, where $w_i = \max(\lvert y_i(t)\rvert, 1.0)$. 1 — absolute error = max(e_i), $i = 1 \ldots, N$. 2 — max(e_i / w_i), $i = 1 \ldots, N$ where $w_i = \max(\lvert y_i(t)\rvert$, FLOOR), and FLOOR is the value PARAM(11). 3 — Scaled Euclidean norm defined as $$\text{YMAX} = \sqrt{\sum\nolimits_{i=1}^{N} e_i^2 / w_i^2}$$ where $w_i = \max(\lvert y_i(t)\rvert, 1.0)$. Other definitions of YMAX can be specified by the user, as explained in Comment 1.
11	FLOOR	Used in the norm computation associated the parameter INORM. Default: 1.0.
12	METH	Integration method indicator. 1 = METH selects the Adams-Moulton method. 2 = METH selects Gear's BDF method. Default: 1.

	PARAM	**Meaning**
13	`MITER`	Nonlinear solver method indicator. *Note:* If the problem is stiff and a chord or modified Newton method is most efficient, use `MITER` = 1 or = 2. 0 = `MITER` selects functional iteration. The value `IATYPE` must be set to zero with this option. 1 = `MITER` selects a chord method with a user-provided Jacobian. 2 = `MITER` selects a chord method with a divided-difference Jacobian. 3 = `MITER` selects a chord method with the Jacobian replaced by a diagonal matrix based on a directional derivative. The value `IATYPE` must be set to zero with this option. Default: 0.
14	`MTYPE`	Matrix type for *A* (if used) and the Jacobian (if `MITER` = 1 or = 2). When both are used, *A* and the Jacobian must be of the same type. 0 = `MTYPE` selects full matrices. 1 = `MTYPE` selects banded matrices. 2 = `MTYPE` selects symmetric positive definite matrices. 3 = `MTYPE` selects banded symmetric positive definite matrices. Default: 0.
15	`NLC`	Number of lower codiagonals, used if `MTYPE` = 1. Default: 0.
16	`NUC`	Number of upper codiagonals, used if `MTYPE` = 1 or `MTYPE` = 3. Default: 0.
17		Not used.
18	`EPSJ`	Relative tolerance used in computing divided difference Jacobians. Default: `SQRT`(`AMACH`(4)) .
19	`IATYPE`	Type of the matrix *A*. 0 = `IATYPE` implies *A* is not used (the system is explicit). 1 = `IATYPE` if *A* is a constant matrix. 2 = `IATYPE` if *A* depends on *t*. Default: 0.

	PARAM	**Meaning**
20	LDA	Leading dimension of array *A* exactly as specified in the dimension statement in the calling program. Used if IATYPE is not zero. Default: N if MTYPE = 0 or = 2 NUC + NLC + 1 if MTYPE = 1 NUC + 1 if MTYPE = 3
21–30		Not used.

The following entries in the array PARAM are set by the program:

	PARAM	**Meaning**
31	HTRIAL	Current trial step size.
32	HMINC	Computed minimum step size.
33	HMAXC	Computed maximum step size.
34	NSTEP	Number of steps taken.
35	NFCN	Number of function evaluations used.
36	NJE	Number of Jacobian evaluations.
37–50		Not used.

Y — Array of size N of dependent variables, $y(t)$. (Input/Output)
On input, Y contains the initial values, $y(t_0)$. On output, Y contains the approximate solution, $y(t)$.

Comments

1. Automatic workspace usage is

   IVPAG 4N + NMETH + NPW + NIPVT, or
   DIVPAG 8N + 2 * NMETH + 2 * NPW + NIPVT units.

   Here,

   NMETH = 13N if METH is 1,

   NMETH = 6N if METH is 2.

   NPW = 2N + NPWM + NPWA

   where

   NPWM = 0 if MITER is 0 or 3,

   NPWM = N^2 if MITER is 1 or 2, and if MTYPE is 0 or 2.

   NPWM = N(2 * NLC + NUC + 1) if MITER is 1 or 2 and MTYPE = 1.

   NPWM = N(NLC + 1) if MITER is 1 or 2 and if MTYPE = 3.

   NPWA = 0 if IATYPE is 0.

NPWA = N^2 if IATYPE is nonzero and MTYPE = 0,

NPWA = N(2 * NLC + NUC + 1) if IATYPE is nonzero and MTYPE = 1

NIPVT = N if MITER is 1 or 2 and MTYPE is 0 or 1,

NIPVT = 1, otherwise.

Workspace and a user-supplied error norm subroutine may be explicitly provided, if desired, by use of I2PAG/DI2PAG. The reference is

```
CALL I2PAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL,
            PARAM, Y, YTEMP, YMAX, ERROR, SAVE1,
            SAVE2, PW, IPVT, VNORM)
```

None of the additional array arguments should be changed from the first call with IDO = 1 until after the final call with IDO = 3. The additional arguments are as follows:

YTEMP — Array of size NMETH. (Workspace)

YMAX — Array of size N containing the maximum Y-values computed so far. (Output)

ERROR — Array of size N containing error estimates for each component of Y. (Output)

SAVE1 — Array of size N. (Workspace)

SAVE2 — Array of size N. (Workspace)

PW — Array of size NPW. PW is used both to store the Jacobian and as workspace. (Workspace)

IPVT — Array of size N. (Workspace)

VNORM — A Fortran SUBROUTINE to compute the norm of the error. (Input)
The routine may be provided by the user, or the IMSL routine I3PRK/DI3PRK may be used. In either case, the name must be declared in a Fortran ENTERNAL statement. If usage of the IMSL routine is intended, then the name I3PRK/DI3PRK should be specified. The usage of the error norm routine is
CALL VNORM (N, V, Y, YMAX, ENORM) where

Arg.	**Definition**
N	Number of equations. (Input)
V	Array of size N containing the vector whose norm is to be computed. (Input)
Y	Array of size N containing the values of the dependent variable. (Input)
YMAX	Array of size N containing the maximum values of $\|y(t)\|$. (Input)
ENORM	Norm of the vector V. (Output)

VNORM must be declared EXTERNAL in the calling program.

2. Informational errors

Type	Code	
4	1	After some initial success, the integration was halted by repeated error-test failures.
4	2	The maximum number of function evaluations have been used.
4	3	The maximum number of steps allowed have been used. The problem may be stiff.
4	4	On the next step `T` + `H` will equal `T`. Either `TOL` is too small, or the problem is stiff. Note: If the Adams-Moulton method is the one used in the integration, then users can switch to the BDF methods. If the BDF methods are being used, then these comments are gratuitous and indicate that the problem is too stiff for this combination of method and value of `TOL`.
4	5	After some initial success, the integration was halted by a test on `TOL`.
4	6	Integration was halted after failing to pass the error test even after dividing the initial step size by a factor of 1.0E + 10. The value `TOL` may be too small.
4	7	Integration was halted after failing to achieve corrector convergence even after dividing the initial step size by a factor of `1.0E` + 10. The value `TOL` may be too small.
4	8	`IATYPE` is nonzero and the input matrix A multiplying y' is singular.

3. Both explicit systems, of the form $y' = f(t, y)$, and implicit systems, $Ay' = f(t, y)$, can be solved. If the system is explicit, then `PARAM`(19) = 0; and the matrix A is not referenced. If the system is implicit, then `PARAM`(14) determines the data structure of the array `A`. If `PARAM`(19) = 1, then `A` is assumed to be a constant matrix. The value of `A` used on the first call (with `IDO` = 1) is saved until after a call with `IDO` = 3. The value of `A` must not be changed between these calls. If `PARAM`(19) = 2, then the matrix is assumed to be a function of t.

4. If `MTYPE` is greater than zero, then `MITER` must equal 1 or 2.

5. If `PARAM`(7) is nonzero, the subroutine returns with `IDO`= 4 and will resume calculation at the point of interruption if re-entered with `IDO` = 4. If `PARAM`(8) is nonzero, the subroutine will interrupt immediately after decides to accept the result of the most recent trial step. The value `IDO` = 5 is returned if the routine plans to accept, or `IDO` = 6 if it plans to reject. The value `IDO` may be changed by the user (by changing `IDO` from 6 to 5) to force acceptance of a step that would otherwise be rejected. Relevant parameters to observe after return from

an interrupt are IDO, HTRIAL, NSTEP, NFCN, NJE, T and Y. The array Y contains the newly computed trial value $y(t)$.

Algorithm

The routine IVPAG solves a system of first-order ordinary differential equations of the form $y' = f(t, y)$ or $Ay' = f(t, y)$ with initial conditions where A is a square nonsingular matrix of order N. Two classes of implicit linear multistep methods are available. The first is the implicit Adams-Moulton method (up to order twelve); the second uses the backward differentiation formulas BDF (up to order five). The BDF method is often called Gear's stiff method. In both cases, because basic formulas are implicit, a system of nonlinear equations must be solved at each step. The deriviative matrix in this system has the form $L = A + \eta J$ where η is a small number computed by IVPAG and J is the Jacobian. When it is used, this matrix is computed in the user-supplied routine FCNJ or else it is approximated by divided differences as a default. Using defaults, A is the identity matrix. The data structure for the matrix L may be identified to be real general, real banded, symmetric positive definite, or banded symmetric positive definite. The default structure for L is real general.

Example 1

Euler's equation for the motion of a rigid body not subject to external forces is

$$\begin{aligned} y_1' &= y_2 y_3 & y_1(0) &= 0 \\ y_2' &= -y_1 y_3 & y_2(0) &= 1 \\ y_3' &= -0.51 y_1 y_2 & y_3(0) &= 1 \end{aligned}$$

Its solution is, in terms of Jacobi elliptic functions, $y_1(t) = \mathrm{sn}(t; k)$, $y_2(t) = \mathrm{cn}(t; k)$, $y_3(t) = \mathrm{dn}(t; k)$ where $k^2 = 0.51$. The Adams-Moulton method of IVPAG is used to solve this system, since this is the default. All parameters are set to defaults.

The last call to IVPAG with IDO = 3 releases IMSL workspace that was reserved on the first call to IVPAG. It is not necessary to release the workspace in this example because the program ends after solving a single problem. The call to release workspace is made as a model of what would be needed if the program included further calls to IMSL routines.

Because PARAM(13) = MITER = 0, functional iteration is used and so subroutine FCNJ is never called. It is included only because the calling sequence for IVPAG requires it.

```
      INTEGER    N, NPARAM
      PARAMETER  (N=3, NPARAM=50)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, IEND, NOUT
      REAL       A(1,1), PARAM(NPARAM), T, TEND, TOL, Y(N)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPAG, SSET, UMACH
```

```
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN, FCNJ
C                                 Initialize
      CALL SSET (NPARAM, 0.0, PARAM, 1)
C
      IDO  = 1
      T    = 0.0
      Y(1) = 0.0
      Y(2) = 1.0
      Y(3) = 1.0
      TOL  = 1.0E-6
C                                 Write title
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
C                                 Integrate ODE
      IEND = 0
   10 CONTINUE
      IEND = IEND + 1
      TEND = IEND
C                                 The array a(*,*) is not used.
      CALL IVPAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL, PARAM, Y)
      IF (IEND .LE. 10) THEN
         WRITE (NOUT,99999) T, Y
C                                 Finish up
         IF (IEND .EQ. 10) IDO = 3
         GO TO 10
      END IF
99998 FORMAT (11X, 'T', 14X, 'Y(1)', 11X, 'Y(2)', 11X, 'Y(3)')
99999 FORMAT (4F15.5)
      END
C
      SUBROUTINE FCN (N, X, Y, YPRIME)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       X, Y(N), YPRIME(N)
C
      YPRIME(1) = Y(2)*Y(3)
      YPRIME(2) = -Y(1)*Y(3)
      YPRIME(3) = -0.51*Y(1)*Y(2)
      RETURN
      END
C
      SUBROUTINE FCNJ (N, X, Y, DYPDY)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       X, Y(N), DYPDY(N,*)
C                                 This subroutine is never called
      RETURN
      END
```

Output

```
     T              Y(1)           Y(2)           Y(3)
 1.00000        0.80220        0.59705        0.81963
 2.00000        0.99537       -0.09615        0.70336
 3.00000        0.64141       -0.76720        0.88892
 4.00000       -0.26961       -0.96296        0.98129
 5.00000       -0.91173       -0.41079        0.75899
 6.00000       -0.95751        0.28841        0.72967
 7.00000       -0.42877        0.90342        0.95197
 8.00000        0.51092        0.85963        0.93106
 9.00000        0.97567        0.21926        0.71730
10.00000        0.87790       -0.47884        0.77906
```

Example 2

The BDF method of IVPAG is used to solve Example 2 of IVPRK, page 645. We set PARAM(12) = 2 to designate the BDF method. A chord or modified Newton method, with the Jacobian computed by divided differences, is used to solve the nonlinear equations. Thus, we set PARAM(13) = 2. The number of evaluations of y' is printed after the last output point, showing the efficiency gained when using a stiff solver compared to using IVPRK on this problem. The number of evaluations may vary, depending on the accuracy and other arithmetic characteristics of the computer.

```
      INTEGER    MXPARM, N
      PARAMETER  (MXPARM=50, N=2)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    MABSE, MBDF, MSOLVE
      PARAMETER  (MABSE=1, MBDF=2, MSOLVE=2)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, ISTEP, NOUT
      REAL       A(1,1), PARAM(MXPARM), T, TEND, TOL, Y(N)
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPAG, SSET, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN, FCNJ
C
      CALL UMACH (2, NOUT)
C                                 Set initial conditions
      T = 0.0
      Y(1) = 1.0
      Y(2) = 0.0
C                                 Set error tolerance
      TOL = 0.001
C                                 Set PARAM to defaults
      CALL SSET (MXPARM, 0.0, PARAM, 1)
C                                 Select absolute error control
      PARAM(10) = MABSE
C                                 Select BDF method
      PARAM(12) = MBDF
C                                 Select chord method and
C                                 a divided difference Jacobian.
      PARAM(13) = MSOLVE
C                                 Print header
      WRITE (NOUT,99998)
      IDO = 1
      ISTEP = 0
   10 CONTINUE
      ISTEP = ISTEP + 24
      TEND = ISTEP
C                                 The array a(*,*) is not used.
      CALL IVPAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL, PARAM, Y)
      IF (ISTEP .LE. 240) THEN
         WRITE (NOUT,’(I6,3F12.3)’) ISTEP/24, T, Y
C                                 Final call to release workspace
```

```
         IF (ISTEP .EQ. 240) IDO = 3
         GO TO 10
      END IF
C                                   Show number of function calls.
      WRITE (NOUT,99999) PARAM(35)
99998 FORMAT (4X, 'ISTEP', 5X, 'Time', 9X, 'Y1', 11X, 'Y2')
99999 FORMAT (4X, 'Number of fcn calls with IVPAG =', F6.0)
      END
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                   SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), YPRIME(N)
C                                   SPECIFICATIONS FOR SAVE VARIABLES
      REAL       AK1, AK2, AK3
      SAVE       AK1, AK2, AK3
C
      DATA AK1, AK2, AK3/294.0E0, 3.0E0, 0.01020408E0/
C
      YPRIME(1) = -Y(1) - Y(1)*Y(2) + AK1*Y(2)
      YPRIME(2) = -AK2*Y(2) + AK3*(1.0E0-Y(2))*Y(1)
      RETURN
      END
      SUBROUTINE FCNJ (N, T, Y, DYPDY)
C                                   SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), DYPDY(N,*)
C
      RETURN
      END
```

Output

```
ISTEP     Time           Y1          Y2
 1      24.000        0.689       0.002
 2      48.000        0.636       0.002
 3      72.000        0.590       0.002
 4      96.000        0.550       0.002
 5     120.000        0.515       0.002
 6     144.000        0.485       0.002
 7     168.000        0.458       0.002
 8     192.000        0.434       0.001
 9     216.000        0.412       0.001
10     240.000        0.392       0.001
Number of fcn calls with IVPAG =   73.
```

Example 3

The BDF method of IVPAG is used to solve the so-called Robertson problem:

$$
\begin{aligned}
y_1' &= -c_1 y_1 + c_2 y_2 y_3 & \qquad y_1(0) &= 1 \\
y_2' &= -y_1' - y_3' & \qquad y_2(0) &= 0 \\
y_3' &= c_3 y_2^2 & \qquad y_3(0) &= 0 \\
c_1 = 0.04, c_2 &= 10^4, c_3 = 3 \times 10^7 & \qquad 0 \le t &\le 10
\end{aligned}
$$

Output is obtained after each unit of the independent variable. A user-provided subroutine for the Jacobian matrix is used. An absolute error tolerance of 10^{-5} is required.

```
      INTEGER    MXPARM, N
      PARAMETER  (MXPARM=50, N=3)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    MABSE, MBDF, MSOLVE
      PARAMETER  (MABSE=1, MBDF=2, MSOLVE=1)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, ISTEP, NOUT
      REAL       A(1,1), PARAM(MXPARM), T, TEND, TOL, Y(N)
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPAG, SSET, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN, FCNJ
C
      CALL UMACH (2, NOUT)
C                                 Set initial conditions
      T = 0.0
      Y(1) = 1.0
      Y(2) = 0.0
      Y(3) = 0.0
C                                 Set error tolerance
      TOL = 1.0E-5
C                                 Set PARAM to defaults
      CALL SSET (MXPARM, 0.0, PARAM, 1)
C                                 Select absolute error control
      PARAM(10) = MABSE
C                                 Select BDF method
      PARAM(12) = MBDF
C                                 Select chord method and
C                                 a user-provided Jacobian.
      PARAM(13) = MSOLVE
C                                 Print header
      WRITE (NOUT,99998)
      IDO = 1
      ISTEP = 0
   10 CONTINUE
      ISTEP = ISTEP + 1
      TEND = ISTEP
C                                 The array a(*,*) is not used.
      CALL IVPAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL, PARAM, Y)
      IF (ISTEP .LE. 10) THEN
         WRITE (NOUT,’(I6,F12.2,3F13.5)’) ISTEP, T, Y
C                                 Final call to release workspace
         IF (ISTEP .EQ. 10) IDO = 3
         GO TO 10
      END IF
99998 FORMAT (4X, ’ISTEP’, 5X, ’Time’, 9X, ’Y1’, 11X, ’Y2’, 11X,
     &       ’Y3’)
      END
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), YPRIME(N)
C                                 SPECIFICATIONS FOR SAVE VARIABLES
      REAL       C1, C2, C3
      SAVE       C1, C2, C3
C
      DATA C1, C2, C3/0.04E0, 1.0E4, 3.0E7/
C
      YPRIME(1) = -C1*Y(1) + C2*Y(2)*Y(3)
      YPRIME(3) = C3*Y(2)**2
      YPRIME(2) = -YPRIME(1) - YPRIME(3)
      RETURN
      END
      SUBROUTINE FCNJ (N, T, Y, DYPDY)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), DYPDY(N,*)
C                                  SPECIFICATIONS FOR SAVE VARIABLES
      REAL       C1, C2, C3
      SAVE       C1, C2, C3
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSET
C
      DATA C1, C2, C3/0.04E0, 1.0E4, 3.0E7/
C                                  Clear array to zero
      CALL SSET (N**2, 0.0, DYPDY, 1)
C                                  Compute partials
      DYPDY(1,1) = -C1
      DYPDY(1,2) = C2*Y(3)
      DYPDY(1,3) = C2*Y(2)
      DYPDY(3,2) = 2.0*C3*Y(2)
      DYPDY(2,1) = -DYPDY(1,1)
      DYPDY(2,2) = -DYPDY(1,2) - DYPDY(3,2)
      DYPDY(2,3) = -DYPDY(1,3)
      RETURN
      END
```

Output

```
 ISTEP     Time         Y1           Y2           Y3
 1         1.00      0.96647      0.00003      0.03350
 2         2.00      0.94164      0.00003      0.05834
 3         3.00      0.92191      0.00002      0.07806
 4         4.00      0.90555      0.00002      0.09443
 5         5.00      0.89153      0.00002      0.10845
 6         6.00      0.87928      0.00002      0.12070
 7         7.00      0.86838      0.00002      0.13160
 8         8.00      0.85855      0.00002      0.14143
 9         9.00      0.84959      0.00002      0.15039
10        10.00      0.84136      0.00002      0.15862
```

Example 4

Solve the partial differential equation

$$e^{-t}\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

with the initial condition

$$u(t = 0, x) = \sin x$$

and the boundary conditions

$$u(t, x = 0) = u(t, x = \pi) = 0$$

on the square [0, 1] × [0, π], using the method of lines with a piecewise-linear Galerkin discretization. The exact solution is $u(t, x) = \exp(1 - e^t) \sin x$. The interval $[0, \pi]$ is divided into equal intervals by choosing breakpoints $x_k = k\pi/(N + 1)$ for $k = 0, \ldots, N + 1$. The unknown function $u(t, x)$ is approximated by

$$\sum\nolimits_{k=1}^{N} c_k(t)\phi_k(x)$$

where $\phi_k(x)$ is the piecewiselinear function that equals 1 at x_k and is zero at all of the other breakpoints. We approximate the partial differential equation by a system of N ordinary differential equations, $A\, dc/dt = Rc$ where A and R are matrices of order N. The matrix A is given by

$$A_{ij} = e^{-t}\int_0^\pi \phi_i(x)\,\phi_j(x)\,dx = \begin{array}{ll} e^{-t}2h/3 & \text{if } i = j \\ e^{-t}h/6 & \text{if } i = j \pm 1 \\ 0 & \text{otherwise} \end{array}$$

where $h = 1/(N + 1)$ is the mesh spacing. The matrix R is given by

$$R_{ij} = \int_0^\pi \phi_i''(x)\phi_j(x)\,dx = -\int_0^\pi \phi_i'(x)\phi_j'(x)dx = \begin{array}{ll} -2/h & \text{if } i = j \\ 1/h & \text{if } i = j \pm 1 \\ 0 & \text{otherwise} \end{array}$$

The integrals involving

$$\phi_i''$$

are assigned the values of the integrals on the right-hand side, by using the boundary values and integration by parts. Because this system may be stiff, Gear's BDF method is used.

In the following program, the array Y(1:N) corresponds to the vector of coefficients, c. Note that Y contains N + 2 elements; Y(0) and Y(N + 1) are used to store the boundary values. The matrix A depends on t so we set PARAM(19) = 2 and evaluate A when IVPAG returns with IDO = 7. The subroutine FCN computes the vector Rc, and the subroutine FCNJ computes R. The matrices A and R are stored as band-symmetric positive-definite structures having one upper co-diagonal.

```
      INTEGER    LDA, N, NPARAM, NUC
      PARAMETER  (N=9, NPARAM=50, NUC=1, LDA=NUC+1)
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    NSTEP
      PARAMETER  (NSTEP=4)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IATYPE, IDO, IMETH, INORM, ISTEP, MITER, MTYPE
      REAL       A(LDA,N), C, HINIT, PARAM(NPARAM), PI, T, TEND, TMAX,
     &           TOL, XPOINT(0:N+1), Y(0:N+1)
```

```
      CHARACTER  TITLE*10
C                                  SPECIFICATIONS FOR COMMON /COMHX/
      COMMON     /COMHX/ HX
      REAL       HX
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  EXP, REAL, SIN
      REAL       EXP, REAL, SIN
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   IVPAG, SSET, WRRRN
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST, FCN, FCNJ
      REAL       CONST
C                                  Initialize PARAM
      HINIT  = 1.0E-3
      INORM  = 1
      IMETH  = 2
      MITER  = 1
      MTYPE  = 3
      IATYPE = 2
      CALL SSET (NPARAM, 0.0, PARAM, 1)
      PARAM(1)  = HINIT
      PARAM(10) = INORM
      PARAM(12) = IMETH
      PARAM(13) = MITER
      PARAM(14) = MTYPE
      PARAM(16) = NUC
      PARAM(19) = IATYPE
C                                  Initialize other arguments
      PI = CONST('PI')
      HX = PI/REAL(N+1)
      CALL SSET (N-1, HX/6., A(1,2), LDA)
      CALL SSET (N, 2.*HX/3., A(2,1), LDA)
      DO 10  I=0, N + 1
         XPOINT(I) = I*HX
         Y(I)      = SIN(XPOINT(I))
   10 CONTINUE
      TOL  = 1.0E-6
      T    = 0.0
      TMAX = 1.0
C                                  Integrate ODE
      IDO   = 1
      ISTEP = 0
   20 CONTINUE
      ISTEP = ISTEP + 1
      TEND  = TMAX*REAL(ISTEP)/REAL(NSTEP)
   30 CALL IVPAG (IDO, N, FCN, FCNJ, A, T, TEND, TOL, PARAM, Y(1))
C                                  Set matrix A
      IF (IDO .EQ. 7) THEN
         C = EXP(-T)
         CALL SSET (N-1, C*HX/6., A(1,2), LDA)
         CALL SSET (N, 2.*C*HX/3., A(2,1), LDA)
         GO TO 30
      END IF
      IF (ISTEP .LE. NSTEP) THEN
C                                  Print solution
         WRITE (TITLE,'(A,F5.3,A)') 'U(T=', T, ')'
         CALL WRRRN (TITLE, 1, N+2, Y, 1, 0)
C                                  Final call to release workspace
         IF (ISTEP .EQ. NSTEP) IDO = 3
         GO TO 20
       END IF
       END
C
      SUBROUTINE FCN (N, T, Y, YPRIME)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(*), YPRIME(N)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I
C                                  SPECIFICATIONS FOR COMMON /COMHX/
      COMMON     /COMHX/ HX
      REAL       HX
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSCAL
C
      YPRIME(1) = -2.0*Y(1) + Y(2)
      DO 10  I=2, N - 1
         YPRIME(I) = -2.0*Y(I) + Y(I-1) + Y(I+1)
   10 CONTINUE
      YPRIME(N) = -2.0*Y(N) + Y(N-1)
      CALL SSCAL (N, 1.0/HX, YPRIME, 1)
      RETURN
      END
C
      SUBROUTINE FCNJ (N, T, Y, DYPDY)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(*), DYPDY(2,*)
C                                  SPECIFICATIONS FOR COMMON /COMHX/
      COMMON     /COMHX/ HX
      REAL       HX
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSET
C
      CALL SSET (N-1, 1.0/HX, DYPDY(1,2), 2)
      CALL SSET (N, -2.0/HX, DYPDY(2,1), 2)
      RETURN
      END
```

Output

```
                            U(T=0.250)
     1        2        3        4        5        6        7        8
0.0000   0.2321   0.4414   0.6076   0.7142   0.7510   0.7142   0.6076


     9       10       11
0.4414   0.2321   0.0000


                            U(T=0.500)
     1        2        3        4        5        6        7        8
0.0000   0.1607   0.3056   0.4206   0.4945   0.5199   0.4945   0.4206


     9       10       11
0.3056   0.1607   0.0000
```

```
                            U(T=0.750)
     1        2        3        4        5        6        7        8
0.0000   0.1002   0.1906   0.2623   0.3084   0.3243   0.3084   0.2623


     9       10       11
0.1906   0.1002   0.0000


                            U(T=1.000)
     1        2        3        4        5        6        7        8
0.0000   0.0546   0.1039   0.1431   0.1682   0.1768   0.1682   0.1431


     9       10       11
0.1039   0.0546   0.0000
```

BVPFD/DBVPFD (Single/Double precision)

Solve a (parameterized) system of differential equations with boundary conditions at two points, using a variable order, variable step size finite difference method with deferred corrections.

Usage

```
CALL BVPFD (FCNEQN, FCNJAC, FCNBC, FCNPEQ, FCNPBC, N,
            NLEFT, NCUPBC, TLEFT, TRIGHT, PISTEP, TOL,
            NINIT, TINIT, YINIT, LDYINI, LINEAR, PRINT,
            MXGRID, NFINAL, TFINAL, YFINAL, LDYFIN, ERREST)
```

Arguments

FCNEQN — User-supplied `SUBROUTINE` to evaluate derivatives. The usage is `CALL FCNEQN (N, T, Y, P, DYDT)`, where

`N` – Number of differential equations. (Input)
`T` – Independent variable, *t*. (Input)
`Y` – Array of size `N` containing the dependent variable values, *y*(t). (Input)
`P` – Continuation parameter, *p*. (Input)
See Comment 3.
`DYDT` – Array of size `N` containing the derivatives *y′*(*t*). (Output)

The name `FCNEQN` must be declared `EXTERNAL` in the calling program.

FCNJAC — User-supplied `SUBROUTINE` to evaluate the Jacobian. The usage is `CALL FCNJAC (N, T, Y, P, DYPDY)`, where

`N` – Number of differential equations. (Input)
`T` – Independent variable, *t*. (Input)
`Y` – Array of size `N` containing the dependent variable values. (Input)
`P` – Continuation parameter, *p*. (Input)
See Comments 3.
`DYPDY` – `N` by `N` array containing the partial derivatives $a_{i,j} = \partial f_i / \partial y_j$

evaluated at (t, y). The values $a_{i,j}$ are returned in DYPDY(i, j).
(Output)
The name FCNJAC must be declared EXTERNAL in the calling program.

FCNBC — User-supplied SUBROUTINE to evaluate the boundary conditions. The usage is CALL FCNBC (N, YLEFT, YRIGHT, P, H), where

N – Number of differential equations. (Input)
YLEFT – Array of size N containing the values of the dependent variable at the left endpoint. (Input)
YRIGHT – Array of size N containing the values of the dependent variable at the right endpoint. (Input)
P – Continuation parameter, p. (Input)
See Comment 3.
H – Array of size N containing the boundary condition residuals. (Output)
The boundary conditions are defined by $h_i = 0$; for $i = 1, \ldots, N$. The left endpoint conditions must be defined first, then, the conditions involving both endpoints, and finally the right endpoint conditions.

The name FCNBC must be declared EXTERNAL in the calling program.

FCNPEQ — User-supplied SUBROUTINE to evaluate the partial derivative of y' with respect to the parameter p. The usage is
CALL FCNPEQ (N, T, Y, P, DYPDP), where

N – Number of differential equations. (Input)
T – Dependent variable, t. (Input)
Y – Array of size N containing the dependent variable values. (Input)
P – Continuation parameter, p. (Input)
See Comment 3.
DYPDP – Array of size N containing the partial derivatives $a_{i,j} = \partial f_i / \partial y_j$ evaluated at (t, y). The values $a_{i,j}$ are returned in DYPDY(i, j). (Output)

The name FCNPEQ must be declared EXTERNAL in the calling program.

FCNPBC — User-supplied SUBROUTINE to evaluate the derivative of the boundary conditions with respect to the parameter p. The usage is
CALL FCNPBC (N, YLEFT, YRIGHT, P, H), where

N – Number of differential equations. (Input)
YLEFT – Array of size N containing the values of the dependent variable at the left endpoint. (Input)
YRIGHT – Array of size N containing the values of the dependent variable at the right endpoint. (Input)
P – Continuation parameter, p. (Input)
See Comment 3.
H – Array of size N containing the derivative of f_i with respect to p. (Output)

The name FCNPBC must be declared EXTERNAL in the calling program.

N — Number of differential equations.   (Input)

NLEFT — Number of initial conditions.   (Input)
The value NLEFT must be greater than or equal to zero and less than N.

NCUPBC — Number of coupled boundary conditions.   (Input)
The value NLEFT + NCUPBC must be greater than zero and less than or equal to N.

TLEFT — The left endpoint.   (Input)

TRIGHT — The right endpoint.   (Input)

PISTEP — Initial increment size for *p*.   (Input)
If this value is zero, continuation will not be used in this problem. The routines FCNPEQ and FCNPBC will not be called.

TOL — Relative error control parameter.   (Input)
The computations stop when ABS(ERROR(J, I))/MAX(ABS(Y(J, I)), 1.0).LT.TOL for all J = 1, …, N and I = 1, …, NGRID. Here, ERROR(J, I) is the estimated error in Y(J, I).

NINIT — Number of initial grid points, including the endpoints.   (Input)
It must be at least 4.

TINIT — Array of size NINIT containing the initial grid points.   (Input)

YINIT — Array of size N by NINIT containing an initial guess for the values of Y at the points in TINIT.   (Input)

LDYINI — Leading dimension of YINIT exactly as specified in the dimension statement of the calling program.   (Input)

LINEAR — Logical .TRUE. if the differential equations and the boundary conditions are linear.   (Input)

PRINT — Logical .TRUE. if intermediate output is to be printed.   (Input)

MXGRID — Maximum number of grid points allowed.   (Input)

NFINAL — Number of final grid points, including the endpoints.   (Output)

TFINAL — Array of size MXGRID containing the final grid points.   (Output)
Only the first NFINAL points are significant.

YFINAL — Array of size N by MXGRID containing the values of Y at the points in TFINAL.   (Output)

LDYFIN — Leading dimension of YFINAL exactly as specified in the dimension statement of the calling program.   (Input)

ERREST — Array of size N.   (Output)
ERREST(J) is the estimated error in Y(J).

Comments

1.   Automatic workspace usage is

BVPFD N(3N * MXGRID + 4N + 1) + MXGRID * (7N + 2) + 2N * MXGRID + N + MXGRID

DBVPFD 2N(3N * MXGRID + 4N + 1) + 2 * MXGRID(7N + 2) + 2N * MXGRID + N + MXGRID

Workspace may be explicitly provided, if desired, by use of B2PFD/DB2PFD. The reference is

```
CALL B2PFD (FCNEQN, FCNJAC, FCNBC, FCNPEQ, FCNPBC,
            N, NLEFT, NCUPBC, TLEFT, TRIGHT, PISTEP,
            TOL, NINIT, TINIT, YINIT, LDYINI,
            LINEAR, PRINT, MXGRID, NFINAL, TFINAL,
            YFINAL, LDYFIN, ERREST, RWORK, IWORK)
```

The additional arguments are as follows:

RWORK — Floating-point work array of size N(3N * MXGRID + 4N + 1) + MXGRID * (7N + 2).

IWORK — Integer work array of size 2N * MXGRID + N + MXGRID.

2. Informational errors

Type	Code	
4	1	More than MXGRID grid points are needed to solve the problem.
4	2	Newton's method diverged.
3	3	Newton's method reached roundoff error level.

3. If the value of PISTEP is greater than zero, then the routine BVPFD assumes that the user has embedded the problem into a one-parameter family of problems:

$$y' = y'(t, y, p)$$

$$h(y_{tleft}, y_{tright}, p) = 0$$

such that for $p = 0$ the problem is simple. For $p = 1$, the original problem is recovered. The routine BVPFD automatically attempts to increment from $p = 0$ to $p = 1$. The value PISTEP is the beginning increment used in this continuation. The increment will usually be changed by routine BVPFD, but an arbitrary minimum of 0.01 is imposed.

4. The vectors TINIT and TFINAL may be the same.

5. The arrays YINIT and YFINAL may be the same.

Algorithm

The routine BVPFD is based on the subprogram PASVA3 by M. Lentini and V. Pereyra (see Pereyra 1978). The basic discretization is the trapezoidal rule over a nonuniform mesh. This mesh is chosen adaptively, to make the local error approximately the same size everywhere. Higher-order discretizations are obtained by deferred corrections. Global error estimates are produced to control the computation. The resulting nonlinear algebraic system is solved by Newton's method with step control. The linearized system of equations is solved by a special form of Gauss elimination that preserves the sparseness.

Example 1

This example solves the third-order linear equation

$$y''' - 2y'' + y' - y = \sin t$$

subject to the boundary conditions $y(0) = y(2\pi)$ and $y'(0) = y'(2\pi) = 1$. (Its solution is $y = \sin t$.) To use BVPFD, the problem is reduced to a system of first-order equations by defining $y_1 = y$, $y_2 = y'$ and $y_3 = y''$. The resulting system is

$$\begin{aligned} y_1' &= y_2 & \qquad y_2(0) - 1 &= 0 \\ y_2' &= y_3 & \qquad y_1(0) - y_1(2\pi) &= 0 \\ y_3' &= 2y_3 - y_2 + y_1 + \sin t & \qquad y_2(2\pi) - 1 &= 0 \end{aligned}$$

Note that there is one boundary condition at the left endpoint $t = 0$ and one boundary condition coupling the left and right endpoints. The final boundary condition is at the right endpoint. The total number of boundary conditions must be the same as the number of equations (in this case 3).

Note that since the parameter p is not used in the call to BVPFD, the routines FCNPEQ and FCNPBC are not needed. Therefore, in the call to BVPFD, FCNEQN and FCNBC were used in place of FCNPEQ and FCNPBC.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDYFIN, LDYINI, MXGRID, NEQNS, NINIT
      PARAMETER  (MXGRID=45, NEQNS=3, NINIT=10, LDYFIN=NEQNS,
     &           LDYINI=NEQNS)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, J, NCUPBC, NFINAL, NLEFT, NOUT
      REAL       ERREST(NEQNS), PISTEP, TFINAL(MXGRID), TINIT(NINIT),
     &           TLEFT, TOL, TRIGHT, YFINAL(LDYFIN,MXGRID),
     &           YINIT(LDYINI,NINIT)
      LOGICAL    LINEAR, PRINT
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  FLOAT
      REAL       FLOAT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BVPFD, SSET, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST, FCNBC, FCNEQN, FCNJAC
      REAL       CONST, FCNBC, FCNEQN, FCNJAC
C                                 Set parameters
      NLEFT  = 1
      NCUPBC = 1
      TOL    = .001
      TLEFT  = 0.0
      TRIGHT = 2.0*CONST('PI')
      PISTEP = 0.0
      PRINT  = .FALSE.
      LINEAR = .TRUE.
```

```
C                                  Define TINIT
      DO 10  I=1, NINIT
         TINIT(I) = TLEFT + (I-1)*(TRIGHT-TLEFT)/FLOAT(NINIT-1)
   10 CONTINUE
C                                  Set YINIT to zero
      DO 20  I=1, NINIT
         CALL SSET (NEQNS, 0.0, YINIT(1,I), 1)
   20 CONTINUE
C                                  Solve problem
      CALL BVPFD (FCNEQN, FCNJAC, FCNBC, FCNEQN, FCNBC, NEQNS, NLEFT,
     &            NCUPBC, TLEFT, TRIGHT, PISTEP, TOL, NINIT, TINIT,
     &            YINIT, LDYINI, LINEAR, PRINT, MXGRID, NFINAL,
     &            TFINAL, YFINAL, LDYFIN, ERREST)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) (I,TFINAL(I),(YFINAL(J,I),J=1,NEQNS),I=1,
     &                  NFINAL)
      WRITE (NOUT,99999) (ERREST(J),J=1,NEQNS)
99997 FORMAT (4X, 'I', 7X, 'T', 14X, 'Y1', 13X, 'Y2', 13X, 'Y3')
99998 FORMAT (I5, 1P4E15.6)
99999 FORMAT (' Error estimates', 4X, 1P3E15.6)
      END
      SUBROUTINE FCNEQN (NEQNS, T, Y, P, DYDX)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYDX(NEQNS)
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  SIN
      REAL       SIN
C                                  Define PDE
      DYDX(1) = Y(2)
      DYDX(2) = Y(3)
      DYDX(3) = 2.0*Y(3) - Y(2) + Y(1) + SIN(T)
      RETURN
      END
      SUBROUTINE FCNJAC (NEQNS, T, Y, P, DYPDY)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYPDY(NEQNS,NEQNS)
C                                  Define d(DYDX)/dY
      DYPDY(1,1) = 0.0
      DYPDY(1,2) = 1.0
      DYPDY(1,3) = 0.0
      DYPDY(2,1) = 0.0
      DYPDY(2,2) = 0.0
      DYPDY(2,3) = 1.0
      DYPDY(3,1) = 1.0
      DYPDY(3,2) = -1.0
      DYPDY(3,3) = 2.0
      RETURN
      END
      SUBROUTINE FCNBC (NEQNS, YLEFT, YRIGHT, P, F)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       P, YLEFT(NEQNS), YRIGHT(NEQNS), F(NEQNS)
C                                  Define boundary conditions
      F(1) = YLEFT(2) - 1.0
      F(2) = YLEFT(1) - YRIGHT(1)
      F(3) = YRIGHT(2) - 1.0
      RETURN
      END
```

Output

```
 I       T              Y1             Y2             Y3
 1   0.000000E+00  -1.123191E-04   1.000000E+00   6.242319E05
 2   3.490659E-01   3.419107E-01   9.397087E-01  -3.419580E01
 3   6.981317E-01   6.426908E-01   7.660918E-01  -6.427230E-01
 4   1.396263E+00   9.847531E-01   1.737333E-01  -9.847453E-01
 5   2.094395E+00   8.660529E-01  -4.998747E-01  -8.660057E-01
 6   2.792527E+00   3.421830E-01  -9.395474E-01  -3.420648E-01
 7   3.490659E+00  -3.417234E-01  -9.396111E-01   3.418948E-01
 8   4.188790E+00  -8.656880E-01  -5.000588E-01   8.658733E-01
 9   4.886922E+00  -9.845794E-01   1.734571E-01   9.847518E-01
10   5.585054E+00  -6.427721E-01   7.658258E-01   6.429526E-01
11   5.934120E+00  -3.420819E-01   9.395434E-01   3.423986E-01
12   6.283185E+00  -1.123186E-04   1.000000E+00   6.743190E-04
Error estimates     2.840430E-04   1.792939E-04   5.588399E-04
```

Example 2

In this example, the following nonlinear problem is solved:

$$y'' - y^3 + (1 + \sin^2 t) \sin t = 0$$

with $y(0) = y(\pi) = 0$. Its solution is $y = \sin t$. As in Example 1, this equation is reduced to a system of first-order differential equations by defining $y_1 = y$ and $y_2 = y'$. The resulting system is

$$\begin{aligned} y_1' &= y_2 & \qquad y_1(0) &= 0 \\ y_2' &= y_1^3 - \left(1 + \sin^2 t\right) \sin t & \qquad y_1(\pi) &= 0 \end{aligned}$$

In this problem, there is one boundary condition at the left endpoint and one at the right endpoint; there are no coupled boundary conditions.

Note that since the parameter p is not used, in the call to BVPFD the routines FCNPEQ and FCNPBC are not needed. Therefore, in the call to BVPFD, FCNEQN and FCNBC were used in place of FCNPEQ and FCNPBC.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDYFIN, LDYINI, MXGRID, NEQNS, NINIT
      PARAMETER  (MXGRID=45, NEQNS=2, NINIT=12, LDYFIN=NEQNS,
     &           LDYINI=NEQNS)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, J, NCUPBC, NFINAL, NLEFT, NOUT
      REAL       ERREST(NEQNS), PISTEP, TFINAL(MXGRID), TINIT(NINIT),
     &           TLEFT, TOL, TRIGHT, YFINAL(LDYFIN,MXGRID),
     &           YINIT(LDYINI,NINIT)
      LOGICAL    LINEAR, PRINT
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  FLOAT
      REAL       FLOAT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BVPFD, UMACH
```

```
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST, FCNBC, FCNEQN, FCNJAC
      REAL       CONST
C                                 Set parameters
      NLEFT  = 1
      NCUPBC = 0
      TOL    = .001
      TLEFT  = 0.0
      TRIGHT = CONST('PI')
      PISTEP = 0.0
      PRINT  = .FALSE.
      LINEAR = .FALSE.
C                                 Define TINIT and YINIT
      DO 10  I=1, NINIT
         TINIT(I)   = TLEFT + (I-1)*(TRIGHT-TLEFT)/FLOAT(NINIT-1)
         YINIT(1,I) = 0.4*(TINIT(I)-TLEFT)*(TRIGHT-TINIT(I))
         YINIT(2,I) = 0.4*(TLEFT-TINIT(I)+TRIGHT-TINIT(I))
   10 CONTINUE
C                                 Solve problem
      CALL BVPFD (FCNEQN, FCNJAC, FCNBC, FCNEQN, FCNBC, NEQNS, NLEFT,
     &            NCUPBC, TLEFT, TRIGHT, PISTEP, TOL, NINIT, TINIT,
     &            YINIT, LDYINI, LINEAR, PRINT, MXGRID, NFINAL,
     &            TFINAL, YFINAL, LDYFIN, ERREST)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) (I,TFINAL(I),(YFINAL(J,I),J=1,NEQNS),I=1,
     &                  NFINAL)
      WRITE (NOUT,99999) (ERREST(J),J=1,NEQNS)
99997 FORMAT (4X, 'I', 7X, 'T', 14X, 'Y1', 13X, 'Y2')
99998 FORMAT (I5, 1P3E15.6)
99999 FORMAT (' Error estimates', 4X, 1P2E15.6)
      END
      SUBROUTINE FCNEQN (NEQNS, T, Y, P, DYDT)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYDT(NEQNS)
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  SIN
      REAL       SIN
C                                 Define PDE
      DYDT(1) = Y(2)
      DYDT(2) = Y(1)**3 - SIN(T)*(1.0+SIN(T)**2)
      RETURN
      END
      SUBROUTINE FCNJAC (NEQNS, T, Y, P, DYPDY)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYPDY(NEQNS,NEQNS)
C                                 Define d(DYDT)/dY
      DYPDY(1,1) = 0.0
      DYPDY(1,2) = 1.0
      DYPDY(2,1) = 3.0*Y(1)**2
      DYPDY(2,2) = 0.0
      RETURN
      END
      SUBROUTINE FCNBC (NEQNS, YLEFT, YRIGHT, P, F)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       P, YLEFT(NEQNS), YRIGHT(NEQNS), F(NEQNS)
C                                 Define boundary conditions
      F(1) = YLEFT(1)
      F(2) = YRIGHT(1)
      RETURN
      END
```

Output

```
 I       T              Y1              Y2
 1   0.000000E+00   0.000000E+00   9.999277E-01
 2   2.855994E-01   2.817682E-01   9.594315E-01
 3   5.711987E-01   5.406458E-01   8.412407E-01
 4   8.567980E-01   7.557380E-01   6.548904E-01
 5   1.142397E+00   9.096186E-01   4.154530E-01
 6   1.427997E+00   9.898143E-01   1.423307E-01
 7   1.713596E+00   9.898143E-01  -1.423307E-01
 8   1.999195E+00   9.096185E-01  -4.154530E-01
 9   2.284795E+00   7.557380E-01  -6.548903E-01
10   2.570394E+00   5.406460E-01  -8.412405E-01
11   2.855994E+00   2.817683E-01  -9.594313E-01
12   3.141593E+00   0.000000E+00  -9.999274E-01
Error estimates     3.906105E-05   7.124186E-05
```

Example 3

In this example, the following nonlinear problem is solved:

$$y'' - y^3 = \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} - \left(t - \frac{1}{2}\right)^8$$

with $y(0) = y(1) = \pi/2$. As in the previous examples, this equation is reduced to a system of first-order differential equations by defining $y_1 = y$ and $y_2 = y'$. The resulting system is

$$\begin{aligned} y_1' &= y_2 & y_1(0) &= \pi / 2 \\ y_2' &= y_1^3 - \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} + \left(t - \frac{1}{2}\right)^8 & y_1(1) &= \pi / 2 \end{aligned}$$

The problem is embedded in a family of problems by introducing the parameter p and by changing the second differential equation to

$$y_2' = p y_1^3 + \frac{40}{9}\left(t - \frac{1}{2}\right)^{2/3} - \left(t - \frac{1}{2}\right)^8$$

At $p = 0$, the problem is linear; and at $p = 1$, the original problem is recovered. The derivatives $\partial y'/\partial p$ must now be specified in the subroutine `FCNPEQ`. The derivatives $\partial f/\partial p$ are zero in `FCNPBC`.

```
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDYFIN, LDYINI, MXGRID, NEQNS, NINIT
      PARAMETER  (MXGRID=45, NEQNS=2, NINIT=5, LDYFIN=NEQNS,
     &           LDYINI=NEQNS)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
```

```
      INTEGER    NCUPBC, NFINAL, NLEFT, NOUT
      REAL       ERREST(NEQNS), PISTEP, TFINAL(MXGRID), TLEFT, TOL,
     &           XRIGHT, YFINAL(LDYFIN,MXGRID)
      LOGICAL    LINEAR, PRINT
C                                 SPECIFICATIONS FOR SAVE VARIABLES
      INTEGER    I, J
      REAL       TINIT(NINIT), YINIT(LDYINI,NINIT)
      SAVE       I, J, TINIT, YINIT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BVPFD, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCNBC, FCNEQN, FCNJAC, FCNPBC, FCNPEQ
C
      DATA TINIT/0.0, 0.4, 0.5, 0.6, 1.0/
      DATA ((YINIT(I,J),J=1,NINIT),I=1,NEQNS)/0.15749, 0.00215, 0.0,
     &     0.00215, 0.15749, -0.83995, -0.05745, 0.0, 0.05745, 0.83995/
C                                 Set parameters
      NLEFT  = 1
      NCUPBC = 0
      TOL    = .001
      TLEFT  = 0.0
      XRIGHT = 1.0
      PISTEP = 0.1
      PRINT  = .FALSE.
      LINEAR = .FALSE.
C
      CALL BVPFD (FCNEQN, FCNJAC, FCNBC, FCNPEQ, FCNPBC, NEQNS, NLEFT,
     &            NCUPBC, TLEFT, XRIGHT, PISTEP, TOL, NINIT, TINIT,
     &            YINIT, LDYINI, LINEAR, PRINT, MXGRID, NFINAL,
     &            TFINAL, YFINAL, LDYFIN, ERREST)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) (I,TFINAL(I),(YFINAL(J,I),J=1,NEQNS),I=1,
     &                  NFINAL)
      WRITE (NOUT,99999) (ERREST(J),J=1,NEQNS)
99997 FORMAT (4X, 'I', 7X, 'T', 14X, 'Y1', 13X, 'Y2')
99998 FORMAT (I5, 1P3E15.6)
99999 FORMAT (' Error estimates', 4X, 1P2E15.6)
      END
      SUBROUTINE FCNEQN (NEQNS, T, Y, P, DYDT)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYDT(NEQNS)
C                                 Define PDE
      DYDT(1) = Y(2)
      DYDT(2) = P*Y(1)**3 + 40./9.*((T-0.5)**2)**(1./3.) - (T-0.5)**8
      RETURN
      END
      SUBROUTINE FCNJAC (NEQNS, T, Y, P, DYPDY)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYPDY(NEQNS,NEQNS)
C                                 Define d(DYDT)/dY
      DYPDY(1,1) = 0.0
      DYPDY(1,2) = 1.0
      DYPDY(2,1) = P*3.*Y(1)**2
      DYPDY(2,2) = 0.0
      RETURN
      END
      SUBROUTINE FCNBC (NEQNS, YLEFT, YRIGHT, P, F)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       P, YLEFT(NEQNS), YRIGHT(NEQNS), F(NEQNS)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       PI
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST
      REAL       CONST
C                                  Define boundary conditions
      PI   = CONST('PI')
      F(1) = YLEFT(1) - PI/2.0
      F(2) = YRIGHT(1) - PI/2.0
      RETURN
      END
      SUBROUTINE FCNPEQ (NEQNS, T, Y, P, DYPDP)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       T, P, Y(NEQNS), DYPDP(NEQNS)
C                                  Define d(DYDT)/dP
      DYPDP(1) = 0.0
      DYPDP(2) = Y(1)**3
      RETURN
      END
      SUBROUTINE FCNPBC (NEQNS, YLEFT, YRIGHT, P, DFDP)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       P, YLEFT(NEQNS), YRIGHT(NEQNS), DFDP(NEQNS)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSET
C                                  Define dF/dP
      CALL SSET (NEQNS, 0.0, DFDP, 1)
      RETURN
      END
```

Output

```
 I       T              Y1             Y2
 1   0.000000E+00   1.570796E+00  -1.949336E+00
 2   4.444445E-02   1.490495E+00  -1.669567E+00
 3   8.888889E-02   1.421951E+00  -1.419465E+00
 4   1.333333E-01   1.363953E+00  -1.194307E+00
 5   2.000000E-01   1.294526E+00  -8.958461E-01
 6   2.666667E-01   1.243628E+00  -6.373191E-01
 7   3.333334E-01   1.208785E+00  -4.135206E-01
 8   4.000000E-01   1.187783E+00  -2.219351E-01
 9   4.250000E-01   1.183038E+00  -1.584200E-01
10   4.500000E-01   1.179822E+00  -9.973146E-02
11   4.625000E-01   1.178748E+00  -7.233893E-02
12   4.750000E-01   1.178007E+00  -4.638248E-02
13   4.812500E-01   1.177756E+00  -3.399763E-02
14   4.875000E-01   1.177582E+00  -2.205547E-02
15   4.937500E-01   1.177480E+00  -1.061177E-02
16   5.000000E-01   1.177447E+00  -1.479182E-07
17   5.062500E-01   1.177480E+00   1.061153E-02
18   5.125000E-01   1.177582E+00   2.205518E-02
19   5.187500E-01   1.177756E+00   3.399727E-02
20   5.250000E-01   1.178007E+00   4.638219E-02
```

```
21   5.375000E-01   1.178748E+00   7.233876E-02
22   5.500000E-01   1.179822E+00   9.973124E-02
23   5.750000E-01   1.183038E+00   1.584199E-01
24   6.000000E-01   1.187783E+00   2.219350E-01
25   6.666667E-01   1.208786E+00   4.135205E-01
26   7.333333E-01   1.243628E+00   6.373190E-01
27   8.000000E-01   1.294526E+00   8.958461E-01
28   8.666667E-01   1.363953E+00   1.194307E+00
29   9.111111E-01   1.421951E+00   1.419465E+00
30   9.555556E-01   1.490495E+00   1.669566E+00
31   1.000000E+00   1.570796E+00   1.949336E+00
Error estimates     3.448358E-06   5.549869E-05
```

BVPMS/DBVPMS (Single/Double precision)

Solve a (parameterized) system of differential equations with boundary conditions at two points, using a multiple-shooting method.

Usage

```
CALL BVPMS (FCNEQN, FCNJAC, FCNBC, NEQNS, TLEFT, TRIGHT,
            DTOL, BTOL, MAXIT, NINIT, TINIT, YINIT, LDYINI,
            NMAX, NFINAL, TFINAL, YFINAL, LDYFIN)
```

Arguments

FCNEQN — User-supplied SUBROUTINE to evaluate derivatives. The usage is CALL FCNEQN (NEQNS, T, Y, P, DYDT), where

NEQNS – Number of equations. (Input)
T – Independent variable, *t*. (Input)
Y – Array of length NEQNS containing the dependent variable. (Input)
P – Continuation parameter used in solving highly nonlinear problems. (Input)
See Comment 4.
DYDT – Array of length NEQNS containing y' at T. (Output)

The name FCNEQN must be declared EXTERNAL in the calling program.

FCNJAC — User-supplied SUBROUTINE to evaluate the Jacobian. The usage is CALL FCNJAC (NEQNS, T, Y, P, DYPDY), where

NEQNS – Number of equations. (Input)
T – Independent variable. (Input)
Y – Array of length NEQNS containing the dependent variable. (Input)
P – Continuation parameter used in solving highly nonlinear problems. (Input)
See Comment 4.
DYPDY – Array of size NEQNS by NEQNS containing the Jacobian. (Output)
The entry DYPDY(i, j) contains the partial derivative $\partial f_i/\partial y_j$ evaluated at (t, y).

The name FCNJAC must be declared EXTERNAL in the calling program.

FCNBC — User-supplied SUBROUTINE to evaluate the boundary conditions. The usage is CALL FCNBC (NEQNS, YLEFT, YRIGHT, P, H), where

> NEQNS – Number of equations. (Input)
> YLEFT – Array of length NEQNS containing the values of Y at TLEFT. (Input)
> YRIGHT – Array of length NEQNS containing the values of Y at TRIGHT. (Input)
> P – Continuation parameter used in solving highly nonlinear problems. (Input)
> See Comment 4.
> H – Array of length NEQNS containing the boundary function values. (Output)
> The computed solution satisfies (within BTOL) the conditions $h_i = 0$, $i = 1, \ldots,$ NEQNS.

The name FCNBC must be declared EXTERNAL in the calling program.

NEQNS — Number of differential equations. (Input)

TLEFT — The left endpoint. (Input)

TRIGHT — The right endpoint. (Input)

DTOL — Differential equation error tolerance. (Input)
An attempt is made to control the local error in such a way that the global error is proportional to DTOL.

BTOL — Boundary condition error tolerance. (Input)
The computed solution satisfies the boundary conditions, within BTOL tolerance.

MAXIT — Maximum number of Newton iterations allowed. (Input)
Iteration stops if convergence is achieved sooner. Suggested values are MAXIT = 2 for linear problems and MAXIT = 9 for nonlinear problems.

NINIT — Number of shooting points supplied by the user. (Input)
It may be 0. A suggested value for the number of shooting points is 10.

TINIT — Vector of length NINIT containing the shooting points supplied by the user. (Input)
If NINIT = 0, then TINIT is not referenced and the routine chooses all of the shooting points. This automatic selection of shooting points may be expensive and should only be used for linear problems. If NINIT is nonzero, then the points must be an increasing sequence with TINIT(1) = TLEFT and TINIT(NINIT) = TRIGHT.

YINIT — Array of size NEQNS by NINIT containing an initial guess for the values of Y at the points in TINIT. (Input)
YINIT is not referenced if NINIT = 0.

LDYINI — Leading dimension of YINIT exactly as specified in the dimension statement of the calling program. (Input)

NMAX — Maximum number of shooting points to be allowed. (Input)
If `NINIT` is nonzero, then `NMAX` must equal `NINIT`. It must be at least 2.

NFINAL — Number of final shooting points, including the endpoints. (Output)

TFINAL — Vector of length `NMAX` containing the final shooting points. (Output)
Only the first `NFINAL` points are significant.

YFINAL — Array of size `NEQNS` by `NMAX` containing the values of `Y` at the points in `TFINAL`. (Output)

LDYFIN — Leading dimension of `YFINAL` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   `BVPMS` `NEQNS * (NEQNS` + 1)(`NMAX` + 12) + 2 `* NEQNS` + 30
   `DBVPMS` 2 `* NEQNS * (NEQNS` + 1)(`NMAX` + 12) + 3 `* NEQNS` + 60

   Workspace may be explicitly provided, if desired, by use of `B2PMS`/`DB2PMS`. The reference is

   ```
   CALL B2PMS (FCNEQN, FCNJAC, FCNBC, NEQNS, TLEFT,
               TRIGHT, DTOL, BTOL, MAXIT, NINIT, TINIT,
               YINIT, LDYINI, NMAX, NFINAL, TFINAL,
               YFINAL, LDYFIN, WORK, IWK)
   ```

   The additional arguments are as follows:

   WORK — Work array of length `NEQNS * (NEQNS` + 1)(`NMAX` + 12) + `NEQNS` + 30.

   IWK — Work array of length `NEQNS`.

2. Informational errors

Type	Code	
1	5	Convergence has been achieved; but to get acceptably accurate approximations to *y*(*t*), it is often necessary to start an initial-value solver, for example `IVPRK` (page 645), at the nearest `TFINAL`(*i*) point to *t* with *t* ≥ `TFINAL` (*i*). The vectors `YFINAL`(*j*, *i*), *j* = 1, …, `NEQNS` are used as the initial values.
4	1	The initial-value integrator failed. Relax the tolerance `DTOL` or see Comment 3.
4	2	More than `NMAX` shooting points are needed for stability.
4	3	Newton's iteration did not converge in `MAXIT` iterations. If the problem is linear, do an extra iteration. If this error still occurs, check that the

		routine FCNJAC is giving the correct derivatives. If this does not fix the problem, see Comment 3.
4	4	Linear-equation solver failed. The problem may not have a unique solution, or the problem may be highly nonlinear. In the latter case, see Comment 3.

3. Many linear problems will be successfully solved using program-selected shooting points. Nonlinear problems may require user effort and input data. If the routine fails, then increase NMAX or parameterize the problem. With many shooting points the program essentially uses a finite-difference method, which has less trouble with nonlinearities than shooting methods. After a certain point, however, increasing the number of points will no longer help convergence. To parameterize the problem, see Comment 4.

4. If the problem to be solved is highly nonlinear, then to obtain convergence it may be necessary to embed the problem into a one-parameter family of boundary value problems, $y' = f(t, y, p)$, $h(y(t_a, t_b, p)) = 0$ such that for $p = 0$, the problem is simple, e.g., linear; and for $p = 1$, the stated problem is solved. The routine BVPMS/DBVPMS automatically moves the parameter from $p = 0$ toward $p = 1$.

5. This routine is not recommended for stiff systems of differential equations.

Algorithm

Define N = NEQNS, M = NFINAL, t_a = TLEFT and t_b = TRIGHT. The routine BVPMS uses a multiple-shooting technique to solve the differential equation system $y' = f(t, y)$ with boundary conditions of the form

$$h_k(y_1(t_a), \ldots, y_N(t_a), y_1(t_b), \ldots, y_N(t_b)) = 0 \quad \text{for } k = 1, \ldots, N$$

A modified version of IVPRK, page 645, is used to compute the initial-value problem at each "shot." If there are M shooting points (including the endpoints t_a and t_b), then a system of NM simultaneous nonlinear equations must be solved. Newton's method is used to solve this system, which has a Jacobian matrix with a "periodic band" structure. Evaluation of the NM functions and the $NM \times NM$ (almost banded) Jacobian for one iteration of Newton's method is accomplished in one pass from t_a to t_b of the modified IVPRK, operating on a system of $N(N + 1)$ differential equations. For most problems, the total amount of work should not be highly dependent on M. Multiple shooting avoids many of the serious ill-conditioning problems that plague simple shooting methods. For more details on the algorithm, see Sewell (1982).

The boundary functions should be scaled so that all components h_k are of comparable magnitude since the absolute error in each is controlled.

Example

The differential equations that model an elastic beam are (see Washizu 1968, pages 142–143):

$$\mathbf{M}_{xx} - \frac{\mathbf{NM}}{\mathbf{EI}} + \mathbf{L}(x) = 0$$

$$\mathbf{EIW}_{xx} + \mathbf{M} = 0$$

$$\mathbf{EA}_0\left(\mathbf{U}_x + \mathbf{W}_x^2 / 2\right) - \mathbf{N} = 0$$

$$\mathbf{N}_x = 0$$

where **U** is the axial displacement, **W** is the transverse displacement, **N** is the axial force, **M** is the bending moment, **E** is the elastic modulus, **I** is the moment of inertia, $\mathbf{A}_0$ is the cross-sectional area, and **L**(x) is the transverse load.

Assume we have a clamped cylindrical beam of radius 0.1in, a length of 10in, and an elastic modulus $\mathbf{E} = 10.6 \times 10^6$ lb/in^2. Then, $\mathbf{I} = 0.784 \times 10^{-4}$, and $\mathbf{A}_0 = \pi 10^{-2}$ in^2, and the boundary conditions are $\mathbf{U} = \mathbf{W} = \mathbf{W}_x = 0$ at each end. If we let $y_1 = \mathbf{U}$, $y_2 = \mathbf{N}/\mathbf{EA}_0$,

$y_3 = \mathbf{W}$, $y_4 = \mathbf{W}_x$, $y_5 = \mathbf{M}/\mathbf{EI}$, and $y_6 = \mathbf{M}_x/\mathbf{EI}$, then the above nonlinear equations can be written as a system of six first-order equations.

$$y_1' = y_2 - \frac{y_4^2}{2}$$

$$y_2' = 0$$

$$y_3' = y_4$$

$$y_4' = -y_5$$

$$y_5' = y_6$$

$$y_6' = \frac{\mathbf{A}_0 y_2 y_5}{\mathbf{I}} - \frac{\mathbf{L}(x)}{\mathbf{EI}}$$

The boundary conditions are $y_1 = y_3 = y_4 = 0$ at $x = 0$ and at $x = 10$. The loading function is $\mathbf{L}(x) = -2$, if $3 \le x \le 7$, and is zero elsewhere.

The material parameters, $\mathbf{A}_0$ = `A0`, **I** = `AI`, and **E**, are passed to the evaluation subprograms using the common block `PARAM`.

```
      INTEGER    LDY, NEQNS, NMAX
      PARAMETER  (NEQNS=6, NMAX=21, LDY=NEQNS)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, MAXIT, NFINAL, NINIT, NOUT
      REAL       TOL, X(NMAX), XLEFT, XRIGHT, Y(LDY,NMAX)
C                                 SPECIFICATIONS FOR COMMON /PARAM/
      COMMON     /PARAM/ A0, AI, E
      REAL       A0, AI, E
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  REAL
      REAL       REAL
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BVPMS, SSET, UMACH,
     & FCNBC, FCNEQN, FCNJAC
C                                 Set material parameters
      A0 = 3.14E-2
      AI = 0.784E-4
      E  = 10.6E6
C                                 Set parameters for BVPMS
      XLEFT  = 0.0
      XRIGHT = 10.0
      TOL    = 1.0E-4
      MAXIT  = 19
      NINIT  = NMAX
C                                 Define the shooting points
      DO 10  I=1, NINIT
         X(I) = XLEFT + REAL(I-1)/REAL(NINIT-1)*(XRIGHT-XLEFT)
         CALL SSET (NEQNS, 0.0, Y(1,I), 1)
   10 CONTINUE
C                                 Solve problem
      CALL BVPMS (FCNEQN, FCNJAC, FCNBC, NEQNS, XLEFT, XRIGHT, TOL,
     &            TOL, MAXIT, NINIT, X, Y, LDY, NMAX, NFINAL, X, Y,
     &            LDY)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,’(26X,A/12X,A,10X,A,7X,A)’) ’Displacement’,
     &                                      ’X’, ’Axial’, ’Transvers’//
     &                                      ’e’
      WRITE (NOUT,’(F15.1,1P2E15.3)’) (X(I),Y(1,I),Y(3,I),I=1,NFINAL)
      END
      SUBROUTINE FCNEQN (NEQNS, X, Y, P, DYDX)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       X, P, Y(NEQNS), DYDX(NEQNS)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       FORCE
C                                 SPECIFICATIONS FOR COMMON /PARAM/
      COMMON     /PARAM/ A0, AI, E
      REAL       A0, AI, E
C                                 Define derivatives
      FORCE = 0.0
      IF (X.GT.3.0 .AND. X.LT.7.0) FORCE = -2.0
      DYDX(1) = Y(2) - P*0.5*Y(4)**2
      DYDX(2) = 0.0
      DYDX(3) = Y(4)
      DYDX(4) = -Y(5)
      DYDX(5) = Y(6)
      DYDX(6) = P*A0*Y(2)*Y(5)/AI - FORCE/E/AI
      RETURN
      END
      SUBROUTINE FCNBC (NEQNS, YLEFT, YRIGHT, P, F)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       P, YLEFT(NEQNS), YRIGHT(NEQNS), F(NEQNS)
C                                 SPECIFICATIONS FOR COMMON /PARAM/
      COMMON     /PARAM/ A0, AI, E
      REAL       A0, AI, E
C                                 Define boundary conditions
      F(1) = YLEFT(1)
      F(2) = YLEFT(3)
      F(3) = YLEFT(4)
      F(4) = YRIGHT(1)
      F(5) = YRIGHT(3)
      F(6) = YRIGHT(4)
      RETURN
      END
      SUBROUTINE FCNJAC (NEQNS, X, Y, P, DYPDY)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NEQNS
      REAL       X, P, Y(NEQNS), DYPDY(NEQNS,NEQNS)
C                                 SPECIFICATIONS FOR COMMON /PARAM/
      COMMON     /PARAM/ A0, AI, E
      REAL       A0, AI, E
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SSET
C                                 Define partials, d(DYDX)/dY
      CALL SSET (NEQNS**2, 0.0, DYPDY, 1)
      DYPDY(1,2) = 1.0
      DYPDY(1,4) = -P*Y(4)
      DYPDY(3,4) = 1.0
      DYPDY(4,5) = -1.0
      DYPDY(5,6) = 1.0
      DYPDY(6,2) = P*Y(5)*A0/AI
      DYPDY(6,5) = P*Y(2)*A0/AI
      RETURN
      END
```

Output

```
                 Displacement
  X         Axial       Transverse
  0.0      1.631E-11     -8.677E-10
  5.0      1.914E-05     -1.273E-03
 10.0      2.839E-05     -4.697E-03
 15.0      2.461E-05     -9.688E-03
 20.0      1.008E-05     -1.567E-02
 25.0     -9.550E-06     -2.206E-02
 30.0     -2.721E-05     -2.830E-02
 35.0     -3.644E-05     -3.382E-02
 40.0     -3.379E-05     -3.811E-02
 45.0     -2.016E-05     -4.083E-02
 50.0     -4.414E-08     -4.176E-02
 55.0      2.006E-05     -4.082E-02
 60.0      3.366E-05     -3.810E-02
 65.0      3.627E-05     -3.380E-02
 70.0      2.702E-05     -2.828E-02
 75.0      9.378E-06     -2.205E-02
 80.0     -1.021E-05     -1.565E-02
 85.0     -2.468E-05     -9.679E-03
 90.0     -2.842E-05     -4.692E-03
 95.0     -1.914E-05     -1.271E-03
100.0      0.000E+00      0.000E+00
```

DASPG/DDASPG (Single/Double precision)

Solve a first order differential-algebraic system of equations, $g(t, y, y') = 0$, using the Petzold−Gear BDF method.

Usage

```
CALL DASPG (N, T, TOUT, IDO, Y, YPR, GCN)
```

Arguments

N — Number of differential equations. (Input)

T — Independent variable, t. (Input/Output)
Set `T` to the starting value t_0 at the first step.

TOUT — Final value of the independent variable. (Input)
Update this value when re-entering after output, `IDO` = 2.

IDO — Flag indicating the state of the computation. (Input/Output)

IDO	**State**
1	Initial entry
2	Normal re-entry after obtaining output
3	Release workspace
4	Return because of an error condition

The user sets `IDO` = 1 or `IDO` = 3. All other values of `IDO` are defined as output. The initial call is made with `IDO` = 1 and `T` = t_0. The routine then sets `IDO` = 2, and this value is used for all but the last entry that is made with `IDO` = 3. This call is used to release workspace and other final tasks. Values of `IDO` larger than 4 occur only when calling the second-level routine `D2SPG` and using the options associated with reverse communication.

Y — Array of size `N` containing the dependent variable values, y. This array must contain initial values. (Input/Output)

YPR — Array of size `N` containing derivative values, y'. This array must contain initial values. (Input/Output)
The routine will solve for consistent values of y' to satisfy the equations at the starting point.

GCN — User-supplied `SUBROUTINE` to evaluate $g(t, y, y')$. The usage is `CALL GCN (N, T, Y, YPR, GVAL)`, where `GCN` must be declared `EXTERNAL` in the calling program. The routine will solve for values of $y'(t_0)$ so that $g(t_0, y, y') = 0$. The user can signal that g is not defined at requested values of (t, y, y') using an option. This causes the routine to reduce the step size or else quit.

`GVAL` – Array of size `N` containing the function values, $g(t, y, y')$. (Output)

Comments

Users can often get started using the routine DASPG/DDASPG without reading beyond this point in the documentation. There is often no reason to use options when getting started. Those readers who do not want to use options can turn directly to the first two examples. The following tables give numbers and key phrases for the options. A detailed guide to the options is given below in Comment 2.

Value	Brief or Key Phrase for INTEGER Option
6	INTEGER option numbers
7	Floating-point option numbers
IN(1)	First call to DASPG, D2SPG
IN(2)	Scalar or vector tolerances
IN(3)	Return for output at intermediate steps
IN(4)	Creep up on special point, TSTOP
IN(5)	Provide (analytic) partial derivative formulas
IN(6)	Maximum number of steps
IN(7)	Control maximum step size
IN(8)	Control initial step size
IN(9)	*Not Used*
IN(10)	Constrain dependent variables
IN(11)	Consistent initial data
IN(12-15)	*Not Used*
IN(16)	Number of equations
IN(17)	What routine did, if any errors
IN(18)	Maximum BDF order
IN(19)	Order of BDF on next move
IN(20)	Order of BDF on previous move
IN(21)	Number of steps
IN(22)	Number of *g* evaluations
IN(23)	Number of derivative matrix evaluations
IN(24)	Number of error test failures
IN(25)	Number of convergence test failures
IN(26)	Reverse communiction for *g*
IN(27)	Where is *g* stored?
IN(28)	Panic flag
IN(29)	Reverse communication, for partials

Value	Brief or Key Phrase for INTEGER Option
IN(30)	Where are partials stored?
IN(31)	Reverse communication, for solving
IN(32)	*Not Used*
IN(33)	Where are vector tolerances stored?
IN(34)	Is partial derivative array allocated?
IN(35)	User's work arrays sizes are checked
IN(36-50)	*Not used*

Table 1. Key Phrases for Floating-Point Options

Value	Brief or Key Phrase for Floating-Point Option
INR(1)	Value of *t*
INR(2)	Farthest internal *t* vaue of integration
INR(3)	Value of `TOUT`
INR(4)	A stopping point of integration before `TOUT`
INR(5)	Values of two scalars `ATOL`, `RTOL`
INR(6)	Initial step size to use
INR(7)	Maximum step allowed
INR(8)	Condition number reciprocal
INR(9)	Value of c_j for partials
INR(10)	Step size on the next move
INR(11)	Step size on the previous move
INR(12-20)	*Not Used*

Table 2. Number and Key Phrases for Floating-Point Options

1. Automatic workspace usage is

   `DASPG` 76 + (`MAXORD` + 6)`N` + (`N` + `K`)`N`(1 - `L`) units, or
   `DDASPG` 117 + 2((`MAXORD` + 6)`N` + (`N` + `K`)`N`(1 - `L`)) + `N` units.

   The default value for `MAXORD` is 5. The value for the nonnegative integer `K` is determined by option **16** of `LSLRG` (page 12), values `IVAL`(3) and `IVAL`(4). The default value for `K` does not exceed `1`. The default value for `L` is `0`. With option **IN(34)**, `L` can be replaced by `1`. This reduction in the amount of work space required by the routine is provided for users who are going to save their partial derivative matrix and solve their own linear algebraic equations using reverse communication.

   Workspace may be explicitly provided, and many of the options utilized by directly calling `D2SPG`/`DD2SPG`. The reference is

```
CALL D2SPG (N, T, TOUT, IDO, Y, YPR, GCN, JGCN, IWK,
            WK)
```

The additional arguments are as follows:

IDO	State
5	Return for evaluation of $g(t, y, y')$
6	Return for evaluation of matrix $A = [\partial g/\partial y + c_j \partial g/\partial y']$
7	Return for factorization of the matrix $A = [\partial g/\partial y + c_j \partial g/\partial y']$
8	Return for solution of $A\Delta y = \Delta g$

These values of IDO occur only when calling the second-level routine D2SPG and using options associated with reverse communication. The routine D2SPG/DD2SPG is reentered.

GCN — A Fortran SUBROUTINE to compute $g(t, y, y')$. This routine is normally provided by the user. That is the default case. The dummy IMSL routine DGSPG/DDGSPG may be used as this argument when $g(t, y, y')$ is evaluated by reverse communication. In either case, a name must be declared in a Fortran EXTERNAL statement. If usage of the dummy IMSL routine is intended, then the name DGSPG/DDGSPG should be specified. The dummy IMSL routine will never be called under this optional usage of reverse communication. An example of reverse communication for evaluation of g is given in Example 4.

JGCN — A Fortran SUBROUTINE to compute partial derivatives of $g(t, y, y')$. This routine may be provided by the user. The dummy IMSL routine DJSPG/DDJSPG may be used as this argument when partial derivatives are computed using divided differences. This is the default. The dummy routine is not called under default conditions. If partial derivatives are to be explicitly provided, the routine JGCN must be written by the user or reverse communication can be used. An example of reverse communication for evaluation of the partials is given in Example 4.

If the user writes a routine with the *fixed* name DJSPG/DDJSPG, then partial derivatives can be provided while calling DASPG. An option is used to signal that formulas for partial derivatives are being supplied. This is illustrated in Example 3. The name of the partial derivative routine must be declared in a Fortran EXTERNAL statement when calling D2SPG. If usage of the dummy IMSL routine is intended, then the name DJSPG/DDJSPG should be specified for this EXTERNAL name. Whenever the user provides partial derivative evaluation formulas, by whatever means, that must be noted with an option. Usage of the derivative evaluation routine is CALL JGCN (N, T, Y, YPR, CJ, PDG, LDPDG) where

Arg	**Definition**
N	Number of equations. (Input)
T	Independent variable, *t*. (Input)
Y	Array of size N containing the values of the dependent variables, *y*. (Input)
YPR	Array of size N containing the values of the derivatives, y'. (Input)
CJ	The value c_j used in computing the partial derivatives returned in PDG. (Input)
PDG	Array of size LDPDG * N containing the partial derivatives $A = [\partial g/\partial y + c_j \partial g/\partial y']$. Each nonzero derivative entry a_{ij} is returned in the array location PDG(i, j). The array contents are zero when the routine is called. Thus, only the nonzero derivatives have to be defined in the routine JGCN. (Output)
LDPDG	The leading dimension of PDG. Normally, this value is N. It is a value larger than N under the conditions explained in option **16** of LSLRG (page 12).

JGCN must be declared EXTERNAL in the calling program.

IWK — Work array of integer values. The size of this array is 35 + N. The contents of IWK must not be changed from the first call with IDO = 1 until after the final call with IDO = 3.

WK — Work array of floating-point values in the working precision. The size of this array is 41 + (MAXORD + 6)N + (N + K)N(1 − L) where K is determined from the values IVAL(3) and IVAL(4) of option **16** of LSLRG (page 1173). The value of L is 0 unless option **IN(34)** is used to avoid allocation of the array containing the partial derivatives. With the use of this option, L can be set to 1. The contents of array WK must not be changed from the first call with IDO = 1 until after the final call.

2. Integer and Floating-Point Options with Chapter 10 Options Manager

The routine DASPG allows the user access to many interface parameters and internal working variables by the use of options. The options manager subprograms IUMAG (page 1173), SUMAG (page 1175) and DUMAG (page 1178), are used to change options from their default values or obtain the current values of required parameters.

Options of type INTEGER:

6 This is the list of numbers used for INTEGER options. Users will typically call this option first to get the numbers, IN(I), I = 1, 50. This option has 50 entries. The default values are IN(I) = I + 50, I = 1, 50.

7 This is the list of numbers used for REAL and DOUBLE PRECISION options. Users will typically call this option first to get the numbers, INR(I), I = 1,20. This option has 20 entries. The default values are INR(I) = I + 50, I = 1, 20.

IN(1) This is the first call to the routine DASPG or D2SPG. Value is 0 for the first call, 1 for further calls. Setting IDO = 1 resets this option to its default. Default value is 0.

IN(2) This flag controls the kind of tolerances to be used for the solution. Value is 0 for scalar values of absolute and relative tolerances applied to all components. Value is 1 when arrays for both these quantities are specified. In this case, the option **IN(33)** is used to get the offset into WK where the 2N array values are to be placed: all ATOL values followed by all RTOL values. This offset is defined after the call to the routine D2SPG so users will have to call the options manager at a convenient place in the GCN routine or during reverse communication. Default value is 0.

IN(3) This flag controls when the code returns to the user with output values of *y* and *y*′. If the value is 0, it returns to the user at T = TOUT only. If the value is 1, it returns to the user at an internal working step. Default value is 0.

IN(4) This flag controls whether the code should integrate past a special point, TSTOP, and then interpolate to get *y* and *y*′at TOUT. If the value is 0, this is permitted. If the value is 1, the code assumes the equations either change on the alternate side of TSTOP or they are undefined there. In this case, the code creeps up to TSTOP in the direction of integration. The value of TSTOP is set with option INR(4). Default value is 0.

IN(5) This flag controls whether partial derivatives are computed using divided onesided differences, or they are to be computed using user-supplied evaluation formulas. If the value is 0, use divided differences. If the value is 1, use formulas for the partial derivatives. See Example 3 for an illustration of one way to do this. Default value is 0.

IN(6) The maximum number of steps. Default value is 500.

IN(7) This flag controls a maximum magnitude constraint for the step size. If the value is 0, the routine picks its own maximum. If the value is 1, a maximum is specified by the user. That value is set with option number **INR(7)**. Default value is 0.

IN(8) This flag controls an initial value for the step size. If the value is 0, the routine picks its own initial step size. If the value is 1, a starting step size is specified by the user. That value is set with option number **INR(6)**. Default value is 0.

IN(9) Not used. Default value is 0.

IN(10) This flag controls attempts to constrain all components to be nonnegative. If the value is 0, no constraints are enforced. If value is 1, constraint is enforced. Default value is 0.

IN(11) This flag controls whether the initial values (t, y, y') are consistent. If the value is 0, $g(t, y, y') = 0$ at the initial point. If the value is 1, the routine will try to solve for y' to make this equation satisfied. Default value is 1.

IN(12-15) Not used. Default value is 0 for each option.

IN(16) The number of equations in the system, n. Default value is 0.

IN(17) This value reports what the routine did. Default value is 0.

Value	Explanation
1	A step was taken in the intermediate output mode. The value `TOUT` has not been reached.
2	The integration to exactly `TSTOP` was completed.
3	The integration to `TSTOP` was completed by stepping past `TSTOP` and interpolating to evaluate y and y'.
–1	Too many steps taken.
–2	Error tolerances are too small.
–3	A pure relative error tolerance can’t be satisfied.
–6	There were repeated error test failures on the last step.
–7	The BDF corrector equation solver did not converge.
–8	The matrix of partial derivatives is singular.
–10	The BDF corrector equation solver did not converge because the evaluation failure flag was raised.
–11	The evaluation failure flag was raised to quit.
–12	The iteration for the initial vaule of y' did not converge.
–33	There is a fatal error, perhaps caused by invalid input.

Table 3. What the Routine `DASPG` or `D2SPG` Did

IN(18) The maximum order of BDF formula the routine should use. Default value is 5.

IN(19) The order of the BDF method the routine will use on the next step. Default value is `IMACH`(5).

IN(20) The order of the BDF method used on the last step. Default value is `IMACH`(5).

IN(21) The number of steps taken so far. Default value is 0.

IN(22) The number of times that g has been evaluated. Default value is 0.

IN(23) The number of times that the partial derivative matrix has been evaluated. Default value is 0.

IN(24) The total number of error test failures so far. Default value is 0.

IN(25) The total number of convergence test failures so far. This includes singular iteration matrices. Default value is 0.

IN(26) Use reverse communication to evaluate g when this value is 0. If the value is 1, forward communication is used. Use the routine `D2SPG` for reverse communication. With reverse communication, a return will be made with `IDO` = 5. Compute the value of *g*, place it into the array `WK` at the offset obtained with option **IN(27**), and re-enter the routine. Default value is 1.

IN(27) The user is to store the evaluated function *g* during reverse communication in the work array `WK` using this value as an offset. Default value is `IMACH`(5).

IN(28) This value is a “panic flag.” After an evaluation of *g*, this value is checked. The value of *g* is used if the flag is 0. If it has the value −1, the routine reduces the step size and possibly the order of the BDF. If the value is −2, the routine returns control to the user immediately. This option is also used to signal a singular or poorly conditioned partial derivative matrix encountered during the factor phase in reverse communication. Use a nonzero value when the matrix is singular. Default value is 0.

IN(29) Use reverse communication to evaluate the partial derivative matrix when this value is 0. If the value is 1, forward communication is used. Use the routine `D2SPG` for reverse communication. With reverse communication, a return will be made with `IDO` = 6. Compute the partial derivative matrix *A* and re-enter the routine. If forward communication is used for the linear solver, return the partials using the offset into the array `WK`. This offset value is obtained with option **IN(30)**. Default value is 1.

IN(30) The user is to store the values of the partial derivative matrix *A* by columns in the work array `WK` using this value as an offset. The option **16** for `LSLRG` is used here to compute the row dimension of the internal working array that contains *A*. Users can also choose to store this matrix in some convenient form in their calling program if they are providing linear system solving using reverse communication. See options **IN(31)** and **IN(34)**. Default value is `IMACH`(5).

IN(31) Use reverse communication to solve the linear system $A\Delta y = \Delta g$ if this value is 0. If the value is 1, use forward communication into the routines `L2CRG` (page 15) and `LFSRG` (page 20) for the linear system solving. Return the solution using the offset into the array `WK` where *g* is stored. This offset value is obtained with option **IN(27)**. With reverse communication, a return will be made with `IDO` = 7 for factorization of *A* and with `IDO` = 8 for solving the system. Re-enter the routine in both cases. If the matrix A is singular or poorly conditioned, raise the "panic flag," option **IN(28)**, during the factorization. Default value is 1.

IN(32) Not used. Default value is 0.

IN(33) The user is to store the vector of values for `ATOL` and `RTOL` in the array `WK` using this value as an offset. The routine `D2SPG` must be called before this value is defined.

IN(34) This flag is used if the user has not allocated storage for the matrix *A* in the array `WK`. If the value is 0, storage is allocated. If the value is 1, storage was not allocated. In this case, the user must be using reverse communication to evaluate the partial derivative matrix and to solve the linear systems $A\Delta y = \Delta g$. Default value is 0.

IN(35) These two values are the sizes of the arrays `IWK` and `WK` allocated in the users program. The values are checked against the program requirements. These checks are made only if the values are positive. Users will normally set this option when directly calling `D2SPG`. Default values are (0, 0).

Options of type `REAL` or `DOUBLE PRECISION`:

INR(1) The value of the independent variable, *t*. Default value is `AMACH`(6).

INR(2) The farthest working *t* point the integration has reached. Default value is `AMACH`(6) .

INR(3) The current value of `TOUT`. Default value is `AMACH`(6).

INR(4) The next special point, `TSTOP`, before reaching `TOUT`. Default value is `AMACH`(6). Used with option **IN(4)**.

INR(5) The pair of scalar values `ATOL` and `RTOL` that apply to the error estimates of all components of *y*. Default values for both are `SQRT`(`AMACH`(4)).

INR(6) The initial step size if `DASPG` is not to compute it internally. Default value is `AMACH`(6).

INR(7) The maximum step size allowed. Default value is `AMACH`(2).

INR(8) This value is the reciprocal of the condition number of the matrix *A*. It is defined when forward communication is used to solve for the linear updates to the BDF corrector equation. No further program action, such as declaring a singular system, based on the condition number. Users can declare the system to be singular by raising the “panic flag” using option **IN(28)**. Default value is `AMACH`(6).

INR(9) The value of c_j used in the partial derivative matrix for reverse communication evaluation. Default value is `AMACH`(6).

INR(10) The step size to be attempted on the next move. Default value is `AMACH`(6).

INR(11) The step size taken on the previous move. Default value is `AMACH`(6).

4. Norm Function Subprogram

The routine `DASPG` uses a weighted Euclidean-RMS norm to measure the size of the estimated error in each step. This is done using a `FUNCTION` subprogram: `REAL FUNCTION D10PG (N, V, WT)`. This routine returns the value of the RMS weighted norm given by:

$$\mathrm{D10PG} = \sqrt{N^{-1} \sum_{i=1}^{N} \left(v_i / wt_i\right)^2}$$

Users can replace this function with one of their own choice. This should be done only for problem-related reasons.

Algorithm

Routine `DASPG` finds an approximation to the solution of a system of differential-algebraic equations $g(t, y, y') = 0$, with given initial data for y and y'. The routine uses BDF formulas, appropriate for systems of stiff ODEs, and attempts to keep the global error proportional to a user-specified tolerance. See Brenan et al. (1989). This routine is efficient for stiff systems of index 1 or index 0. See Brenan et al. (1989) for a definition of *index*. Users are encouraged to use `DOUBLE PRECISION` accuracy on machines with a short `REAL` precision accuracy. The examples given below are in `REAL` accuracy because of the desire for consistency with the rest of IMSL MATH/LIBRARY examples. The routine `DASPG` is based on the code `DASSL` designed by L. Petzold (1982-1990).

Example 1

The Van der Pol equation $u'' + \mu(u^2 - 1)\, u' + u = 0$, $\mu > 0$, is a single ordinary differential equation with a periodic limit cycle. See Hartman (1964, page 181). For the value $\mu = 5$, the equations are integrated from $t = 0$ until the limit has clearly developed at $t = 26$. The (arbitrary) initial conditions used here are $u(0) = 2$ and $u'(0) = - 2/3$. Except for these initial conditions and the final t value, this is problem (E2) of the Enright and Pryce (1987) test package. This equation is solved as a differential-algebraic system by defining the first-order system:

$$\begin{aligned}\varepsilon &= 1/\mu \\ y_1 &= u \\ g_1 &= y_2 - y_1' = 0 \\ g_2 &= \left(1-y_1^2\right)y_2 - \varepsilon\left(y_1 + y_2'\right) = 0\end{aligned}$$

Note that the initial condition for

$$y_2'$$

in the sample program is not consistent, $g_2 \neq 0$ at $t = 0$. The routine DASPG solves for this starting value. No options need to be changed for this usage. The set of pairs ($u(t_j)$, $u'(t_j)$) are accumulated for the 260 values $t_j = 0.1, 26, (0.1)$.

```
      INTEGER    N, NP
      PARAMETER  (N=2, NP=260)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    ISTEP, NOUT, NSTEP
      REAL DELT,  T, TEND, U(NP), UPR(NP), Y(N), YPR(N)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   DASPG, UMACH
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   GCN
C                                  Define initial data
      IDO = 1
      T = 0.0
      TEND = 26.0
      DELT = 0.1
      NSTEP = TEND/DELT
C                                  Initial values
      Y(1) = 2.0
      Y(2) = -2.0/3.0
C                                  Initial derivatives
      YPR(1) = Y(2)
      YPR(2) = 0.
C                                  Write title
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
C                                  Integrate ODE/DAE
      ISTEP = 0
   10 CONTINUE
      ISTEP = ISTEP + 1
      CALL DASPG (N, T, T+DELT, IDO, Y, YPR, GCN)
C                                  Save solution for plotting
      IF (ISTEP .LE. NSTEP) THEN
         U(ISTEP) = Y(1)
         UPR(ISTEP) = YPR(1)
C                                  Release work space
         IF (ISTEP .EQ. NSTEP) IDO = 3
         GO TO 10
      END IF
      WRITE (NOUT,99999) TEND, Y, YPR
99998 FORMAT (11X, ’T’, 14X, ’Y(1)’, 11X, ’Y(2)’, 10X, ’Y’’(1)’, 10X,
     &       ’Y’’(2)’)
```

```
99999 FORMAT (5F15.5)
C                                        Start plotting
C      CALL SCATR (NSTEP, U, UPR)
C      CALL EFSPLT (0, ' ')
      STOP
      END
C
      SUBROUTINE GCN (N, T, Y, YPR, GVAL)
C                                        SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL T, Y(N), YPR(N), GVAL(N)
C                                        SPECIFICATIONS FOR LOCAL VARIABLES
      REAL EPS
C
      EPS = 0.2
C
      GVAL(1) = Y(2) - YPR(1)
      GVAL(2) = (1.0-Y(1)**2)*Y(2) - EPS*(Y(1)+YPR(2))
      RETURN
      END
```

Output

```
     T               Y(1)           Y(2)           Y'(1)          Y'(2)
26.00000          1.45330       -0.24486       -0.24713       -0.09399
```

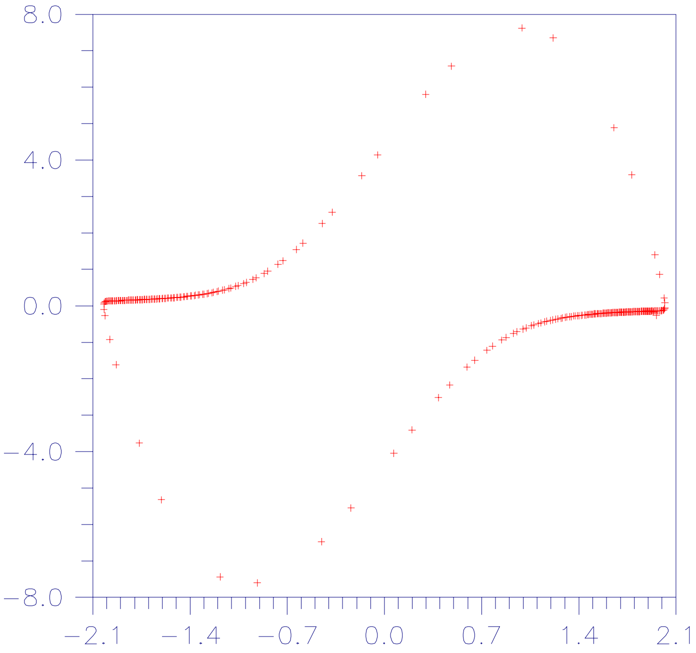


Figure 5-1   Van der Pol Cycle, (u(t), u'(t)), μ = 5.

Example 2

The first-order equations of motion of a point-mass *m* suspended on a massless wire of length ℓ under the influence of gravity force, *mg* and tension value λ, in Cartesian coordinates, (*p*, *q*), are

$$\begin{aligned} p' &= u \\ q' &= v \\ mu' &= -p\lambda \\ mv' &= -q\lambda - mg \\ p^2 + q^2 - \ell^2 &= 0 \end{aligned}$$

This is a genuine differential-algebraic system. The problem, as stated, has an index number equal to the value 3. Thus, it cannot be solved with `DASPG` directly. Unfortunately, the fact that the index is greater than 1 must be deduced indirectly. Typically there will be an error processed which states that the (BDF) corrector equation did not converge. The user then differentiates and replaces the constraint equation. This example is transformed to a problem of index number of value 1 by differentiating the last equation twice. This resulting equation, which replaces the given equation, is the total energy balance:

$$m(u^2 + v^2) - mgq - \ell^2\lambda = 0$$

With initial conditions and systematic definitions of the dependent variables, the system becomes:

$$\begin{aligned} p(0) &= \ell, q(0) = u(0) = v(0) = \lambda(0) = 0 \\ y_1 &= p \\ y_2 &= q \\ y_3 &= u \\ y_4 &= v \\ y_5 &= \lambda \\ g_1 &= y_3 - y_1' = 0 \\ g_2 &= y_4 - y_2' = 0 \\ g_3 &= -y_1 y_5 - my_3' = 0 \\ g_4 &= -y_2 y_5 - mg - my_4' = 0 \\ g_5 &= m\left(y_3^2 + y_4^2\right) - mgy_2 - \ell^2 y_5 = 0 \end{aligned}$$

The problem is given in English measurement units of feet, pounds, and seconds. The wire has length 6.5 *ft*, and the mass at the end is 98 *lb*. Usage of the software does not require it, but standard or “SI” units are used in the numerical model. This conversion of units is done as a first step in the user-supplied evaluation routine, GCN. A set of initial conditions, corresponding to the pendulum starting in a horizontal position, are provided as output for the input signal of $n = 0$. The maximum magnitude of the tension parameter, $\lambda(t) = y_5(t)$, is computed at the output points, $t = 0.1, \pi, (0.1)$. This extreme value is converted to English units and printed.

```
      INTEGER    N
      PARAMETER  (N=5)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, ISTEP, NOUT, NSTEP
      REAL       DELT, GVAL(N), MAXLB, MAXTEN, T, TEND, TMAX, Y(N),
     &           YPR(N)
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  ABS
      REAL       ABS
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   CUNIT, DASPG, GCN, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST
      REAL       CONST
C                                 Define initial data
      IDO   = 1
      T     = 0.0
      TEND  = CONST(’pi’)
      DELT  = 0.1
      NSTEP = TEND/DELT
      CALL UMACH (2, NOUT)
C                                 Get initial conditions
      CALL GCN (0, T, Y, YPR, GVAL)
      ISTEP  = 0
      MAXTEN = 0.
   10 CONTINUE
      ISTEP = ISTEP + 1
      CALL DASPG (N, T, T+DELT, IDO, Y, YPR, GCN)
      IF (ISTEP .LE. NSTEP) THEN
C                                 Note max tension value
         IF (ABS(Y(5)) .GT. ABS(MAXTEN)) THEN
            TMAX   = T
            MAXTEN = Y(5)
         END IF
         IF (ISTEP .EQ. NSTEP) IDO = 3
         GO TO 10
      END IF
C                                  Convert to English units
      CALL CUNIT (MAXTEN, ’kg/s**2’, MAXLB, ’lb/s**2’)
C                                  Print maximum tension
      WRITE (NOUT,99999) MAXLB, TMAX
99999 FORMAT (’ Extreme string tension of’, F10.2, ’ (lb/s**2)’,
     &       ’ occurred at ’, ’time ’, F10.2)
      END
C
      SUBROUTINE GCN (N, T, Y, YPR, GVAL)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(*), YPR(*), GVAL(*)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       FEETL, GRAV, LENSQ, MASSKG, MASSLB, METERL, MG
```

```
C                                  SPECIFICATIONS FOR SAVE VARIABLES
      LOGICAL    FIRST
      SAVE       FIRST
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   CUNIT
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST
      REAL       CONST
C
      DATA FIRST/.TRUE./
C
      IF (FIRST) GO TO 20
   10 CONTINUE
C                                 Define initial conditions
      IF (N .EQ. 0) THEN
C                                 The pendulum is horizontal
C                                 with these initial y values
         Y(1)   = METERL
         Y(2)   = 0.
         Y(3)   = 0.
         Y(4)   = 0.
         Y(5)   = 0.
         YPR(1) = 0.
         YPR(2) = 0.
         YPR(3) = 0.
         YPR(4) = 0.
         YPR(5) = 0.
         RETURN
      END IF
C                                 Compute residuals
      GVAL(1) = Y(3) - YPR(1)
      GVAL(2) = Y(4) - YPR(2)
      GVAL(3) = -Y(1)*Y(5) - MASSKG*YPR(3)
      GVAL(4) = -Y(2)*Y(5) - MASSKG*YPR(4) - MG
      GVAL(5) = MASSKG*(Y(3)**2+Y(4)**2) - MG*Y(2) - LENSQ*Y(5)
      RETURN
C                                 Convert from English to
C                                 Metric units:
   20 CONTINUE
      FEETL  = 6.5
      MASSLB = 98.0
C                                 Change to meters
      CALL CUNIT (FEETL, ’ft’, METERL, ’meter’)
C                                 Change to kilograms
      CALL CUNIT (MASSLB, ’lb’, MASSKG, ’kg’)
C                                 Get standard gravity
      GRAV  = CONST(’StandardGravity’)
      MG    = MASSKG*GRAV
      LENSQ = METERL**2
      FIRST = .FALSE.
      GO TO 10
      END
```

Output

```
Extreme string tension of   1457.24 (lb/s**2) occurred at time       2.50
```

Example 3

In this example, we solve a stiff ordinary differential equation (E5) from the test package of Enright and Pryce (1987). The problem is nonlinear with nonreal eigenvalues. It is included as an example because it is a stiff problem, and its partial derivatives are provided in the usersupplied routine with the fixed name `DJSPG`. Users who require a variable routine name for partial derivatives can use the routine `D2SPG`. Providing explicit formulas for partial derivatives is an important consideration for problems where evaluations of the function $g(t, y, y')$ are expensive. Signaling that a derivative matrix is provided requires a call to the Chapter 10 options manager utility, `IUMAG`. In addition, an initial integration step-size is given for this test problem. A signal for this is passed using the options manager routine `IUMAG`. The error tolerance is changed from the defaults to a pure absolute tolerance of 0.1 `*` `SQRT`(`AMACH`(4)). Also see `IUMAG` (page 1178), `SUMAG`, page 1175, and `DUMAG`, page 1178 for further details about the options manager routines.

```
      INTEGER    N
      PARAMETER  (N=4)
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    ICHAP, IGET, INUM, IPUT, IRNUM
      PARAMETER  (ICHAP=5, IGET=1, INUM=6, IPUT=2, IRNUM=7)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IDO, IN(50), INR(20), IOPT(2), IVAL(2), NOUT
      REAL       C0, SVAL(3), T, TEND, Y(N), YPR(N)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   DASPG, IUMAG, SUMAG, UMACH
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   GCN
C                                  Define initial data
      IDO = 1
      T = 0.0
      TEND = 1000.0
C                                  Initial values
      C0 = 1.76E-3
      Y(1) = C0
      Y(2) = 0.
      Y(3) = 0.
      Y(4) = 0.
C                                  Initial derivatives
      YPR(1) = 0.
      YPR(2) = 0.
      YPR(3) = 0.
      YPR(4) = 0.
C                                  Get option numbers
      IOPT(1) = INUM
      CALL IUMAG (’math’, ICHAP, IGET, 1, IOPT, IN)
      IOPT(1) = IRNUM
      CALL IUMAG (’math’, ICHAP, IGET, 1, IOPT, INR)
C                                  Provide initial step
      IOPT(1) = INR(6)
      SVAL(1) = 5.0E-5
C                                  Provide absolute tolerance
      IOPT(2) = INR(5)
      SVAL(2) = 0.1*SQRT(AMACH(4))
      SVAL(3) = 0.0
      CALL SUMAG (’math’, ICHAP, IPUT, 2, IOPT, SVAL)
C                                  Using derivatives and
      IOPT(1) = IN(5)
      IVAL(1) = 1
C                                  providing initial step
      IOPT(2) = IN(8)
      IVAL(2) = 1

      CALL IUMAG (’math’, ICHAP, IPUT, 2, IOPT, IVAL)
C                                  Write title
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
C                                  Integrate ODE/DAE
      CALL DASPG (N, T, TEND, IDO, Y, YPR, GCN)
      WRITE (NOUT,99999) T, Y, YPR
C                                  Reset floating options
C                                  to defaults
      IOPT(1) = -INR(5)
      IOPT(2) = -INR(6)
C
      CALL SUMAG (’math’, ICHAP, IPUT, 2, IOPT, SVAL)
C                                  Reset integer options
C                                  to defaults
      IOPT(1) = -IN(5)
      IOPT(2) = -IN(8)
C
      CALL IUMAG (’math’, ICHAP, IPUT, 2, IOPT, IVAL)

99998 FORMAT (11X, ’T’, 14X, ’Y followed by Y’’’)
99999 FORMAT (F15.5/(4F15.5))
      END
C
      SUBROUTINE GCN (N, T, Y, YPR, GVAL)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N
      REAL       T, Y(N), YPR(N), GVAL(N)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       C1, C2, C3, C4
C
      C1 = 7.89E-10
      C2 = 1.1E7
      C3 = 1.13E9
      C4 = 1.13E3
C
      GVAL(1) = -C1*Y(1) - C2*Y(1)*Y(3) - YPR(1)
      GVAL(2) = C1*Y(1) - C3*Y(2)*Y(3) - YPR(2)
      GVAL(3) = C1*Y(1) - C2*Y(1)*Y(3) + C4*Y(4) - C3*Y(2)*Y(3) -
     &          YPR(3)
      GVAL(4) = C2*Y(1)*Y(3) - C4*Y(4) - YPR(4)
      RETURN
      END
      SUBROUTINE DJSPG (N, T, Y, YPR, CJ, PDG, LDPDG)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    N, LDPDG
      REAL       T, CJ, Y(N), YPR(N), PDG(LDPDG,N)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      REAL       C1, C2, C3, C4
```

```
C
      C1 = 7.89E-10
      C2 = 1.1E7
      C3 = 1.13E9
      C4 = 1.13E3
C
      PDG(1,1) = -C1 - C2*Y(3) - CJ
      PDG(1,3) = -C2*Y(1)
      PDG(2,1) = C1
      PDG(2,2) = -C3*Y(3) - CJ
      PDG(2,3) = -C3*Y(2)
      PDG(3,1) = C1 - C2*Y(3)
      PDG(3,2) = -C3*Y(3)
      PDG(3,3) = -C2*Y(1) - C3*Y(2) - CJ
      PDG(3,4) = C4
      PDG(4,1) = C2*Y(3)
      PDG(4,3) = C2*Y(1)
      PDG(4,4) = -C4 - CJ
      RETURN
      END
```

Output

```
      T              Y followed by Y′
1000.00000
   0.00162        0.00000        0.00000        0.00000
   0.00000        0.00000        0.00000        0.00000
```

Example 4

In this final example, we compute the solution of $n = 10$ ordinary differential equations, $g = Hy - y'$, where $y(0) = y_0 = (1, 1, \ldots, 1)^T$. The value

$$\sum_{i=1}^{n} y_i(t)$$

is evaluated at $t = 1$. The constant matrix H has entries $h_{i,j} = \min(j - i, 0)$ so it is lower Hessenberg. We use reverse communication for the evaluation of the following intermediate quantities:

1. The function g,

2. The partial derivative matrix $A = \partial g/\partial y + c_j \partial g/\partial y' = H - c_j I$,

3. The solution of the linear system $A\Delta y = \Delta g$.

In addition to the use of reverse communication, we evaluate the partial derivatives using formulas. No storage is allocated in the floating-point work array for the matrix. Instead, the matrix A is stored in an array `A` within the main program unit. Signals for this organization are passed using the routine `IUMAG` (page 1178).

An algorithm appropriate for this matrix, Givens transformations applied from the right side, is used to factor the matrix A. The rotations are reconstructed during the solve step. See `SROTG` (page 1045) for the formulas.

The routine `D2SPG` stores the value of c_j. We get it with a call to the options manager routine `SUMAG` (page 1175). A pointer, or offset into the work array, is obtained as an integer option. This gives the location of g and Δg. The solution vector Δy replaces Δg at that location. *Caution*: If a user writes code wherein g is computed with reverse communication and partials are evaluated with divided differences, then there will be *two* distinct places where g is to be stored. This example shows a correct place to get this offset.

This example also serves as a prototype for large, structured (possibly nonlinear) DAE problems where the user must use special methods to store and factor the matrix A and solve the linear system $A\Delta y = \Delta g$. The word “factor” is used literally here. A user could, for instance, solve the system using an iterative method. Generally, the factor step can be any preparatory phase required for a later solve step.

```
      INTEGER    N
      PARAMETER  (N=10)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    ICHAP, IGET, INUM, IPUT, IRNUM
      PARAMETER  (ICHAP=5, IGET=1, INUM=6, IPUT=2, IRNUM=7)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IDO, IN(50), INR(20), IOPT(6), IVAL(7), IWK(35+N),
     &           J, NOUT
      REAL       A(N,N), GVAL(N), H(N,N), SC, SS, SUMY, SVAL(1), T,
     &           TEND, WK(41+11*N), Y(N), YPR(N), Z
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  ABS, SQRT
      REAL       ABS, SQRT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   D2SPG, IUMAG, SAXPY, SCOPY, SGEMV, SROT, SROTG, SSET,
     &           SUMAG, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   DGSPG, DJSPG
C                                 Define initial data
      IDO = 1
      T = 0.0E0
      TEND = 1.0E0
C                                 Initial values
      CALL SSET (N, 1.0E0, Y, 1)
      CALL SSET (N, 0.0, YPR, 1)
C                                 Initial lower Hessenberg matrix
      CALL SSET (N*N, 0.0E0, H, 1)
      DO 20  I=1, N - 1
         DO 10  J=1, I + 1
            H(I,J) = J - I
   10    CONTINUE
   20 CONTINUE
      DO 30  J=1, N
         H(N,J) = J - N
   30 CONTINUE
C                                 Get integer option numbers
      IOPT(1) = INUM
      CALL IUMAG ('math', ICHAP, IGET, 1, IOPT, IN)
C                                 Get floating point option numbers
      IOPT(1) = IRNUM
      CALL IUMAG ('math', ICHAP, IGET, 1, IOPT, INR)
C                                 Set for reverse communication
C                                 evaluation of g.
      IOPT(1) = IN(26)
      IVAL(1) = 0
C                                 Set for evaluation of partial
C                                 derivatives.
      IOPT(2) = IN(5)
      IVAL(2) = 1
C                                 Set for reverse communication
C                                 evaluation of partials.
      IOPT(3) = IN(29)
      IVAL(3) = 0
C                                 Set for reverse communication
C                                 solution of linear equations.
      IOPT(4) = IN(31)
      IVAL(4) = 0
C                                 Storage for the partial
C                                 derivative array not allocated.
      IOPT(5) = IN(34)
      IVAL(5) = 1
C                                 Set the sizes of IWK, WK
C                                 for internal checking.
      IOPT(6) = IN(35)
      IVAL(6) = 35 + N
      IVAL(7) = 41 + 11*N
C                                 ’Put’ integer options.
      CALL IUMAG (’math’, ICHAP, IPUT, 6, IOPT, IVAL)
C                                 Write problem title.
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
C                                 Integrate ODE/DAE.  Use
C                                 dummy IMSL external names.
   40 CONTINUE
      CALL D2SPG (N, T, TEND, IDO, Y, YPR, DGSPG, DJSPG, IWK, WK)
C                                 Find where g goes.
C                                 (It only goes in one place
C                                 here, but can vary if
C                                 divided differences are used
C                                 for partial derivatives.)
      IOPT(1) = IN(27)
      CALL IUMAG (’math’, ICHAP, IGET, 1, IOPT, IVAL)
C                                 Direct user response.
      GO TO (50, 180, 60, 50, 90, 100, 130, 150), IDO
   50 CONTINUE
C                                 This should not occur.
      WRITE (NOUT,*) ’ Unexpected return with IDO = ’, IDO
   60 CONTINUE
C                                 Reset options to defaults
      DO 70  I=1, 50
         IN(I) = -IN(I)
   70 CONTINUE
      CALL IUMAG (’math’, ICHAP, IPUT, 50, IN, IVAL)
      DO 80  I=1, 20
         INR(I) = -INR(I)
   80 CONTINUE
      CALL SUMAG (’math’, ICHAP, IPUT, 20, INR, SVAL)
      STOP
   90 CONTINUE
C                                 Return came for g evaluation.
      CALL SCOPY (N, YPR, 1, GVAL, 1)
      CALL SGEMV (’NO’, N, N, 1.0E0, H, N, Y, 1, -1.0E0, GVAL, 1)
C                                 Put g into place.
```

```
      CALL SCOPY (N, GVAL, 1, WK(IVAL(1)), 1)
      GO TO 40
  100 CONTINUE
C                                  Return came for partial
C                                  derivative evaluation.
  110 CALL SCOPY (N*N, H, 1, A, 1)
C                                  Get value of c_j for partials.
      IOPT(1) = INR(9)
      CALL SUMAG (’math’, ICHAP, IGET, 1, IOPT, SVAL)
C                                  Subtract c_j from diagonals
C                                  to compute (partials for y’)*c_j.
      DO 120  I=1, N
         A(I,I) = A(I,I) - SVAL(1)
  120 CONTINUE
      GO TO 40
  130 CONTINUE
C                                  Return came for factorization
      DO 140  J=1, N - 1
C                                  Construct and apply Givens
C                                  transformations.
         CALL SROTG (A(J,J), A(J,J+1), SC, SS)
         CALL SROT (N-J, A(J+1,1), 1, A(J+1,J+1), 1, SC, SS)
  140 CONTINUE
      GO TO 40
  150 CONTINUE
C                                  Return came to solve the system
      CALL SCOPY (N, WK(IVAL(1)), 1, GVAL, 1)
      DO 160  J=1, N - 1
         GVAL(J) = GVAL(J)/A(J,J)
         CALL SAXPY (N-J, -GVAL(J), A(J+1,J), 1, GVAL(J+1), 1)
  160 CONTINUE
      GVAL(N) = GVAL(N)/A(N,N)
C                                  Reconstruct Givens rotations
      DO 170  J=N - 1, 1, -1
         Z = A(J,J+1)
         IF (ABS(Z) .LT. 1.0E0) THEN
            SC = SQRT(1.0E0-Z**2)
            SS = Z
         ELSE IF (ABS(Z) .GT. 1.0E0) THEN
            SC = 1.0E0/Z
            SS = SQRT(1.0E0-SC**2)
         ELSE
            SC = 0.0E0
            SS = 1.0E0
         END IF
         CALL SROT (1, GVAL(J), 1, GVAL(J+1), 1, SC, SS)
  170 CONTINUE
      CALL SCOPY (N, GVAL, 1, WK(IVAL(1)), 1)
      GO TO 40
C
  180 CONTINUE
      SUMY = 0.E0
      DO 190  I=1, N
         SUMY = SUMY + Y(I)
  190 CONTINUE
      WRITE (NOUT,99999) TEND, SUMY
C                                  Finish up internally
      IDO = 3
      GO TO 40

99998 FORMAT (11X, ’T’, 6X, ’Sum of Y(i), i=1,n’)
99999 FORMAT (2F15.5)
      END
```

Output

```
   T      Sum of Y(i), i=1,n
1.00000       65.17058
```

MOLCH/DMOLCH (Single/Double precision)

Solve a system of partial differential equations of the form $u_t = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.

Usage

```
CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, XBREAK,
            TOL, HINIT, Y, LDY)
```

Arguments

IDO — Flag indicating the state of the computation. (Input/Output)

IDO	**State**
1	Initial entry
2	Normal reentry
3	Final call, release workspace

Normally, the initial call is made with IDO = 1. The routine then sets IDO = 2, and this value is then used for all but the last call that is made with IDO = 3.

FCNUT — User-supplied SUBROUTINE to evaluate the function u_t. The usage is CALL FCNUT (NPDES, X, T, U, UX, UXX, UT), where

NPDES – Number of equations. (Input)
X – Space variable, *x*. (Input)
T – Time variable, *t*. (Input)
U – Array of length NPDES containing the dependent variable values, *u*. (Input)
UX – Array of length NPDES containing the first derivatives $u_{x.}$ (Input)
UXX – Array of length NPDES containing the second derivative u_{xx}. (Input)
UT – Array of length NPDES containing the computed derivatives, u_t. (Output)

The name FCNUT must be declared EXTERNAL in the calling program.

FCNBC — User-supplied SUBROUTINE to evaluate the boundary conditions. The boundary conditions accepted by MOLCH are $\alpha_k u_k + \beta_k u_x \equiv \gamma_k$. Note: Users must supply the values α_k and β_k, which determine the values γ_k. Since the γ_k

can depend on t, values of γ'_k are also required. Users must supply these values. The usage is `CALL FCNBC (NPDES, X, T, ALPHA, BETA, GAMMAP)`, where

> `NPDES` – Number of equations. (Input)
> `X` – Space variable, x. This value directs which boundary condition to compute. (Input)
> `T` – Time variable, t. (Input)
> `ALPHA` – Array of length `NPDES` containing the α_k values. (Output)
> `BETA` – Array of length `NPDES` containing the β_k values. (Output)
> `GAMMAP` – Array of length `NPDES` containing the values of the derivatives,
>
> $$\frac{d\gamma_k}{dt} = \gamma'_k$$
>
> (Output)

The name `FCNBC` must be declared `EXTERNAL` in the calling program.

NPDES — Number of differential equations. (Input)

T — Independent variable, t. (Input/Output)
On input, `T` supplies the initial time, t_0. On output, `T` is set to the value to which the integration has been updated. Normally, this new value is `TEND`.

TEND — Value of $t = tend$ at which the solution is desired. (Input)

NX — Number of mesh points or lines. (Input)

XBREAK — Array of length `NX` containing the break points for the cubic Hermite splines used in the x discretization. (Input)
The points in the array `XBREAK` must be strictly increasing. The values `XBREAK`(1) and `XBREAK(NX)` are the endpoints of the interval.

TOL — Differential equation error tolerance. (Input)
An attempt is made to control the local error in such a way that the global relative error is proportional to `TOL`.

HINIT — Initial step size in the t integration. (Input)
This value must be nonnegative. If `HINIT` is zero, an initial step size of $0.001|tend - t_0|$ will be arbitrarily used. The step will be applied in the direction of integration.

Y — Array of size `NPDES` by `NX` containing the solution. (Input/Output)
The array `Y` contains the solution as `Y`(k, i) = $u_k(x, tend)$ at x = `XBREAK`(i). On input, `Y` contains the initial values. It ***MUST*** satisfy the boundary conditions. On output, `Y` contains the computed solution.
There is an optional application of MOLCH that uses derivative values, $u_x(x, t_0)$. The user allocates twice the space for `Y` to pass this information. The optional derivative information is input as

$$\mathrm{Y}(\mathrm{k}, \mathrm{i}+\mathrm{NX})=\frac{\partial u_k}{\partial x}(x, t_0)$$

at x = X(i). The array Y contains the optional derivative values as output:

$$\mathrm{Y}(\mathrm{k}, \mathrm{i}+\mathrm{NX})=\frac{\partial u_k}{\partial x}(x, tend)$$

at x = X(i). To signal that this information is provided, use an options manager call as outlined in Comment 3 and illustrated in Examples 3 and 4.

LDY — Leading dimension of Y exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   MOLCH 2NX * NPDES(12 * NPDES^2 + 21 * NPDES + 10)

   DMOLCH 2NX * NPDES(24 * NPDES^2 + 42 * NPDES + 19)

   Workspace may be explicitly provided, if desired, by use of M2LCH/DM2LCH. The reference is

   ```
   CALL M2LCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX,
               XBREAK, TOL, HINIT, Y, LDY, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work array of length 2NX * NPDES(12 * NPDES^2 + 21 * NPDES + 9). WK should not be changed between calls to M2LCH.

   IWK — Work array of length 2NX * NPDES. IWK should not be changed between calls to M2LCH.

2. Informational errors

Type	Code	
4	1	After some initial success, the integration was halted by repeated error test failures.
4	2	On the next step, X + H will equal X. Either TOL is too small or the problem is stiff.
4	3	After some initial success, the integration was halted by a test on TOL.
4	4	Integration was halted after failing to pass the error test even after reducing the step size by a factor of 1.0E + 10. TOL may be too small.
4	5	Integration was halted after failing to achieve corrector convergence even after reducing the step size by a factor of 1.0E + 10. TOL may be too small.

3. Optional usage with Chapter 10 Option Manager

   11 This option consists of the parameter PARAM, an array with 50 components. See IVPAG (page 646) for a more complete documentation of the contents of this array. To reset this option, use the subprogram SUMAG (page 1175) for single precision, and DUMAG (page 1178) for double precision. The entry PARAM(1) is assigned the initial step, HINIT. The entries PARAM(15) and PARAM(16) are assigned the values equal to the number of lower and upper diagonals that will occur in the Newton method for solving the BDF corrector equations. The value PARAM(17) = 1 is used to signal that the *x* derivatives of the initial data are provided in the the array Y. The output values PARAM(31)-PARAM(36) , showing technical data about the ODE integration, are available with another option manager subroutine call. This call is made after the storage for MOLCH is released. The default values for the first 20 entries of PARAM are (0, 0, amach(2), 500., 0., 5., 0, 0, 1., 3., 1., 2., 2., 1., amach(6), amach(6), 0, sqrt(amach(4)), 1., 0.). Entries 21−50 are defaulted to amach(6).

Algorithm

Let M = NPDES, N = NX and x_i = XBREAK(I). The routine MOLCH uses the method of lines to solve the partial differential equation system

$$\frac{\partial u_k}{\partial t} = f_k\left(x, t, u_1, \ldots u_M, \frac{\partial u_1}{\partial x}, \ldots \frac{\partial u_M}{\partial x}, \frac{\partial^2 u_1}{\partial x^2}, \ldots \frac{\partial^2 u_M}{\partial x^2}\right)$$

with the initial conditions

$$u_k = u_k(x, t) \qquad \text{at } t = t_0$$

and the boundary conditions

$$\alpha_k u_k + \beta_k \frac{\partial u_k}{\partial x} = \gamma k \qquad \text{at } x = x_1 \text{ and at } x = x_N$$

for $k = 1, \ldots, M$.

Cubic Hermite polynomials are used in the x variable approximation so that the trial solution is expanded in the series

$$\hat{u}_k(x, t) = \sum_{i=1}^{M}\left(a_{i,k}(t)\phi_i(x) + b_{i,k}(t)\psi_i(x)\right)$$

where $\phi_i(x)$ and $\psi_i(x)$ are the standard basis functions for the cubic Hermite polynomials with the knots $x_1 < x_2 < \ldots < x_N$. These are piecewise cubic polynomials with continuous first derivatives. At the breakpoints, they satisfy

$$\phi_i(x_l) = \delta_{il} \qquad \psi_i(x_l) = 0$$
$$\frac{d\phi_i}{dx}(x_l) = 0 \qquad \frac{d\psi_i}{dx}(x_l) = \delta_{il}$$

According to the collocation method, the coefficients of the approximation are obtained so that the trial solution satisfies the differential equation at the two Gaussian points in each subinterval,

$$p_{2j-1} = x_j + \frac{3-\sqrt{3}}{6}\left(x_{j+1} - x_j\right)$$

$$p_{2j} \;\; = x_j + \frac{3-\sqrt{3}}{6}\left(x_{j+1} + x_j\right)$$

for $j = 1, \ldots, N$. The collocation approximation to the differential equation is

$$\frac{da_{i,k}}{dt}\phi_i\left(p_j\right) + \frac{db_{i,k}}{dt}\psi_i\left(p_j\right) = f_k\left(p_j, t, \hat{u}_1\left(p_j\right), \ldots, \hat{u}_M\left(p_j\right), \ldots, \left(\hat{u}_1\right)_{xx}\left(p_j\right), \ldots, \left(\hat{u}_M\right)_{xx}\left(p_j\right)\right)$$

for $k = 1, \ldots, M$ and $j = 1, \ldots, 2(N - 1)$.

This is a system of $2M(N - 1)$ ordinary differential equations in $2M\,N$ unknown coefficient functions, $a_{i,k}$ and $b_{i,k}$. This system can be written in the matrix–vector form as $A\, dc/dt = F\,(t, y)$ with $c(t_0) = c_0$ where c is a vector of coefficients of length $2M\,N$ and c_0 holds the initial values of the coefficients. The last $2M$ equations are obtained by differentiating the boundary conditions

$$\alpha_k \frac{da_k}{dt} + \beta_k \frac{db_k}{dt} = \frac{d\gamma_k}{dt}$$

for $k = 1, \ldots, M$.

The initial conditions $u_k(x, t_0)$ must satisfy the boundary conditions. Also, the $\gamma_k(t)$ must be continuous and have a smooth derivative, or the boundary conditions will not be properly imposed for $t > t_0$.

If $\alpha_k = \beta_k = 0$, it is assumed that no boundary condition is desired for the k-th unknown at the left endpoint. A similar comment holds for the right endpoint. Thus, collocation is done at the endpoint. This is generally a useful feature for systems of first-order partial differential equations.

If the number of partial differential equations is $M = 1$ and the number of breakpoints is $N = 4$, then

$$A=\begin{bmatrix}
\alpha_1 & \beta_1 & & & & & & \\
\phi_1(p_1) & \psi_1(p_1) & \phi_2(p_1) & \psi_2(p_1) & & & & \\
\phi_1(p_2) & \psi_1(p_2) & \phi_2(p_2) & \psi_2(p_2) & & & & \\
 & & \phi_3(p_3) & \psi_3(p_3) & \phi_4(p_3) & \psi_4(p_3) & & \\
 & & \phi_3(p_4) & \psi_3(p_4) & \phi_4(p_4) & \psi_4(p_4) & & \\
 & & & & \phi_5(p_5) & \psi_5(p_5) & \phi_6(p_5) & \psi_6(p_5) \\
 & & & & \phi_5(p_6) & \psi_5(p_6) & \phi_6(p_6) & \psi_6(p_6) \\
 & & & & & & \alpha_4 & \beta_4
\end{bmatrix}$$

The vector c is

$$c = [a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4]^T$$

and the right-side F is

$$F=\left[\gamma'(x_1), f(p_1), f(p_2), f(p_3), f(p_4), f(p_5), f(p_6), \gamma'(x_4)\right]^T$$

If $M > 1$, then each entry in the above matrix is replaced by an $M \times M$ diagonal matrix. The element α_1 is replaced by diag($\alpha_{1,1}, \ldots, \alpha_{1,M}$). The elements α_N, β_1 and β_N are handled in the same manner. The $\phi_i(p_j)$ and $\psi_i(p_j)$ elements are replaced by $\phi_i(p_j)I_M$ and $\psi_i(p_j)I_M$ where I_M is the identity matrix of order M. See Madsen and Sincovec (1979) for further details about discretization errors and Jacobian matrix structure.

The input/output array `Y` contains the values of the $a_{k,i}$. The initial values of the $b_{k,i}$ are obtained by using the IMSL cubic spline routine `CSINT` (page 423) to construct functions

$$\hat{u}_k(x, t_0)$$

such that

$$\hat{u}_k(x_i, t_0) = a_{ki}$$

The IMSL routine `CSDER`, page 441, is used to approximate the values

$$\frac{d\hat{U}_k}{dx}(x_i, t_0) \equiv b_{k,i}$$

There is an optional usage of `MOLCH` that allows the user to provide the initial values of $b_{k,i}$.

The order of matrix A is $2M\,N$ and its maximum bandwidth is $6M - 1$. The band structure of the Jacobian of F with respect to c is the same as the band structure of A. This system is solved using a modified version of IVPAG, page 661. Some of the linear solvers were removed. Numerical Jacobians are used exclusively. The algorithm is unchanged. Gear's BDF method is used as the default because the system is typically stiff.

We now present four examples of PDEs that illustrate how users can interface their problems with IMSL PDE solving software. The examples are small and not indicative of the complexities that most practitioners will face in their applications. A set of seven sample application problems, some of them with more than one equation, is given in Sincovec and Madsen (1975). Two further examples are given in Madsen and Sincovec (1979).

Example 1

The normalized linear diffusion PDE, $u_t = u_{xx}$, $0 \le x \le 1$, $t > t_0$, is solved. The initial values are $t_0 = 0$, $u(x, t_0) = u_0 = 1$. There is a "zero-flux" boundary condition at $x = 1$, namely $u_x(1, t) = 0$, $(t > t_0)$. The boundary value of $u(0, t)$ is abruptly changed from u_0 to the value $u_1 = 0.1$. This transition is completed by $t = t_\delta = 0.09$.

Due to restrictions in the type of boundary conditions sucessfully processed by MOLCH, it is necessary to provide the derivative boundary value function γ' at $x = 0$ and at $x = 1$. The function γ at $x = 0$ makes a smooth transition from the value u_0 at $t = t_0$ to the value u_1 at $t = t_\delta$. We compute the transition phase for γ' by evaluating a cubic interpolating polynomial. For this purpose, the function subprogram CSDER, page 441, is used. The interpolation is performed as a first step in the user-supplied routine FCNBC. The function and derivative values $\gamma(t_0) = u_0$, $\gamma'(t_0) = 0$, $\gamma(t_\delta) = u_1$, and $\gamma'(t_\delta) = 0$, are used as input to routine C2HER, to obtain the coefficients evaluated by CSDER. Notice that $\gamma'(t) = 0$, $t > t_\delta$. The evaluation routine CSDER will not yield this value so logic in the routine FCNBC assigns $\gamma'(t) = 0$, $t > t_\delta$.

```
C                                SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    LDY, NPDES, NX
      PARAMETER  (NPDES=1, NX=8, LDY=NPDES)
C                                SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IDO, J, NOUT, NSTEP
      REAL       HINIT, T, TEND, TOL, XBREAK(NX), Y(LDY,NX)
      CHARACTER  TITLE*19
C                                SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  FLOAT
      REAL       FLOAT
C                                SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   MOLCH, UMACH, WRRRN
C                                SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCNBC, FCNUT
      REAL       FCNBC, FCNUT
C                                Set breakpoints and initial
C                                conditions
```

```
      U0 = 1.0
      DO 10  I=1, NX
         XBREAK(I) = FLOAT(I-1)/(NX-1)
         Y(1,I)    = U0
   10 CONTINUE
C                                Set parameters for MOLCH
      TOL   = SQRT(AMACH(4))
      HINIT = 0.01*TOL
      T     = 0.0
      IDO   = 1
      NSTEP = 10
      CALL UMACH (2, NOUT)
      J = 0
   20 CONTINUE
      J    = J + 1
      TEND = FLOAT(J)/FLOAT(NSTEP)
C                                This puts more output for small
C                                t values where action is fastest.
      TEND = TEND**2
C                                Solve the problem
      CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, XBREAK, TOL,
     &            HINIT, Y, LDY)
      IF (J .LE. NSTEP) THEN
C                                Print results
         WRITE (TITLE,'(A,F4.2)') 'Solution at T =', T
         CALL WRRRN (TITLE, NPDES, NX, Y, LDY, 0)
C                                Final call to release workspace
         IF (J .EQ. NSTEP) IDO = 3
         GO TO 20
      END IF
      END
      SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
C                                SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, U(*), UX(*), UXX(*), UT(*)
C
C                                Define the PDE
      UT(1) = UXX(1)
      RETURN
      END
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
C                                SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, ALPHA(*), BETA(*), GAMP(*)
C                                SPECIFICATIONS FOR PARAMETERS
      REAL       TDELTA, U0, U1
      PARAMETER  (TDELTA=0.09, U0=1.0, U1=0.1)
C                                SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IWK(2), NDATA
      REAL       DFDATA(2), FDATA(2), XDATA(2)
C                                SPECIFICATIONS FOR SAVE VARIABLES
      REAL       BREAK(2), CSCOEF(4,2)
      LOGICAL    FIRST
      SAVE       BREAK, CSCOEF, FIRST
C                                SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   C2HER, WRRRN
C                                SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CSDER
      REAL       CSDER
```

```
C
      DATA FIRST/.TRUE./
C
      IF (FIRST) GO TO 20
   10 CONTINUE
C
C
C                                Define the boundary conditions
      IF (X .EQ. 0.0) THEN
C                                These are for x=0.
         ALPHA(1) = 1.0
         BETA(1)  = 0.0
         GAMP(1)  = 0.
C                                If in the boundary layer,
C                                compute nonzero gamma prime.
         IF (T .LE. TDELTA) GAMP(1) = CSDER(1,T,1,BREAK,CSCOEF)
      ELSE
C                                These are for x=1.
         ALPHA(1) = 0.0
         BETA(1)  = 1.0
         GAMP(1)  = 0.0
      END IF
      RETURN
   20 CONTINUE
C                                Compute the boundary layer data.
      NDATA     = 2
      XDATA(1)  = 0.0
      XDATA(2)  = TDELTA
      FDATA(1)  = U0
      FDATA(2)  = U1
      DFDATA(1) = 0.0
      DFDATA(2) = 0.0
C                                Do Hermite cubic interpolation.
      CALL C2HER (NDATA, XDATA, FDATA, DFDATA, BREAK, CSCOEF, IWK)
      FIRST = .FALSE.
      GO TO 10
      END
```

Output

```
                    Solution at T =0.01
    1       2       3       4       5       6       7       8
0.969   0.997   1.000   1.000   1.000   1.000   1.000   1.000

                    Solution at T =0.04
    1       2       3       4       5       6       7       8
0.625   0.871   0.963   0.991   0.998   1.000   1.000   1.000

                    Solution at T =0.09
     1         2         3         4         5         6         7         8
0.1000    0.4603    0.7171    0.8673    0.9437    0.9781    0.9917    0.9951

                    Solution at T =0.16
     1         2         3         4         5         6         7         8
0.1000    0.3131    0.5072    0.6682    0.7893    0.8709    0.9168    0.9316

                    Solution at T =0.25
     1         2         3         4         5         6         7         8
0.1000    0.2568    0.4046    0.5355    0.6429    0.7224    0.7710    0.7874
```

```
                    Solution at T =0.36
     1        2        3        4        5        6        7        8
0.1000   0.2176   0.3293   0.4292   0.5126   0.5751   0.6139   0.6270

                    Solution at T =0.49
     1        2        3        4        5        6        7        8
0.1000   0.1852   0.2661   0.3386   0.3992   0.4448   0.4731   0.4827

                    Solution at T =0.64
     1        2        3        4        5        6        7        8
0.1000   0.1588   0.2147   0.2649   0.3067   0.3382   0.3578   0.3644

                    Solution at T =0.81
     1        2        3        4        5        6        7        8
0.1000   0.1387   0.1754   0.2084   0.2360   0.2567   0.2696   0.2739

                    Solution at T =1.00
     1        2        3        4        5        6        7        8
0.1000   0.1242   0.1472   0.1679   0.1851   0.1981   0.2062   0.2089
```

Example 2

Here, we solve Problem C from Sincovec and Madsen (1975). The equation is of diffusion-convection type with discontinuous coefficients. This problem illustrates a simple method for programming the evaluation routine for the derivative, u_t. Note that the weak discontinuities at $x = 0.5$ are not evaluated in the expression for u_t. The problem is defined as

$$u_t = \partial u / \partial t = \partial / \partial x\left(D(x)\partial u / \partial x\right) - v(x)\partial u / \partial x$$

$$x \in [0, 1], t > 0$$

$$D(x) = \begin{cases} 5 & \text{if } 0 \le x < 0.5 \\ 1 & \text{if } 0.5 < x \le 1.0 \end{cases}$$

$$v(x) = \begin{cases} 1000.0 & \text{if } 0 \le x < 0.5 \\ 1 & \text{if } 0.5 < x \le 1.0 \end{cases}$$

$$u(x, 0) = \begin{cases} 1 \text{ if } x = 0 \\ 0 \text{ if } x > 0 \end{cases}$$

$$u(0, t) = 1, \quad u(1, t) = 0$$

```
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    LDY, NPDES, NX
      PARAMETER  (NPDES=1, NX=100, LDY=NPDES)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IDO, J, NOUT, NSTEP
      REAL       HINIT, T, TEND, TOL, XBREAK(NX), Y(LDY,NX)
      CHARACTER  TITLE*19
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  FLOAT
      REAL       FLOAT
C                                 SPECIFICATIONS FOR SUBROUTINES
```

```
      EXTERNAL   MOLCH, UMACH, WRRRN
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCNBC, FCNUT
      REAL       FCNBC, FCNUT
C                                 Set breakpoints and initial
C                                 conditions
      U0 = 1.0
      DO 10  I=1, NX
         XBREAK(I) = FLOAT(I-1)/(NX-1)
         Y(1,I)    = 0.
   10 CONTINUE
      Y(1,1) = U0
C                                 Set parameters for MOLCH
      TOL   = SQRT(AMACH(4))
      HINIT = 0.01*TOL
      T     = 0.0
      IDO   = 1
      NSTEP = 10
      CALL UMACH (2, NOUT)
      J = 0
   20 CONTINUE
      J    = J + 1
      TEND = FLOAT(J)/FLOAT(NSTEP)
C                                 Solve the problem
      CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, XBREAK, TOL,
     &            HINIT, Y, LDY)
C                                 Final call to release workspace
         IF (J .EQ. NSTEP) IDO = 3
         IF (J.LE. NSTEP ) GO TO 20
C                                 Print results
         WRITE (TITLE,’(A,F4.2)’) ’Solution at T =’, T
         CALL WRRRN (TITLE, NPDES, NX, Y, LDY, 0)
      END
      SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, U(*), UX(*), UXX(*), UT(*)
C
C                                 Define the PDE
C                                 This is the nonlinear
C                                 diffusion-convection with
C                                 discontinuous coefficients.
      IF (X .LE. 0.5) THEN
         D = 5.0
         V = 1000.0
      ELSE
         D = 1.0
         V = 1.0
      END IF
      UT(1) = D*UXX(1) - V*UX(1)
      RETURN
      END
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, ALPHA(*), BETA(*), GAMP(*)
C                                 SPECIFICATIONS FOR PARAMETERS
      ALPHA(1) = 1.0
      BETA(1)  = 0.0
      GAMP(1)  = 0.0
      RETURN
      END
```

Output

```
                            Solution at T =1.00
     1      2      3       4       5       6       7       8       9      10
 1.000  1.000  1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000

    11     12     13      14      15      16      17      18      19      20
 1.000  1.000  1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000

    21     22     23      24      25      26      27      28      29      30
 1.000  1.000  1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000

    31     32     33      34      35      36      37      38      39      40
 1.000  1.000  1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000

    41     42     43      44      45      46      47      48      49      50
 1.000  1.000  1.000   1.000   1.000   1.000   1.000   1.000   1.000   0.997

    51     52     53      54      55      56      57      58      59      60
 0.984  0.969  0.953   0.937   0.921   0.905   0.888   0.872   0.855   0.838

    61     62     63      64      65      66      67      68      69      70
 0.821  0.804  0.786   0.769   0.751   0.733   0.715   0.696   0.678   0.659

    71     72     73      74      75      76      77      78      79      80
 0.640  0.621  0.602   0.582   0.563   0.543   0.523   0.502   0.482   0.461

    81     82     83      84      85      86      87      88      89      90
 0.440  0.419  0.398   0.376   0.354   0.332   0.310   0.288   0.265   0.242

    91     92     93      94      95      96      97      98      99     100
 0.219  0.196  0.172   0.148   0.124   0.100   0.075   0.050   0.025   0.000
```

Example 3

In this example, using `MOLCH`, we solve the linear normalized diffusion PDE $u_t = u_{xx}$ but with an optional usage that provides values of the derivatives, u_x, of the initial data. Due to errors in the numerical derivatives computed by spline interpolation, more precise derivative values are required when the initial data is $u(x, 0) = 1 + \cos[(2n - 1)\pi x]$, $n > 1$. The boundary conditions are “zero flux” conditions $u_x(0, t) = u_x(1, t) = 0$ for $t > 0$. Note that the initial data is compatible with these end conditions since the derivative function

$$u_x(x,0) = \frac{du(x,0)}{dx} = -(2n-1)\pi \sin\left[(2n-1)\pi x\right]$$

vanishes at $x = 0$ and $x = 1$.

The example illustrates the use of the IMSL options manager subprograms `SUMAG`, page 1175 or, for double precision, `DUMAG`, page 1178, to reset the array `PARAM` used for control of the specialized version of `IVPAG` that integrates the system of ODEs. This optional usage signals that the derivative of the initial data is passed by the user. The values $u(x, tend)$ and $u_x(x, tend)$ are output at the breakpoints with the optional usage.

```
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    LDY, NPDES, NX
      PARAMETER  (NPDES=1, NX=10, LDY=NPDES)
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    ICHAP, IGET, IPUT, KPARAM
      PARAMETER  (ICHAP=5, IGET=1, IPUT=2, KPARAM=11)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IACT, IDO, IOPT(1), J, JGO, N, NOUT, NSTEP
      REAL       ARG, HINIT, PARAM(50), PI, T, TEND, TOL, XBREAK(NX),
     &           Y(LDY,2*NX)
      CHARACTER  TITLE*36
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  COS, FLOAT, SIN, SQRT
      REAL       COS, FLOAT, SIN, SQRT
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   MOLCH, SUMAG, UMACH, WRRRN
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   AMACH, CONST, FCNBC, FCNUT
      REAL       AMACH, CONST, FCNBC, FCNUT
C                                  Set breakpoints and initial
C                                  conditions.
      N       = 5
      PI      = CONST('pi')
      IOPT(1) = KPARAM
      DO 10  I=1, NX
         XBREAK(I) = FLOAT(I-1)/(NX-1)
         ARG       = (2.*N-1)*PI
C                                  Set function values.
         Y(1,I) = 1. + COS(ARG*XBREAK(I))
C                                  Set first derivative values.
         Y(1,I+NX) = -ARG*SIN(ARG*XBREAK(I))
   10 CONTINUE
C                                  Set parameters for MOLCH
      TOL   = SQRT(AMACH(4))
      HINIT = 0.01*TOL
      T     = 0.0
      IDO   = 1
      NSTEP = 10
      CALL UMACH (2, NOUT)
      J = 0
C                                   Get and reset the PARAM array
C                                   so that user-provided derivatives
C                                   of the initial data are used.
      JGO  = 1
      IACT = IGET
      GO TO 70
   20 CONTINUE
C                                   This flag signals that
C                                   derivatives are passed.
      PARAM(17) = 1.
      JGO       = 2
      IACT      = IPUT
      GO TO 70
   30 CONTINUE
C                                   Look at output at steps
C                                   of 0.001.
```

```
      TEND = 0.
   40 CONTINUE
      J    = J + 1
      TEND = TEND + 0.001
C                                 Solve the problem
      CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, XBREAK, TOL,
     &            HINIT, Y, LDY)
      IF (J .LE. NSTEP) THEN
C                                 Print results
         WRITE (TITLE,’(A,F5.3)’) ’Solution and derivatives at T =’, T
         CALL WRRRN (TITLE, NPDES, 2*NX, Y, LDY, 0)
C                                 Final call to release workspace
         IF (J .EQ. NSTEP) IDO = 3
         GO TO 40
      END IF
C                                 Show, for example, the maximum
C                                 step size used.
      JGO  = 3
      IACT = IGET
      GO TO 70
   50 CONTINUE
      WRITE (NOUT,*) ’ Maximum step size used is:  ’, PARAM(33)
C                                 Reset option to defaults
      JGO     = 4
      IAC     = IPUT
      IOPT(1) = -IOPT(1)
      GO TO 70
   60 CONTINUE
      STOP
C                                 Internal routine to work options
   70 CONTINUE
      CALL SUMAG (’math’, ICHAP, IACT, 1, IOPT, PARAM)
      GO TO (20, 30, 50, 60), JGO
      END
      SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, U(*), UX(*), UXX(*), UT(*)
C
C                                 Define the PDE
      UT(1) = UXX(1)
      RETURN
      END
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
C                                 SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, ALPHA(*), BETA(*), GAMP(*)
C
      ALPHA(1) = 0.0
      BETA(1)  = 1.0
      GAMP(1)  = 0.0
      RETURN
      END
```

Output

```
               Solution and derivatives at T =0.001
    1      2        3      4        5      6        7      8        9     10
1.483  0.517    1.483  0.517    1.483  0.517    1.483  0.517    1.483  0.517

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.002
    1      2        3      4        5      6        7      8        9     10
1.233  0.767    1.233  0.767    1.233  0.767    1.233  0.767    1.233  0.767

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.003
    1      2        3      4        5      6        7      8        9     10
1.113  0.887    1.113  0.887    1.113  0.887    1.113  0.887    1.113  0.887

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.004
    1      2        3      4        5      6        7      8        9     10
1.054  0.946    1.054  0.946    1.054  0.946    1.054  0.946    1.054  0.946

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.005
    1      2        3      4        5      6        7      8        9     10
1.026  0.974    1.026  0.974    1.026  0.974    1.026  0.974    1.026  0.974

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.006
    1      2        3      4        5      6        7      8        9     10
1.012  0.988    1.012  0.988    1.012  0.988    1.012  0.988    1.012  0.988

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.007
    1      2        3      4        5      6        7      8        9     10
1.006  0.994    1.006  0.994    1.006  0.994    1.006  0.994    1.006  0.994

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000

               Solution and derivatives at T =0.008
    1      2        3      4        5      6        7      8        9     10
1.003  0.997    1.003  0.997    1.003  0.997    1.003  0.997    1.003  0.997

   11     12       13     14       15     16       17     18       19     20
0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000    0.000  0.000
```

```
Solution and derivatives at T =0.009
     1      2       3      4       5      6       7      8       9     10
 1.001  0.999   1.001  0.999   1.001  0.999   1.001  0.999   1.001  0.999

    11     12      13     14      15     16      17     18      19     20
 0.000  0.000   0.000  0.000   0.000  0.000   0.000  0.000   0.000  0.000

                Solution and derivatives at T =0.010
     1      2       3      4       5      6       7      8       9     10
 1.001  0.999   1.001  0.999   1.001  0.999   1.001  0.999   1.001  0.999

    11     12      13     14      15     16      17     18      19     20
 0.000  0.000   0.000  0.000   0.000  0.000   0.000  0.000   0.000  0.000
Maximum step size used is:      1.00000E-02
```

Example 4

In this example, we consider the linear normalized hyperbolic PDE, $u_{tt} = u_{xx}$, the “vibrating string” equation. This naturally leads to a system of first order PDEs. Define a new dependent variable $u_t = v$. Then, $v_t = u_{xx}$ is the second equation in the system. We take as initial data $u(x, 0) = \sin(\pi x)$ and $u_t(x, 0) = v(x, 0) = 0$. The ends of the string are fixed so $u(0, t) = u(1, t) = v(0, t) = v(1, t) = 0$. The exact solution to this problem is $u(x, t) = \sin(\pi x) \cos(\pi t)$. Residuals are computed at the output values of t for $0 < t \le 2$. Output is obtained at 200 steps in increments of 0.01.

Even though the sample code `MOLCH` gives satisfactory results for this PDE, users should be aware that for *nonlinear problems*, “shocks” can develop in the solution. The appearance of shocks may cause the code to fail in unpredictable ways. See Courant and Hilbert (1962), pages 488-490, for an introductory discussion of shocks in hyperbolic systems.

```
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    LDY, NPDES, NX
      PARAMETER  (NPDES=2, NX=10, LDY=NPDES)
C                                 SPECIFICATIONS FOR PARAMETERS
      INTEGER    ICHAP, IGET, IPUT, KPARAM
      PARAMETER  (ICHAP=5, IGET=1, IPUT=2, KPARAM=11)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IACT, IDO, IOPT(1), J, JGO, NOUT, NSTEP
      REAL       HINIT, PARAM(50), PI, T, TEND, TOL, XBREAK(NX),
     &           Y(LDY,2*NX), ERROR(NX)
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  COS, FLOAT, SIN, SQRT
      REAL       COS, FLOAT, SIN, SQRT
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   MOLCH, SUMAG, UMACH, WRRRN
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   AMACH, CONST, FCNBC, FCNUT
      REAL       AMACH, CONST, FCNBC, FCNUT
C                                 Set breakpoints and initial
C                                 conditions.
      PI      = CONST('pi')
      IOPT(1) = KPARAM
      DO 10  I=1, NX
         XBREAK(I) = FLOAT(I-1)/(NX-1)
```

```
C                                 Set function values.
         Y(1,I) = SIN(PI*XBREAK(I))
         Y(2,I) = 0.
C                                 Set first derivative values.
         Y(1,I+NX) = PI*COS(PI*XBREAK(I))
         Y(2,I+NX) = 0.0
   10 CONTINUE
C                                 Set parameters for MOLCH
      TOL   = 0.1*SQRT(AMACH(4))
      HINIT = 0.01*TOL
      T     = 0.0
      IDO   = 1
      NSTEP = 200
      CALL UMACH (2, NOUT)
      J = 0
C                                  Get and reset the PARAM array
C                                  so that user-provided derivatives
C                                  of the initial data are used.
      JGO  = 1
      IACT = IGET
      GO TO 90
   20 CONTINUE
C                                  This flag signals that
C                                  derivatives are passed.
      PARAM(17) = 1.
      JGO       = 2
      IACT      = IPUT
      GO TO 90
   30 CONTINUE
C                                  Look at output at steps
C                                  of 0.01 and compute errors.
      ERRU = 0.
      TEND = 0.
   40 CONTINUE
      J    = J + 1
      TEND = TEND + 0.01
C                                 Solve the problem
      CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, XBREAK, TOL,
     &            HINIT, Y, LDY)
      DO 50  I=1, NX
         ERROR(I) = Y(1,I) - SIN(PI*XBREAK(I))*COS(PI*TEND)
   50 CONTINUE
      IF (J .LE. NSTEP) THEN
         DO 60  I=1, NX
            ERRU = AMAX1(ERRU,ABS(ERROR(I)))
   60    CONTINUE
C                                 Final call to release workspace
         IF (J .EQ. NSTEP) IDO = 3
         GO TO 40
      END IF
C                                 Show, for example, the maximum
C                                 step size used.
      JGO  = 3
      IACT = IGET
      GO TO 90
   70 CONTINUE
      WRITE (NOUT,*) ’ Maximum error in u(x,t) divided by TOL: ’,
     &              ERRU/TOL
      WRITE (NOUT,*) ’ Maximum step size used is:  ’, PARAM(33)
```

```
C                                  Reset option to defaults
      JGO     = 4
      IACT    = IPUT
      IOPT(1) = -IOPT(1)
      GO TO 90
   80 CONTINUE
      STOP
C                                  Internal routine to work options
   90 CONTINUE
      CALL SUMAG ('math', ICHAP, IACT, 1, IOPT, PARAM)
      GO TO (20, 30, 70, 80), JGO
      END
      SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, U(*), UX(*), UXX(*), UT(*)
C
C                                  Define the PDE
      UT(1) = U(2)
      UT(2) = UXX(1)
      RETURN
      END
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    NPDES
      REAL       X, T, ALPHA(*), BETA(*), GAMP(*)
C
      ALPHA(1) = 1.0
      BETA(1)  = 0.0
      GAMP(1)  = 0.0
      ALPHA(2) = 1.0
      BETA(2)  = 0.0
      GAMP(2)  = 0.0
      RETURN
      END
```

Output

```
Maximum error in u(x,t) divided by TOL:     1.28094
Maximum step size used is:      9.99999E-02
```

FPS2H/DFPS2H (Single/Double precision)

Solve Poisson’s or Helmholtz’s equation on a two-dimensional rectangle using a fast Poisson solver based on the HODIE finite-difference scheme on a uniform mesh.

Usage

```
CALL FPS2H (PRHS, BRHS, COEFU, NX, NY, AX, BX, AY, BY,
            IBCTY, IORDER, U, LDU)
```

Arguments

PRHS — User-supplied FUNCTION to evaluate the right side of the partial differential equation. The form is PRHS(X, Y), where

X – X-coordinate value. (Input)
Y – Y-coordinate value. (Input)
PRHS – Value of the right side at (X, Y). (Output)

PRHS must be declared EXTERNAL in the calling program.

BRHS — User-supplied FUNCTION to evaluate the right side of the boundary conditions. The form is BRHS(ISIDE, X, Y), where
ISIDE – Side number. (Input)
See IBCTY below for the definition of the side numbers.
X – X-coordinate value. (Input)
Y – Y-coordinate value. (Input)
BRHS – Value of the right side of the boundary condition at (X, Y). (Output)
BRHS must be declared EXTERNAL in the calling program.

COEFU — Value of the coefficient of U in the differential equation. (Input)

NX — Number of grid lines in the X-direction. (Input)
NX must be at least 4. See Comment 2 for further restrictions on NX.

NY — Number of grid lines in the Y-direction. (Input)
NY must be at least 4. See Comment 2 for further restrictions on NY.

AX — The value of X along the left side of the domain. (Input)

BX — The value of X along the right side of the domain. (Input)

AY — The value of Y along the bottom of the domain. (Input)

BY — The value of Y along the top of the domain. (Input)

IBCTY — Array of size 4 indicating the type of boundary condition on each side of the domain or that the solution is periodic. (Input)
The sides are numbered 1 to 4 as follows:

Side	**Location**
1 - Right	(X = BX)
2 - Bottom	(Y = AY)
3 - Left	(X = AX)
4 - Top	(Y = BY)

There are three boundary condition types.

IBCTY	**Boundary Condition**
1	Value of U is given. (Dirichlet)
2	Value of dU/dX is given (sides 1 and/or 3). (Neumann) Value of dU/dY is given (sides 2 and/or 4).
3	Periodic.

IORDER — Order of accuracy of the finite-difference approximation. (Input)
It can be either 2 or 4. Usually, IORDER = 4 is used.

U — Array of size NX by NY containing the solution at the grid points. (Output)

LDU — Leading dimension of `U` exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   `FPS2H` (`NX` + 2)(`NY` + 2) + (`NX` + 1)(`NY` + 1)(`IORDER` – 2)/2 + 6(`NX` + `NY`) + `NX`/2 + 16

   `DFPS2H` 2(`NX` + 2)(`NY` + 2) + (`NX` + 1)(`NY` + 1)(`IORDER` – 2) + 12(`NX` + `NY`) + `NX` + 32

   Workspace may be explicitly provided, if desired, by use of `F2S2H`/`DF2S2H`. The reference is

   ```
   CALL F2S2H (PRHS, BRHS, COEFU, NX, NY, AX, BX, AY,
               BY, IBCTY, IORDER, U, LDU, UWORK, WORK)
   ```

   The additional arguments are as follows:

   UWORK — Work array of size `NX` + 2 by `NY` + 2. If the actual dimensions of `U` are large enough, then `U` and `UWORK` can be the same array.

   WORK — Work array of length (`NX` + 1)(`NY` + 1)(`IORDER` – 2)/2 + 6(`NX` + `NY`) + `NX`/2 + 16.

2. The grid spacing is the distance between the (uniformly spaced) grid lines. It is given by the formulas `HX` = (`BX` – `AX`)/(`NX` – 1) and `HY` = (`BY` – `AY`)/(`NY` – 1). The grid spacings in the `X` and `Y` directions must be the same, i.e., `NX` and `NY` must be such that `HX` equals `HY`. Also, as noted above, `NX` and `NY` must both be at least 4. To increase the speed of the fast Fourier transform, `NX` – 1 should be the product of small primes. Good choices are 17, 33, and 65.

3. If –`COEFU` is nearly equal to an eigenvalue of the Laplacian with homogeneous boundary conditions, then the computed solution might have large errors.

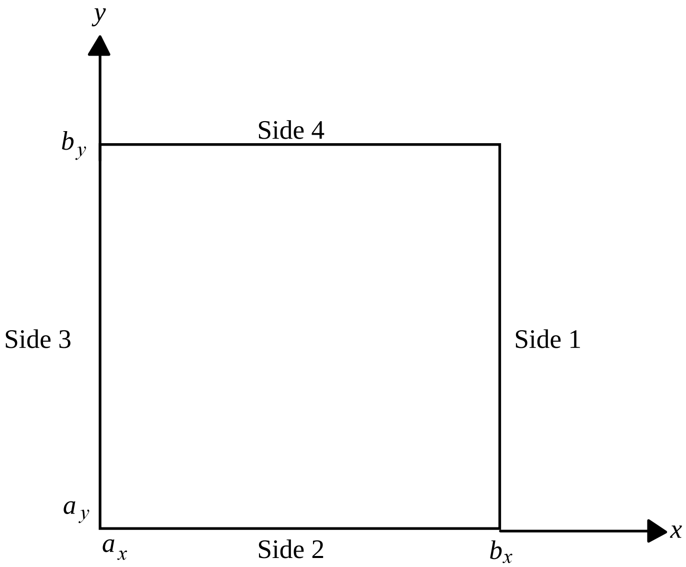
Algorithm

Let c = `COEFU`, a_x = `AX`, b_x = `BX`, a_y = `AY`, b_y = `BY`, n_x = `NX` and n_y = `NY`.

`FPS2H` is based on the code `HFFT2D` by Boisvert (1984). It solves the equation

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+cu=p$$

on the rectangular domain $(a_x, b_x) \times (a_y, b_y)$ with a user-specified combination of Dirichlet (solution prescribed), Neumann (first-derivative prescribed), or periodic boundary conditions. The sides are numbered clockwise, starting with the right side.



When $c = 0$ and only Neumann or periodic boundary conditions are prescribed, then any constant may be added to the solution to obtain another solution to the problem. In this case, the solution of minimum ∞-norm is returned.

The solution is computed using either a second-or fourth-order accurate finite-difference approximation of the continuous equation. The resulting system of linear algebraic equations is solved using fast Fourier transform techniques. The algorithm relies upon the fact that $n_x - 1$ is highly composite (the product of small primes). For details of the algorithm, see Boisvert (1984). If $n_x - 1$ is highly composite then the execution time of `FPS2H` is proportional to $n_x n_y \log_2 n_x$. If evaluations of $p(x, y)$ are inexpensive, then the difference in running time between `IORDER` = 2 and `IORDER` = 4 is small.

Example

In this example, the equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 3u = -2\sin(x + 2y) + 16e^{2x+3y}$$

with the boundary conditions $\partial u/\partial y = 2\cos(x + 2y) + 3\exp(2x + 3y)$ on the bottom side and $u = \sin(x + 2y) + \exp(2x + 3y)$ on the other three sides. The domain is the rectangle[0, 1/4] × [0, 1/2]. The output of `FPS2H` is a 17 × 33 table of U values. The quadratic interpolation routine `QD2VL` is used to print a table of values.

```
      INTEGER    NCVAL, NX, NXTABL, NY, NYTABL
      PARAMETER  (NCVAL=11, NX=17, NXTABL=5, NY=33, NYTABL=5)
C
      INTEGER    I, IBCTY(4), IORDER, J, NOUT
      REAL       AX, AY, BRHS, BX, BY, COEFU, ERROR, FLOAT, PRHS, QD2VL,
     &           TRUE, U(NX,NY), UTABL, X, XDATA(NX), Y, YDATA(NY)
      INTRINSIC  FLOAT
      EXTERNAL   BRHS, FPS2H, PRHS, QD2VL, UMACH
C                                 Set rectangle size
      AX = 0.0
      BX = 0.25
      AY = 0.0
      BY = 0.50
C                                 Set boundary condition types
      IBCTY(1) = 1
      IBCTY(2) = 2
      IBCTY(3) = 1
      IBCTY(4) = 1
C                                 Coefficient of U
      COEFU = 3.0
C                                 Order of the method
      IORDER = 4
C                                 Solve the PDE
      CALL FPS2H (PRHS, BRHS, COEFU, NX, NY, AX, BX, AY, BY, IBCTY,
     &            IORDER, U, NX)
C                                 Setup for quadratic interpolation
      DO 10  I=1, NX
         XDATA(I) = AX + (BX-AX)*FLOAT(I-1)/FLOAT(NX-1)
   10 CONTINUE
      DO 20  J=1, NY
         YDATA(J) = AY + (BY-AY)*FLOAT(J-1)/FLOAT(NY-1)
   20 CONTINUE
C                                 Print the solution
      CALL UMACH (2, NOUT)
      WRITE (NOUT,’(8X,A,11X,A,11X,A,8X,A)’) ’X’, ’Y’, ’U’, ’Error’
      DO 40  J=1, NYTABL
         DO 30  I=1, NXTABL
            X     = AX + (BX-AX)*FLOAT(I-1)/FLOAT(NXTABL-1)
            Y     = AY + (BY-AY)*FLOAT(J-1)/FLOAT(NYTABL-1)
            UTABL = QD2VL(X,Y,NX,XDATA,NY,YDATA,U,NX,.FALSE.)
            TRUE  = SIN(X+2.*Y) + EXP(2.*X+3.*Y)
            ERROR = TRUE - UTABL
            WRITE (NOUT,’(4F12.4)’) X, Y, UTABL, ERROR
   30 CONTINUE
   40 CONTINUE
      END
C
      REAL FUNCTION PRHS (X, Y)
      REAL       X, Y
C
      REAL       EXP, SIN
      INTRINSIC  EXP, SIN
C                                 Define right side of the PDE
      PRHS = -2.*SIN(X+2.*Y) + 16.*EXP(2.*X+3.*Y)
      RETURN
      END
C
      REAL FUNCTION BRHS (ISIDE, X, Y)
      INTEGER    ISIDE
      REAL       X, Y
C
      REAL       COS, EXP, SIN
      INTRINSIC  COS, EXP, SIN
C                                 Define the boundary conditions
```

```
      IF (ISIDE .EQ. 2) THEN
         BRHS = 2.*COS(X+2.*Y) + 3.*EXP(2.*X+3.*Y)
      ELSE
         BRHS = SIN(X+2.*Y) + EXP(2.*X+3.*Y)
      END IF
      RETURN
      END
```

Output

```
   X           Y           U        Error
0.0000      0.0000      1.0000      0.0000
0.0625      0.0000      1.1956      0.0000
0.1250      0.0000      1.4087      0.0000
0.1875      0.0000      1.6414      0.0000
0.2500      0.0000      1.8961      0.0000
0.0000      0.1250      1.7024      0.0000
0.0625      0.1250      1.9562      0.0000
0.1250      0.1250      2.2345      0.0000
0.1875      0.1250      2.5407      0.0000
0.2500      0.1250      2.8783      0.0000
0.0000      0.2500      2.5964      0.0000
0.0625      0.2500      2.9322      0.0000
0.1250      0.2500      3.3034      0.0000
0.1875      0.2500      3.7148      0.0000
0.2500      0.2500      4.1720      0.0000
0.0000      0.3750      3.7619      0.0000
0.0625      0.3750      4.2163      0.0000
0.1250      0.3750      4.7226      0.0000
0.1875      0.3750      5.2878      0.0000
0.2500      0.3750      5.9199      0.0000
0.0000      0.5000      5.3232      0.0000
0.0625      0.5000      5.9520      0.0000
0.1250      0.5000      6.6569      0.0000
0.1875      0.5000      7.4483      0.0000
0.2500      0.5000      8.3380      0.0000
```

FPS3H/DFPS3H (Single/Double precision)

Solve Poisson's or Helmholtz's equation on a three-dimensional box using a fast Poisson solver based on the HODIE finite-difference scheme on a uniform mesh.

Usage

```
CALL FPS3H (PRHS, BRHS, COEFU, NX, NY, NZ, AX, BX, AY, BY,
            AZ, BZ, IBCTY, IORDER, U, LDU, MDU)
```

Arguments

PRHS — User-supplied FUNCTION to evaluate the right side of the partial differential equation. The form is PRHS(X, Y, Z), where

X – The *x*-coordinate value. (Input)
Y – The *y*-coordinate value. (Input)
Z – The *z*-coordinate value. (Input)
PRHS – Value of the right side at (X, Y, Z). (Output)

PRHS must be declared EXTERNAL in the calling program.

BRHS — User-supplied FUNCTION to evaluate the right side of the boundary conditions. The form is BRHS(ISIDE, X, Y, Z), where

ISIDE – Side number. (Input)
See IBCTY for the definition of the side numbers.
X – The *x*-coordinate value. (Input)
Y – The *y*-coordinate value. (Input)
Z – The *z*-coordinate value. (Input)
BRHS – Value of the right side of the boundary condition at (X, Y, Z). (Output)

BRHS must be declared EXTERNAL in the calling program.

COEFU — Value of the coefficient of U in the differential equation. (Input)

NX — Number of grid lines in the *x*-direction. (Input)
NX must be at least 4. See Comment 2 for further restrictions on NX.

NY — Number of grid lines in the *y*-direction. (Input)
NY must be at least 4. See Comment 2 for further restrictions on NY.

NZ — Number of grid lines in the *y*-direction. (Input)
NZ must be at least 4. See Comment 2 for further restrictions on NZ.

AX — Value of X along the left side of the domain. (Input)

BX — Value of X along the right side of the domain. (Input)

AY — Value of Y along the bottom of the domain. (Input)

BY — Value of Y along the top of the domain. (Input)

AZ — Value of Z along the front of the domain. (Input)

BZ — Value of Z along the back of the domain. (Input)

IBCTY — Array of size 6 indicating the type of boundary condition on each face of the domain or that the solution is periodic. (Input)
The sides are numbers 1 to 6 as follows:

Side	**Location**
1 - Right	(X = BX)
2 - Bottom	(Y = AY)
3 - Left	(X = AX)
4 - Top	(Y = BY)
5 - Front	(Z = BZ)
6 - Back	(Z = AZ)

There are three boundary condition types.

IBCTY	Boundary Condition
1	Value of U is given. (Dirichlet)
2	Value of dU/dX is given (sides 1 and/or 3). (Neumann) Value of dU/dY is given (sides 2 and/or 4). Value of dU/dZ is given (sides 5 and/or 6).
3	Periodic.

IORDER — Order of accuracy of the finite-difference approximation.   (Input)
It can be either 2 or 4. Usually, IORDER = 4 is used.

U — Array of size NX by NY by NZ containing the solution at the grid points. (Output)

LDU — Leading dimension of U exactly as specified in the dimension statement of the calling program.   (Input)

MDU — Middle dimension of U exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   FPS3H (NX + 2)(NY + 2)(NZ + 2) + (NX + 1)(NY + 1)(NZ + 1)(IORDER – 2)/2 + 2(NX * NY + NX * NZ + NY * NZ) + 2(NX + NY + 1)+ MAX(2 * NX * NY, 2 * NX + NY + 4 * NZ + (NX + NZ)/2 + 29)

   DFPS3H 2(NX + 2)(NY + 2)(NZ + 2) + (NX + 1(NY + 1)(NZ + 1)(IORDER – 2) + 4(NX * NY + NX * NZ + NY * NZ) + 4(NX + NY + 1) + 2MAX(2 * NX * NY, 2 * NX + NY + 4 * NZ + (NX + NZ)/2 + 29)

   Workspace may be explicitly provided, if desired, by use of F2S3H/DF2S3H. The reference is

   ```
   CALL F2S3H (PRHS, BRHS, COEFU, NX, NY, NZ, AX, BX,
               AY, BY, AZ, BZ, IBCTY, IORDER, U, LDU,
               MDU, UWORK, WORK)
   ```

   The additional arguments are as follows:

   UWORK — Work array of size NX + 2 by NY + 2 by NZ + 2. If the actual dimensions of U are large enough, then U and UWORK can be the same array.

   WORK — Work array of length (NX + 1)(NY + 1)(NZ + 1)(IORDER – 2)/2 + 2(NX * NY + NX * NZ + NY * NZ) + 2(NX + NY + 1) + MAX(2 * NX * NY, 2 * NX + NY + 4 * NZ + (NX + NZ)/2 + 29)

2. The grid spacing is the distance between the (uniformly spaced) grid lines. It is given by the formulas
   HX = (BX – AX)/(NX – 1),
   HY = (BY – AY)/(NY – 1), and
   HZ = (BZ – AZ)/(NZ – 1).
   The grid spacings in the X, Y and Z directions must be the same, i.e.,

NX, NY and NZ must be such that HX = HY = HZ. Also, as noted above, NX, NY and NZ must all be at least 4. To increase the speed of the Fast Fourier transform, NX − 1 and NZ − 1 should be the product of small primes. Good choices for NX and NZ are 17, 33 and 65.

3. If −COEFU is nearly equal to an eigenvalue of the Laplacian with homogeneous boundary conditions, then the computed solution might have large errors.

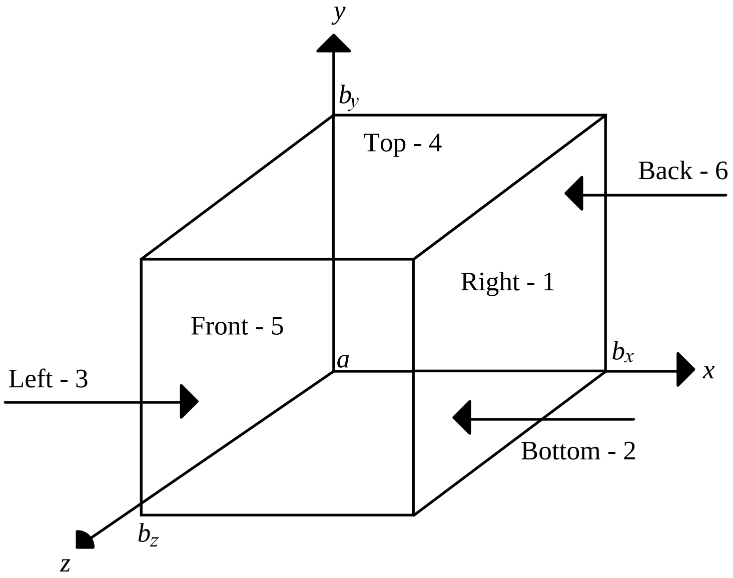
Algorithm

Let c = COEFU, a_x = AX, b_x = BX, n_x = NX, a_y = AY, b_y = BY, n_y = NY, a_z = AZ, b_z = BZ, and n_z = NZ.

FPS3H is based on the code HFFT3D by Boisvert (1984). It solves the equation

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}+cu=p$$

on the domain $(a_x, b_x) \times (a_y, b_y) \times (a_z, b_z)$ (a box) with a user-specified combination of Dirichlet (solution prescribed), Neumann (first derivative prescribed), or periodic boundary conditions. The six sides are numbered as shown in the following diagram.



When c = 0 and only Neumann or periodic boundary conditions are prescribed, then any constant may be added to the solution to obtain another solution to the problem. In this case, the solution of minimum ∞-norm is returned.

The solution is computed using either a second-or fourth-order accurate finite-difference approximation of the continuous equation. The resulting system of linear algebraic equations is solved using fast Fourier transform techniques. The algorithm relies upon the fact that $n_x - 1$ and $n_z - 1$ are highly composite (the product of small primes). For details of the algorithm, see Boisvert (1984). If $n_x - 1$ and $n_z - 1$ are highly composite, then the execution time of FPS3H is proportional to

$$n_x n_y n_z \left(\log_2^2 n_x + \log_2^2 n_z \right)$$

If evaluations of $p(x, y, z)$ are inexpensive, then the difference in running time between IORDER = 2 and IORDER = 4 is small.

Example

This example solves the equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + 10u = -4\cos(3x + y - 2z) + 12e^{x-z} + 10$$

with the boundary conditions $\partial u / \partial z = -2 \sin(3x + y - 2z) - \exp(x - z)$ on the front side and $u = \cos(3x + y - 2z) + \exp(x - z) + 1$ on the other five sides. The domain is the box [0, 1/4] × [0, 1/2] × [0, 1/2]. The output of FPS3H is a $9 \times 17 \times 17$ table of U values. The quadratic interpolation routine QD3VL is used to print a table of values.

```
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDU, MDU, NX, NXTABL, NY, NYTABL, NZ, NZTABL
      PARAMETER  (NX=5, NXTABL=4, NY=9, NYTABL=3, NZ=9,
     &           NZTABL=3, LDU=NX, MDU=NY)
C
      INTEGER    I, IBCTY(6), IORDER, J, K, NOUT
      REAL       AX, AY, AZ, BRHS, BX, BY, BZ, COEFU, FLOAT, PRHS,
     &           QD3VL, U(LDU,MDU,NZ), UTABL, X, ERROR, TRUE,
     &           XDATA(NX), Y, YDATA(NY), Z, ZDATA(NZ)
      INTRINSIC  FLOAT
      EXTERNAL   BRHS, FPS3H, PRHS, QD3VL, UMACH
C                                  Define domain
      AX = 0.0
      BX = 0.125
      AY = 0.0
      BY = 0.25
      AZ = 0.0
      BZ = 0.25
C                                  Set boundary condition types
      IBCTY(1) = 1
      IBCTY(2) = 1
      IBCTY(3) = 1
      IBCTY(4) = 1
      IBCTY(5) = 2
      IBCTY(6) = 1
C                                  Coefficient of U
      COEFU = 10.0
C                                  Order of the method
      IORDER = 4
C                                  Solve the PDE
      CALL FPS3H (PRHS, BRHS, COEFU, NX, NY, NZ, AX, BX, AY, BY, AZ,
     &            BZ, IBCTY, IORDER, U, LDU, MDU)
```

```
C                                 Set up for quadratic interpolation
      DO 10  I=1, NX
         XDATA(I) = AX + (BX-AX)*FLOAT(I-1)/FLOAT(NX-1)
   10 CONTINUE
      DO 20  J=1, NY
         YDATA(J) = AY + (BY-AY)*FLOAT(J-1)/FLOAT(NY-1)
   20 CONTINUE
      DO 30  K=1, NZ
         ZDATA(K) = AZ + (BZ-AZ)*FLOAT(K-1)/FLOAT(NZ-1)
   30 CONTINUE
C                                 Print the solution
      CALL UMACH (2, NOUT)
      WRITE (NOUT,'(8X,5(A,11X))') 'X', 'Y', 'Z', 'U', 'Error'
      DO 60  K=1, NZTABL
         DO 50  J=1, NYTABL
            DO 40  I=1, NXTABL
               X     = AX + (BX-AX)*FLOAT(I-1)/FLOAT(NXTABL-1)
               Y     = AY + (BY-AY)*FLOAT(J-1)/FLOAT(NYTABL-1)
               Z     = AZ + (BZ-AZ)*FLOAT(K-1)/FLOAT(NZTABL-1)
               UTABL = QD3VL(X,Y,Z,NX,XDATA,NY,YDATA,NZ,ZDATA,U,LDU,
     &                 MDU,.FALSE.)
               TRUE = COS(3.0*X+Y-2.0*Z) + EXP(X-Z) + 1.0
               ERROR = UTABL - TRUE
               WRITE (NOUT,'(5F12.4)') X, Y, Z, UTABL, ERROR
   40       CONTINUE
   50    CONTINUE
   60 CONTINUE
      END
C
      REAL FUNCTION PRHS (X, Y, Z)
      REAL       X, Y, Z
C
      REAL       COS, EXP
      INTRINSIC  COS, EXP
C                                 Right side of the PDE
      PRHS = -4.0*COS(3.0*X+Y-2.0*Z) + 12*EXP(X-Z) + 10.0
      RETURN
      END
C
      REAL FUNCTION BRHS (ISIDE, X, Y, Z)
      INTEGER    ISIDE
      REAL       X, Y, Z
C
      REAL       COS, EXP, SIN
      INTRINSIC  COS, EXP, SIN
C                                 Boundary conditions
      IF (ISIDE .EQ. 5) THEN
         BRHS = -2.0*SIN(3.0*X+Y-2.0*Z) - EXP(X-Z)
      ELSE
         BRHS = COS(3.0*X+Y-2.0*Z) + EXP(X-Z) + 1.0
      END IF
      RETURN
      END
```

Output

```
   X           Y           Z           U           Error
 0.0000      0.0000      0.0000      3.0000      0.0000
 0.0417      0.0000      0.0000      3.0348      0.0000
 0.0833      0.0000      0.0000      3.0559      0.0001
 0.1250      0.0000      0.0000      3.0637      0.0001
 0.0000      0.1250      0.0000      2.9922      0.0000
 0.0417      0.1250      0.0000      3.0115      0.0000
 0.0833      0.1250      0.0000      3.0175      0.0000
 0.1250      0.1250      0.0000      3.0107      0.0000
 0.0000      0.2500      0.0000      2.9690      0.0001
 0.0417      0.2500      0.0000      2.9731      0.0000
 0.0833      0.2500      0.0000      2.9645      0.0000
 0.1250      0.2500      0.0000      2.9440     -0.0001
 0.0000      0.0000      0.1250      2.8514      0.0000
 0.0417      0.0000      0.1250      2.9123      0.0000
 0.0833      0.0000      0.1250      2.9592      0.0000
 0.1250      0.0000      0.1250      2.9922      0.0000
 0.0000      0.1250      0.1250      2.8747      0.0000
 0.0417      0.1250      0.1250      2.9211      0.0010
 0.0833      0.1250      0.1250      2.9524      0.0010
 0.1250      0.1250      0.1250      2.9689      0.0000
 0.0000      0.2500      0.1250      2.8825      0.0000
 0.0417      0.2500      0.1250      2.9123      0.0000
 0.0833      0.2500      0.1250      2.9281      0.0000
 0.1250      0.2500      0.1250      2.9305      0.0000
 0.0000      0.0000      0.2500      2.6314     -0.0249
 0.0417      0.0000      0.2500      2.7420     -0.0004
 0.0833      0.0000      0.2500      2.8112     -0.0042
 0.1250      0.0000      0.2500      2.8609     -0.0138
 0.0000      0.1250      0.2500      2.7093      0.0000
 0.0417      0.1250      0.2500      2.8153      0.0344
 0.0833      0.1250      0.2500      2.8628      0.0242
 0.1250      0.1250      0.2500      2.8825      0.0000
 0.0000      0.2500      0.2500      2.7351     -0.0127
 0.0417      0.2500      0.2500      2.8030     -0.0011
 0.0833      0.2500      0.2500      2.8424     -0.0040
 0.1250      0.2500      0.2500      2.8735     -0.0012
```

SLEIG/DSLEIG (Single/Double precision)

Determine eigenvalues, eigenfunctions and/or spectral density functions for Sturm-Liouville problems in the form

$$-\frac{d}{dx}(p(x)\frac{du}{dx})+q(x)u=\lambda r(x)u \text{ for } x \text{ in } (a,b)$$

with boundary conditions (at regular points)

$$a_1u-a_2(pu')=\lambda(a_1'u-a_2'(pu')) \text{ at } a$$

$$b_1u+b_2(pu')=0 \text{ at } b$$

Usage

```
CALL SLEIG (CONS, COEFFN, ENDFIN, NUMEIG, INDEX, TEVLAB,
            TEVLRL, EVAL)
```

Arguments

CONS — Array of size eight containing

$$a_1, a_1', a_2, a_2', b_1, b_2, a \text{ and } b$$

in locations CONS(1) through CONS(8), respectively. (Input)

COEFFN — User-supplied SUBROUTINE to evaluate the coefficient functions. The usage is
CALL COEFFN (X, PX, QX, RX)
X — Independent variable. (Input)
PX — The value of $p(x)$ at X. (Output)
QX — The value of $q(x)$ at X. (Output)
RX — The value of $r(x)$ at X. (Output)
COEFFN must be declared EXTERNAL in the calling program.

ENDFIN — Logical array of size two. ENDFIN(1) = .true. if the endpoint *a* is finite. ENDFIN(2) = .true. if endpoint *b* is finite. (Input)

NUMEIG — The number of eigenvalues desired. (Input)

INDEX — Vector of size NUMEIG containing the indices of the desired eigenvalues. (Input)

TEVLAB — Absolute error tolerance for eigenvalues. (Input)

TELVRL — Relative error tolerance for eigenvalues. (Input)

EVAL — Array of length NUMEIG containing the computed approximations to the eigenvalues whose indices are specified in INDEX.

Comments

1. Automatic workspace is

   S2EIG 4 * NUMEIG + MAX(1000, NUMEIG + 22) units, or
   DS2EIG 8 * NUMEIG + MAX(1000, NUMEIG + 22) units.

   Workspace may be explicitly provided, if desired, by use of S2EIG/DS2EIG. The reference is

   ```
   CALL S2EIG (CONST, COEFFN, ENDFIN, NUMEIG, INDEX,
               TEVLAB, TEVLRL, EVAL, JOB, IPRINT, TOLS,
               NUMX, XEF, NRHO, T, TYPE, EF, PDEF, RHO,
               IFLAG, WORK, IWORK)
   ```

   The additional arguments are as follows:

   JOB — Logical array of length five. (Input)

   JOB(1) = .true. if a set of eigenvalues are to be computed but not their eigenfunctions.

`JOB(2) = .true.` if a set of eigenvalue and eigenfunction pairs are to be computed.

`JOB(3) = .true.` if the spectral function is to be computed over some subinterval of the essential spectrum.

`JOB(4) = .true.` if the normal automatic classification is overridden. If `JOB(4) = .true.` then `TYPE(*,*)` must be entered correctly. Most users will not want to override the classification process, but it might be appropriate for users experimenting with problems for which the coefficient functions do not have power-like behavior near the singular endpoints. The classification is considered sufficiently important for spectral density function calculations that `JOB(4)` is ignored with `JOB(3) = .true.`.

`JOB(5) = .true.` if mesh distribution is chosen by `SLEIG`. If `JOB(5) = .true.` and `NUMX` is zero, the number of mesh points are also chosen by `SLEIG`. If `NUMX` > 0 then `NUMX` mesh points will be used. If `JOB(5) = .false.`, the number `NUMX` and distribution `XEF(*)` must be input by the user.

IPRINT — Control levels of internal printing. (Input)
No printing is performed if `IPRINT` = 0. If either `JOB(1)` or `JOB(2)` is true:

`IPRINT`	**Printed Output**
1	initial mesh (the first 51 or fewer points), eigenvalue estimate at each level
4	the above and at each level matching point for eigenfunction shooting, `X(*), EF(*)` and `PDEF(*)` values
5	the above and at each level the brackets for the eigenvalue search, intermediate shooting information for the eigenfunction and eigenfunction norm.

If `JOB(3) = .true.`

`IPRINT`	**Printed Output**
1	the actual (a, b) used at each iteration and the total number of eigenvalues computed
2	the above and switchover points to the asymptotic formulas, and some intermediate $\rho(t)$ approximations
4	the above and initial meshes for each iteration, the index of the largest eigenvalue which may be computed, and various eigenvalue and R_N values
4	the above and $\hat{\rho}$ values at each level
5	the above and R_N add eigenvalues below the switchover point

If `JOB(4)=.false.`

`IPRINT`	**Printed Output**
2	output a description of the spectrum
3	the above and the constants for the Friedrichs' boundary condition(s)
5	the above and intermediate details of the classification calculation

TOLS — Array of length 4 containing tolerances.   (Input)
`TOLS(1)` — absolute error tolerance for eigenfunctions
`TOLS(2)` — relative error tolerance for eigenfunctions
`TOLS(3)` — absolute error tolerance for eigenfunction derivatives
`TOLS(4)` — relative error tolerance for eigenfunction derivatives

The absolute tolerances must be positive.
The relative tolerances must be at least 100 `*amach(4)`

NUMX — Integer whose value is the number of output points where each eigenfunction is to be evaluated (the number of entries in `XEF(*)`) when `JOB(2) = .true.`. If `JOB(5)= .false.` and `NUMX` is greater than zero, then `NUMX` is the number of points in the initial mesh used. If `JOB(5) = .false.`, the points in `XEF(*)` should be chosen with a reasonable distribution. Since the endpoints *a* and *b* must be part of any mesh, `NUMX` cannot be one in this case. If `JOB(5) = .false.` and `JOB(3) = .true.`, then `NUMX` must be positive. On output, `NUMX` is set to the number of points for eigenfunctions when input `NUMX` = 0, and `JOB(2)` or `JOB(5) = .true.`.   (Input/Output)

XEF — Array of points on input where eigenfunction estimates are desired, if `JOB(2) = .true.`. Otherwise, if `JOB(5) = .false.` and `NUMX` is greater than zero, the user's initial mesh is entered. The entries must be ordered so that *a* = `XEF(1)` < `XEF(2)` < … < `XEF(NUMX)` = *b*. If either endpoint is infinite, the corresponding `XEF(1)` or `XEF(NUMX)` is ignored. However, it is required that `XEF(2)` be negative when `ENDFIN(1) = .false.`, and that `XEF(NUMX-1)` be positive when `ENDFIN(2) = .false.`. On output, `XEF(*)` is changed only if `JOB(2)` and `JOB(5)` are true. If `JOB(2) = .false.`, this vector is not referenced. If `JOB(2) = .true.` and `NUMX` is greater than zero on input, `XEF(*)` should be dimensioned at least `NUMX` + 16. If `JOB(2)` is true and `NUMX` is zero on input, `XEF(*)` should be dimensioned at least 31.

NRHO — The number of output values desired for the array `RHO(*)`. `NRHO` is not used if `JOB(3) = .false.`.   (Input)

T — Real vector of size `NRHO` containing values where the spectral function `RHO(*)` is desired. The entries must be sorted in increasing order. The existence and location of a continuous spectrum can be determined by calling `SLEIG` with the first four entries of `JOB` set to false and `IPRINT` set to 1. `T(*)` is not used if `JOB(3) = .false.`. (Input)

TYPE — 4 by 2 logical matrix. Column 1 contains information about endpoint *a* and column 2 refers to endpoint *b*.
`TYPE(1,*) = .true.` if and only if the endpoint is regular
`TYPE(2,*) = .true.` if and only if the endpoint is limit circle
`TYPE(3,*) = .true.` if and only if the endpoint is nonoscillatory for all eigenvalues
`TYPE(4,*) = .true.` if and only if the endpoint is oscillatory for all eigenvalues
Note: all of these values must be correctly input if `JOB(4) = .true.`. Otherwise, `TYPE(*,*)` is output. (Input/Output)

EF — Array of eigenfunction values. `EF((k – 1)*NUMX + i)` is the estimate of `u(XEF(i))` corresponding to the eigenvalue in `EV(k)`. If `JOB(2) = .false.` then this vector is not referenced. If `JOB(2) = .true.` and `NUMX` is greater than zero on entry, then `EF(*)` should be dimensioned at least `NUMX * NUMEIG`. If `JOB(2) = .true.` and `NUMX` is zero on input, then `EF(*)` should be dimensioned 31 * `NUMEIG`. (Output)

PDEF — Array of eigenfunction derivative values. `PDEF((k-1)*NUMX + i)` is the estimate of (*pu′*) `(XEF(i))` corresponding to the eigenvalue in `EV(k)`. If `JOB(2) = .false.` this vector is not referenced. If `JOB(2) = .true.`, it must be dimensioned the same as `EF(*)`. (Output)

RHO — Array of size `NRHO` containing values for the spectral density function ρ(*t*), `RHO(I)` = `ρ(T(I))`. This vector is not referenced if `JOB(3)` is false. (Output)

IFLAG — Array of size `max(1, numeig)` containing information about the output. `IFLAG(K)` refers to the K-th eigenvalue, when `JOB(1)` or `JOB(2) = .true.`. Otherwise, only `IFLAG(1)` is used. Negative values are associated with fatal errors, and the calculations are ceased. Positive values indicate a warning. (Output)
`IFLAG(K)`

`IFLAG(K)`	**Description**
–1	too many levels needed for the eigenvalue calculation; problem seems too difficult at this tolerance. Are the coefficient functions nonsmooth?
–2	too many levels needed for the eigenfunction calculation; problem seems too difficult at this tolerance. Are the eigenfunctions ill-conditioned?
–3	too many levels needed for the spectral density calculation; problem seems too difficult at this tolerance.
–4	the user has requested the spectral density function for a problem which has no continuous spectrum.

`IFLAG(K)`	**Description**
–5	the user has requested the spectral density function for a problem with both endpoints generating essential spectrum, i.e. both endpoints either OSC or O-NO.
–6	the user has requested the spectral density function for a problem in spectral category 2 for which a proper normalization of the solution at the NONOSC endpoint is not known; for example, problems with an irregular singular point or infinite endpoint at one end and continuous spectrum generated at the other.
–7	problems were encountered in obtaining a bracket.
–8	too small a step was used in the integration. The `TOLS(*)` values may be too small for this problem.
–9	too small a step was used in the spectral density function calculation for which the continuous spectrum is generated by a finite endpoint.
–10	an argument to the circular trig functions is too large. Try running the problem again with a finer initial mesh or, for singular problems, use interval truncation.
–15	$p(x)$ and $r(x)$ are not positive in the interval (a, b).
–20	eigenvalues and/or eigenfunctions were requested for a problem with an OSC singular endpoint. Interval truncation must be used on such problems.
1	failure in the bracketing procedure probably due to a cluster of eigenvalues which the code cannot separate. Calculations have continued but any eigenfunction results are suspect. Try running the problem again with tighter input tolerances to separate the cluster.
2	there is uncertainty in the classification for this problem. Because of the limitations of floating point arithmetic, and the nature of the finite sampling, the routine cannot be certain about the classification information at the requested tolerance.
3	there may be some eigenvalues embedded in the essential spectrum. Use of `IPRINT` greater than zero will provide additional output giving the location of the approximating eigenvalues for the step function problem. These could be extrapolated to estimate the actual eigenvalue embedded in the essential spectrum.
4	a change of variables was made to avoid potentially slow convergence. However, the global error estimates may not be as reliable. Some experimentation using different tolerances is recommended.

`IFLAG(K)`	**Description**
6	there were problems with eigenfunction convergence in a spectral density calculation. The output ρ(*t*) may not be accurate.

WORK — Array of size `MAX`(1000, `NUMEIG` + 22) used for workspace.

IWORK — Integer array of size `NUMEIG` + 3 used for workspace.

Algorithm

This subroutine is designed for the calculation of eigenvalues, eigenfunctions and/or spectral density functions for Sturm-Liouville problems in the form

$$-\frac{d}{dx}(p(x)\frac{du}{dx})+q(x)u=\lambda r(x)u \text{ for } x \text{ in } (a,b) \qquad (1)$$

with boundary conditions (at regular points)

$$a_1u-a_2(pu')=\lambda\left(a_1'u-a_2'(pu')\right) \text{ at } a$$

$$b_1u+b_2(pu')=0 \text{ at } b$$

We assume that

$$a_1'a_2-a_1a_2'>0$$

when $a'_1 \neq 0$ and $a'_2 \neq 0$. The problem is considered regular if and only if

- *a* and *b* are finite,
- *p*(*x*) and *r*(*x*) are positive in (*a*, *b*),
- *1/p*(*x*), *q*(*x*) and *r*(*x*) are locally integrable near the endpoints.

Otherwise the problem is called singular. The theory assumes that *p*, *p*′, *q*, and *r* are at least continuous on (*a*, *b*), though a finite number of jump discontinuities can be handled by suitably defining an input mesh.

For regular problems, there are an infinite number of eigenvalues

$$\lambda_0 < \lambda_1 < \ldots < \lambda_k,\ k \rightarrow \infty$$

Each eigenvalue has an associated eigenfunction which is unique up to a constant. For singular problems, there is a wide range in the behavior of the eigenvalues.

As presented in Pruess and Fulton (1993) the approach is to replace (1) by a new problem

$$-(\hat{p}\hat{u}')'+\hat{q}\hat{u}=\hat{\lambda}\hat{r}\hat{u} \qquad (2)$$

with analogous boundary conditions

$$a_1\hat{u}(a)-a_2(\hat{p}\hat{u}')(a)=\hat{\lambda}\left[a_1'\hat{u}(a)-a_2{}'(\hat{p}\hat{u}')(a)\right]$$

$$b_1\hat{u}(b)+b_2(\hat{p}\hat{u}')(b)=0$$

where

$$\hat{p}, \hat{q} \text{ and } \hat{r}$$

are step function approximations to p, q, and r, respectively. Given the mesh $a = x_1 < x_2 < \ldots < x_{N+1} = b$, the usual choice for the step functions uses midpoint interpolation, i. e.,

$$\hat{p}(x) = p_n \equiv p(\frac{x_n + x_{n+1}}{2})$$

for x in (x_n, x_{n+1}) and similarly for the other coefficient functions. This choice works well for regular problems. Some singular problems require a more sophisticated technique to capture the asymptotic behavior. For the midpoint interpolants, the differential equation (2) has the known closed form solution in (x_n, x_{n+1})

$$\hat{u}(x) = \hat{u}(x_n)\phi_n'(x - x_n) + (\hat{p}\hat{u}')(x_n)\phi_n(x - x_n) / p_n$$

with

$$\phi_n(t) = \begin{cases} \sin \omega_n t / \omega_n, \tau_n > 0 \\ \sinh \omega_n t / \omega_n, \tau_n < 0 \\ t, \tau = 0 \end{cases}$$

where

$$\tau_n = \left(\hat{\lambda} r_n - q_n\right) / p_n$$

and

$$\omega_n = \sqrt{|\tau_n|}$$

Starting with,

$$\hat{u}(a) \text{ and } (\hat{p}\hat{u}')(a)$$

consistent with the boundary condition,

$$\hat{u}(a) = a_2 - a_2'\hat{\lambda}$$
$$(\hat{p}\hat{u}')(a) = a_1 - a_1'\hat{\lambda}$$

an algorithm is to compute for $n = 1, 2, ..., N$,

$$\hat{u}(x_{n+1}) = \hat{u}(x_n)\phi_n'(h_n) + (\hat{p}\hat{u}')(x_n)\phi_n(h_n) / p_n$$
$$(\hat{p}\hat{u}')(x_{n+1}) = -\tau_n p_n \hat{u}(x_n)\phi_n'(h_n) + (\hat{p}\hat{u}')(x_n)\phi_n(h_n)$$

which is a shooting method. For a fixed mesh we can iterate on the approximate eigenvalue until the boundary condition at b is satisfied. This will yield an $O(h^2)$ approximation

$$\hat{\lambda}_k$$

to some λ_k.

The problem (2) has a step spectral function given by

$$\hat{\rho}(t) = \sum \frac{1}{\int \hat{r}(x)\hat{u}_k^2(x)dx + \alpha}$$

where the sum is taken over k such that

$$\hat{\lambda}_k \le t$$

and

$$\alpha = a_1' a_2 - a_1 a_2'$$

Example 1

This examples computes the first ten eigenvalues of the problem from Titchmarsh (1962) given by

$p(x) = r(x) = 1$

$q(x) = x$

$[a, b] = [0, \infty]$

$u(a) = u(b) = 0$

The eigenvalues are known to be the zeros of

$$f(\lambda) = J_{1/3}(\frac{2}{3}\lambda^{3/2}) + J_{-1/3}(\frac{2}{3}\lambda^{3/2})$$

For each eigenvalue λ_k, the program prints k, λ_k and $f(\lambda_k)$.

```
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, INDEX(10), NUMEIG
      REAL       CBS1(1), CBS2(1), CONS(8), EVAL(10), LAMBDA, TEVLAB,
     &           TEVLRL, XNU, Z
      LOGICAL    ENDFIN(2)
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  CMPLX, SQRT
      REAL       SQRT
      COMPLEX    CMPLX
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   CBJS, SLEIG
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   AMACH, COEFF
      REAL       AMACH
C
      CALL UMACH (2, NOUT)
C                                  Define boundary conditions
      CONS(1) = 1.0
      CONS(2) = 0.0
      CONS(3) = 0.0
      CONS(4) = 0.0
      CONS(5) = 1.0
      CONS(6) = 0.0
      CONS(7) = 0.0
      CONS(8) = 0.0
C
      ENDFIN(1) = .TRUE.
      ENDFIN(2) = .FALSE.
C                                  Compute the first 10 eigenvalues
      NUMEIG = 10
      DO 10  I=1, NUMEIG
         INDEX(I) = I - 1
   10 CONTINUE
C                                  Set absolute and relative tolerance
      TEVLAB = 10.0*AMACH(4)
      TEVLRL = SQRT(AMACH(4))
C
      CALL SLEIG (CONS, COEFF, ENDFIN, NUMEIG, INDEX, TEVLAB, TEVLRL,
     &            EVAL)
C
      XNU = -1.0/3.0
      WRITE(NOUT,99998)
      DO 20  I=1, NUMEIG
         LAMBDA = EVAL(I)
         Z      = CMPLX(2.0/3.0*LAMBDA*SQRT(LAMBDA),0.0)
         CALL CBJS (XNU, Z, 1, CBS1)
         CALL CBJS (-XNU, Z, 1, CBS2)
         WRITE (NOUT,99999) I-1, LAMBDA, CBS1(1) + CBS2(1)
   20 CONTINUE
C
99998 FORMAT(/, 2X, ’index’, 5X, ’lambda’, 5X, ’f(lambda)’,/)
99999 FORMAT(I5, F13.4, E15.4)
      END
C
      SUBROUTINE COEFF (X, PX, QX, RX)
C                                  SPECIFICATIONS FOR ARGUMENTS
      REAL       X, PX, QX, RX
C
      PX = 1.0
      QX = X
      RX = 1.0
      RETURN
      END
```

Output

```
 index     lambda     f(lambda)
   0       2.3381    -0.8225E-05
   1       4.0879    -0.1654E-04
   2       5.5205     0.6844E-04
   3       6.7867    -0.4515E-05
   4       7.9440     0.8953E-04
   5       9.0227     0.1122E-04
   6      10.0401     0.1031E-03
   7      11.0084    -0.7913E-04
   8      11.9361    -0.2550E-04
   9      12.8293     0.2321E-03
```

Example 2

In this problem from Scott, Shampine and Wing (1969),

$p(x) = r(x) = 1$

$q(x) = x^2 + x^4$

$[a, b] = [-\infty, \infty]$

$u(a) = u(b) = 0$

the first eigenvalue and associated eigenfunction, evaluated at selected points, are computed. As a rough check of the correctness of the results, the magnitude of the residual

$$-\frac{d}{dx}(p(x)\frac{du}{dx}) + q(x)u - \lambda r(x)u$$

is printed. We compute a spline interpolant to u' and use the function CSDER to estimate the quantity $-(p(x)u')'$.

```
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, IFLAG(1), INDEX(1), IWORK(100), NINTV, NOUT, NRHO,
     &           NUMEIG, NUMX
      REAL       BRKUP(61), CONS(8), CSCFUP(4,61), EF(61), EVAL(1),
     &           LAMBDA, PDEF(61), PX, QX, RESIDUAL, RHO(1), RX, T(1),
     &           TEVLAB, TEVLRL, TOLS(4), WORK(3000), X, XEF(61)
      LOGICAL    ENDFIN(2), JOB(5), TYPE(4,2)
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  ABS, REAL
      REAL       ABS, REAL
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   COEFF, CSAKM, S2EIG, UMACH
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CSDER
      REAL       CSDER
C                                  Define boundary conditions
      CONS(1) = 1.0
      CONS(2) = 0.0
      CONS(3) = 0.0
      CONS(4) = 0.0
      CONS(5) = 1.0
      CONS(6) = 0.0
      CONS(7) = 0.0
      CONS(8) = 0.0
C                                  Compute eigenvalue and eigenfunctions
      JOB(1) = .FALSE.
      JOB(2) = .TRUE.
     JOB(3) = .FALSE.
      JOB(4) = .FALSE.
      JOB(5) = .FALSE.
C
      ENDFIN(1) = .FALSE.
      ENDFIN(2) = .FALSE.
C                                  Compute eigenvalue with index 0
      NUMEIG   = 1
      INDEX(1) = 0
C
```

```
      TEVLAB  = 1.0E-3
      TEVLRL  = 1.0E-3
      TOLS(1) = TEVLAB
      TOLS(2) = TEVLRL
      TOLS(3) = TEVLAB
      TOLS(4) = TEVLRL
      NRHO    = 0
C                                  Set up mesh, points at which u and
C                                  u’ will be computed
      NUMX = 61
      DO 10  I=1, NUMX
         XEF(I) = 0.05*REAL(I-31)
   10 CONTINUE
C
      CALL S2EIG (CONS, COEFF, ENDFIN, NUMEIG, INDEX, TEVLAB, TEVLRL,
     &            EVAL, JOB, 0, TOLS, NUMX, XEF, NRHO, T, TYPE, EF,
     &            PDEF, RHO, IFLAG, WORK, IWORK)
C
      LAMBDA = EVAL(1)
   20 CONTINUE
C                                  Compute spline interpolant to u’
C
      CALL CSAKM (NUMX, XEF, PDEF, BRKUP, CSCFUP)
      NINTV = NUMX - 1
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99997) ’     lambda = ’, LAMBDA
      WRITE (NOUT,99999)
C                                  At a subset of points from the
C                                  input mesh, compute residual =
C                                  abs( -(u’)’ + q(x)u - lambda*u ).
C                                  We know p(x) = 1 and r(x) = 1.
      DO 30  I=1, 41, 2
         X = XEF(I+10)
         CALL COEFF (X, PX, QX, RX)
C
C                                  Use the spline fit to u’ to
C                                  estimate u’’ with CSDER
C
         RESIDUAL = ABS(-CSDER(1,X,NINTV,BRKUP,CSCFUP)+QX*EF(I+10)-
     &              LAMBDA*EF(I+10))
         WRITE (NOUT,99998) X, EF(I+10), PDEF(I+10), RESIDUAL
   30 CONTINUE
C
99997 FORMAT (/, A14, F10.5, /)
99998 FORMAT (5X, F4.1, 3F15.5)
99999 FORMAT (7X, ’x’, 11X, ’u(x)’, 10X, ’u’’(x)’, 9X, ’residual’, /)
      END
C
      SUBROUTINE COEFF (X, PX, QX, RX)
C                                  SPECIFICATIONS FOR ARGUMENTS
      REAL       X, PX, QX, RX
C
      PX = 1.0
      QX = X*X + X*X*X*X
      RX = 1.0
      RETURN
      END
```

Output

```
lambda =    1.39247
    x           u(x)          u′(x)         residual
 -1.0        0.38632        0.65019        0.00189
 -0.9        0.45218        0.66372        0.00081
 -0.8        0.51837        0.65653        0.00023
 -0.7        0.58278        0.62827        0.00113
 -0.6        0.64334        0.57977        0.00183
 -0.5        0.69812        0.51283        0.00230
 -0.4        0.74537        0.42990        0.00273
 -0.3        0.78366        0.33393        0.00265
 -0.2        0.81183        0.22811        0.00273
 -0.1        0.82906        0.11570        0.00278
  0.0        0.83473        0.00000        0.00136
  0.1        0.82893       -0.11568        0.00273
  0.2        0.81170       -0.22807        0.00273
  0.3        0.78353       -0.33388        0.00267
  0.4        0.74525       -0.42983        0.00265
  0.5        0.69800       -0.51274        0.00230
  0.6        0.64324       -0.57967        0.00182
  0.7        0.58269       -0.62816        0.00113
  0.8        0.51828       -0.65641        0.00023
  0.9        0.45211       -0.66361        0.00081
  1.0        0.38626       -0.65008        0.00189
```

SLCNT/DSLCNT (Single/Double precision)

Calculate the indices of eigenvalues of a Sturm-Liouville problem of the form for

$$-\frac{d}{dx}(p(x)\frac{du}{dx})+q(x)u=\lambda r(x)u \text{ for } x \text{ in } [a,b]$$

with boundary conditions (at regular points)

$$a_1u-a_2(pu')=\lambda(a_1'u-a_2'(pu')) \text{ at } a$$

$$b_1u+b_2(pu')=0 \text{ at } b$$

in a specified subinterval of the real line, [α, β].

Usage

```
CALL SLCNT (ALPHA, BETA, CONS, COEFFN, ENDFIN, IFIRST,
            NTOTAL)
```

Arguments

ALPHA — Value of the left end point of the search interval. (Input)

BETA — Value of the right end point of the search interval. (Input)

CONS — Array of size eight containing

$$a_1, a_1', a_2, a_2', b_1, b_2, a \text{ and } b$$

in locations `CONS(1)` … `CONS(8)`, respectively.   (Input)

COEFFN — User-supplied `SUBROUTINE` to evaluate the coefficient functions. The usage is
`CALL COEFFN (X, PX, QX, RX)`
`X` – Independent variable.   (Input)
`PX` – The value of $p(x)$ at `X`.   (Output)
`QX` – The value of $q(x)$ at `X`.   (Output)
`RX` – The value of $r(x)$ at `X`.   (Output)
`COEFFN` must be declared `EXTERNAL` in the calling program.

ENDFIN — Logical array of size two. `ENDFIN = .true.` if and only if the endpoint a is finite. `ENDFIN(2) = .true.` if and only if endpoint b is finite. (Input)

IFIRST — The index of the first eigenvalue greater than α.   (Output)

NTOTAL — Total number of eigenvalues in the interval [α, β].   (Output)

Algorithm

This subroutine computes the indices of eigenvalues, if any, in a subinterval of the real line for Sturm-Liouville problems in the form

$$-\frac{d}{dx}(p(x)\frac{du}{dx})+q(x)u=\lambda r(x)u \text{ for } x \text{ in } [a,b]$$

with boundary conditions (at regular points)

$$a_1u-a_2(pu')=\lambda\left(a_1'u-a_2'(pu')\right) \text{ at } a$$

$$b_1u+b_2(pu')=0 \text{ at } b$$

It is intended to be used in conjunction with `SLEIG`, page 745. `SLCNT` is based on the routine `INTERV` from the package `SLEDGE`.

Example

Consider the harmonic oscillator (Titchmarsh) defined by

$p(x) = 1$

$q(x) = x^2$

$r(x) = 1$

$[a, b] = [-\infty, \infty]$

$u(a) = 0$

$u(b) = 0$

The eigenvalues of this problem are known to be

$\lambda_k = 2k + 1, k = 0, 1, \ldots$

Therefore in the interval [10, 16] we expect SLCNT to note three eigenvalues, with the first of these having index five.

```
C                                SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IFIRST, NOUT, NTOTAL
      REAL       ALPHA, BETA, CONS(8)
      LOGICAL    ENDFIN(2)
C                                SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   SLCNT, UMACH
C                                SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   COEFFN
C
      CALL UMACH (2, NOUT)
C                                set u(a) = 0, u(b) = 0
      CONS(1) = 1.0E0
      CONS(2) = 0.0E0
      CONS(3) = 0.0E0
      CONS(4) = 0.0E0
      CONS(5) = 1.0E0
      CONS(6) = 0.0E0
      CONS(7) = 0.0E0
      CONS(8) = 0.0E0
C
      ENDFIN(1) = .FALSE.
      ENDFIN(2) = .FALSE.
C
      ALPHA = 10.0
      BETA  = 16.0
C
      CALL SLCNT (ALPHA, BETA, CONS, COEFFN, ENDFIN, IFIRST, NTOTAL)
C
      WRITE (NOUT,99998) ALPHA, BETA, IFIRST
      WRITE (NOUT,99999) NTOTAL
C
99998 FORMAT (/, 'Index of first eigenvalue in [', F5.2, ',', F5.2,
     &       '] IS ', I2)
99999 FORMAT ('Total number of eigenvalues in this interval: ', I2)
C
      END
C
      SUBROUTINE COEFFN (X, PX, QX, RX)
C                                SPECIFICATIONS FOR ARGUMENTS
      REAL       X, PX, QX, RX
C
      PX = 1.0E0
      QX = X*X
      RX = 1.0E0
      RETURN
      END
```

Output

```
Index of first eigenvalue in [10.00,16.00] is 5
Total number of eigenvalues in this interval: 3
```

Chapter 6: Transforms

Routines

6.1. **Real Trigonometric FFT**

6.2. **Complex Exponential FFT**

6.3. **Real Sine and Cosine FFTs**

6.4. **Real Quarter Sine and Quarter Cosine FFTs**

6.5. **Two- and Three-Dimensional Complex FFTs**

6.6. **Convolutions and Correlations**

Usage Notes

Fast Fourier Transforms

A Fast Fourier Transform (FFT) is simply a discrete Fourier transform that can be computed efficiently. Basically, the straightforward method for computing the Fourier transform takes approximately N^2 operations where N is the number of points in the transform, while the FFT (which computes the same values) takes approximately $N \log N$ operations. The algorithms in this chapter are modeled on the Cooley-Tukey (1965) algorithm; hence, the computational savings occur, not for all integers N, but for N which are highly composite. That is, N (or in certain cases $N + 1$ or $N - 1$) should be a product of small primes.

All of the FFT routines compute a *discrete* Fourier transform. The routines accept a vector x of length N and return a vector

$$\hat{x}$$

defined by

$$\hat{x}_m := \sum_{n=1}^{N} x_n \omega_{nm}$$

The various transforms are determined by the selection of ω. In the following table, we indicate the selection of ω for the various transforms. This table should not be mistaken for a definition since the precise transform definitions (at times) depend on whether N or m is even or odd.

Routine	ω_{nm}
FFTRF	cos or sin $\frac{(m-1)(n-1)2\pi}{N}$
FFTRB	cos or sin $\frac{(m-1)(n-1)2\pi}{N}$
FFTCF	$\mathrm{e}^{-2\pi i(n\text{-}1)(m-1)/N}$
FFTCB	$\mathrm{e}^{2\pi i(n\text{-}1)(m-1)/N}$
FSINT	$\sin \frac{nm\pi}{N+1}$
FCOST	$\cos \frac{(n-1)(m-1)\pi}{N-1}$
QSINF	$2 \sin \frac{(2m-1)n\pi}{2N}$
QSINB	$4 \sin \frac{(2n-1)m\pi}{2N}$
QCOSF	$2 \cos \frac{(2m-1)(n-1)\pi}{2N}$
QCOSB	$4 \cos \frac{(2\mathrm{n}-1)(m-1)\pi}{2N}$

For many of the routines listed above, there is a corresponding “I” (for initialization) routine. Use these routines *only* when repeatedly transforming sequences of the same length. In this situation, the “I” routine will compute the initial setup once, and then the user will call the corresponding “2” routine. This can result in substantial computational savings. For more information on the usage of these routines, the user should consult the documentation under the appropriate routine name.

In addition to the one-dimensional transformations described above, we also provide complex twoand three-dimensional FFTs and their inverses based on calls to either FFTCF (page 772) or FFTCB (page 774). If you need a higher dimensional transform, then you should consult the example program for FFTCI (page 777) which suggests a basic strategy one could employ.

Continuous versus discrete Fourier transform

There is, of course, a close connection between the discrete Fourier transform and the continuous Fourier transform. Recall that the continuous Fourier transform is defined (Brigham, 1974) as

$$\hat{f}(\omega) = (F\,f)(\omega) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i\omega t}dt$$

We begin by making the following approximation:

$$\begin{aligned}\hat{f}(\omega) &\approx \int_{-T/2}^{T/2} f(t)e^{-2\pi i\omega t}dt \\ &= \int_0^T f(t-T/2)e^{-2\pi i\omega(t-T/2)}dt \\ &= e^{\pi i\omega T}\int_0^T f(t-T/2)e^{-2\pi i\omega t}dt\end{aligned}$$

If we approximate the last integral using the rectangle rule with spacing $h = T/N$, we have

$$\hat{f}(\omega) \approx e^{\pi i\omega T} h\sum_{k=0}^{N-1} e^{-2\pi i\omega kh} f(kh - T/2)$$

Finally, setting $\omega = j/T$ for $j = 0, \ldots, N-1$ yields

$$\hat{f}(j/T) \approx e^{\pi ij} h\sum_{k=0}^{N-1} e^{-2\pi ijk/N} f(kh - T/2) = (-1)^j h\sum_{k=0}^{N-1} e^{-2\pi ijk/N} f_k^h$$

where the vector $f^{\,h} = (f(-T/2), \ldots, f((N-1)h - T/2))$. Thus, after scaling the components by $(-1)^j h$, the discrete Fourier transform as computed in `FFTCF` (with input f^h) is related to an approximation of the continuous Fourier transform by the above formula. This is seen more clearly by making a change of variables in the last sum. Set

$$n = k+1,\ m = j+1, \text{ and } f_k^h = x_n$$

then, for $m = 1, \ldots, N$ we have

$$\hat{f}\big((m-1)/T\big) \approx -(-1)^m h\hat{x}_m = -(-1)^m h\sum_{n=1}^{N} e^{-2\pi i(m-1)(n-1)/N} x_n$$

If the function f is expressed as a FORTRAN function routine, then the continuous Fourier transform

$$\hat{f}$$

can be approximated using the IMSL routine `QDAWF` (page 604).

Inverse Laplace transform

The last two routines described in this chapter, `INLAP` (page 827) and `SINLP` (page 830), compute the inverse Laplace transforms.

FFTRF/DFFTRF (Single/Double precision)

Compute the Fourier coefficients of a real periodic sequence.

Usage

```
CALL FFTRF (N, SEQ, COEF)
```

Arguments

N — Length of the sequence to be transformed. (Input)

SEQ — Array of length `N` containing the periodic sequence. (Input)

COEF — Array of length `N` containing the Fourier coefficients. (Output)

Comments

1. Automatic workspace usage is

   `FFTRF` 2`N` + 15 units, or
   `DFFTRF` 4`N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `F2TRF`/`DF2TRF`. The reference is

   `CALL F2TRF (N, SEQ, COEF, WFFTR)`

   The additional argument is

   WFFTR — Array of length 2`N` + 15 initialized by `FFTRI` (page 770). (Input)
   The initialization depends on `N`.

2. The routine `FFTRF` is most efficient when `N` is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `FFTRF`/`FFTRB` is used repeatedly with the same value of `N`, then call `FFTRI` followed by repeated calls to `F2TRF`/`F2TRB`. This is more efficient than repeated calls to `FFTRF`/`FFTRB`.

Algorithm

The routine `FFTRF` computes the discrete Fourier transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm that is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an *N*-vector *s* = `SEQ`, `FFTRF` returns in *c* = `COEF`, if *N* is even:

$$c_{2m-2} = \sum_{n=1}^{N} s_n \cos\left[\frac{(m-1)(n-1)2\pi}{N}\right] \qquad m = 2, \ldots, N/2+1$$

$$c_{2m-1} = -\sum_{n=1}^{N} s_n \sin\left[\frac{(m-1)(n-1)2\pi}{N}\right] \qquad m = 2, \ldots, N/2$$

$$c_1 = \sum_{n=1}^{N} s_n$$

If N is odd, c_m is defined as above for m from 2 to $(N + 1)/2$.

We now describe a fairly common usage of this routine. Let f be a real valued function of time. Suppose we sample f at N equally spaced time intervals of length Δ seconds starting at time t_0. That is, we have

$$\mathtt{SEQ}_i := f(t_0 + (i-1)\Delta) \quad i = 1, 2, \ldots, N$$

The routine `FFTRF` treats this sequence as if it were periodic of period N. In particular, it assumes that $f(t_0) = f(t_0 + N\Delta)$. Hence, the period of the function is assumed to be $T = N\Delta$.

Now, `FFTRF` accepts as input `SEQ` and returns as output coefficients c = `COEF` that satisfy the following relation when N is odd (N even is similar):

$$\mathrm{SEQ}_i = \frac{1}{N}\left[c_1 + 2\sum_{n=2}^{(N+1)/2} c_{2n-2}\cos\left[\frac{2\pi(n-1)(i-1)}{N}\right] - 2\sum_{n=2}^{(N+1)/2} c_{2n-1}\sin\left[\frac{2\pi(n-1)(i-1)}{N}\right]\right]$$

This formula is very revealing. It can be interpreted in the following manner. The coefficients produced by `FFTRF` produce an interpolating trigonometric polynomial to the data. That is, if we define

$$\begin{aligned} g(t) &:= \frac{1}{N}\left[c_1 + 2\sum_{n=2}^{(N+1)/2} c_{2n-2}\cos\left[\frac{2\pi(n-1)(t-t_0)}{N\Delta}\right] - 2\sum_{n=2}^{(N+1)/2} c_{2n-1}\sin\left[\frac{2\pi(n-1)(t-t_0)}{N\Delta}\right]\right] \\ &= \frac{1}{N}\left[c_1 + 2\sum_{n=2}^{(N+1)/2} c_{2n-2}\cos\left[\frac{2\pi(n-1)(t-t_0)}{T}\right] - 2\sum_{n=2}^{(N+1)/2} c_{2n-1}\sin\left[\frac{2\pi(n-1)(t-t_0)}{T}\right]\right] \end{aligned}$$

then, we have

$$f(t_0 + (\mathrm{i} - 1)\Delta) = g(t_0 + (i-1)\Delta)$$

Now, suppose we want to discover the dominant frequencies. One forms the vector P of length $N/2$ as follows:

$$P_1 := |c_1|$$

$$P_k := \sqrt{c_{2k-2}^2 + c_{2k-1}^2} \quad k = 2, 3, \ldots, (N+1)/2$$

These numbers correspond to the energy in the spectrum of the signal. In particular, P_k corresponds to the energy level at frequency

$$\frac{k-1}{T} = \frac{k-1}{N\Delta} \quad k = 1, 2, \ldots, \frac{N+1}{2}$$

Furthermore, note that there are only $(N + 1)/2 \approx T/(2\Delta)$ resolvable frequencies when N observations are taken. This is related to the Nyquist phenomenon, which is induced by discrete sampling of a continuous signal.

Similar relations hold for the case when N is even.

Finally, note that the Fourier transform has an (unnormalized) inverse that is implemented in FFTRB (page 768). The routine FFTRF is based on the real FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, a pure cosine wave is used as a data vector, and its Fourier series is recovered. The Fourier series is a vector with all components zero except at the appropriate frequency where it has an N.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, TWOPI, SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, FFTRF, UMACH
C
      TWOPI = 2.0*CONST('PI')
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 This loop fills out the data vector
C                                 with a pure exponential signal
      DO 10  I=1, N
         SEQ(I) = COS(FLOAT(I-1)*TWOPI/FLOAT(N))
   10 CONTINUE
C                                 Compute the Fourier transform of SEQ
      CALL FFTRF (N, SEQ, COEF)
C                                 Print results
      WRITE (NOUT,99998)
99998 FORMAT (9X, 'INDEX', 5X, 'SEQ', 6X, 'COEF')
      WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99999 FORMAT (1X, I11, 5X, F5.2, 5X, F5.2)
      END
```

Output

```
INDEX     SEQ      COEF
  1      1.00      0.00
  2      0.62      3.50
  3     -0.22      0.00
  4     -0.90      0.00
  5     -0.90      0.00
  6     -0.22      0.00
  7      0.62      0.00
```

FFTRB/DFFTRB (Single/Double precision)

Compute the real periodic sequence from its Fourier coefficients.

Usage

```
CALL FFTRB (N, COEF, SEQ)
```

Arguments

N — Length of the sequence to be transformed. (Input)

COEF — Array of length `N` containing the Fourier coefficients. (Input)

SEQ — Array of length `N` containing the periodic sequence. (Output)

Comments

1. Automatic workspace usage is

   `FFTRB` 2`N` + 15 units, or
   `DFFTRB` 4`N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `F2TRB`/`DF2TRB`. The reference is

   ```
   CALL F2TRB (N, COEF, SEQ, WFFTR)
   ```

   The additional argument is

   WFFTR — Array of length 2`N` + 15 initialized by `FFTRI` (page 770). (Input)
   The initialization depends on `N`.

2. The routine `FFTRB` is most efficient when `N` is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `FFTRF`/`FFTRB` is used repeatedly with the same value of `N`, then call `FFTRI` (page 770) followed by repeated calls to `F2TRF`/`F2TRB`. This is more efficient than repeated calls to `FFTRF`/`FFTRB`.

Algorithm

The routine `FFTRB` is the unnormalized inverse of the routine `FFTRF` (page 764). This routine computes the discrete inverse Fourier transform of a real vector of size N. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when N is a product of small prime factors. If N satisfies this condition, then the computational effort is proportional to $N \log N$.

Specifically, given an N-vector c = `COEF`, `FFTRB` returns in s = `SEQ`, if N is even:

$$\begin{aligned} s_m = c_1 + (-1)^{(m-1)} c_N &+ 2\sum_{n=2}^{N/2} c_{2n-2} \cos \frac{\left[(n-1)(m-1)2\pi\right]}{N} \\ &-2\sum_{n=2}^{N/2} c_{2n-1} \sin \frac{\left[(n-1)(m-1)2\pi\right]}{N} \end{aligned}$$

If N is odd:

$$\begin{aligned} s_m = c_1 &+ 2\sum_{n=2}^{(N+1)/2} c_{2n-2} \cos \frac{\left[(n-1)(m-1)2\pi\right]}{N} \\ &-2\sum_{n=2}^{(N+1)/2} c_{2n-1} \sin \frac{\left[(n-1)(m-1)2\pi\right]}{N} \end{aligned}$$

The routine `FFTRB` is based on the inverse real FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

We compute the forward real FFT followed by the inverse operation. In this example, we first compute the Fourier transform

$$\hat{x} = \text{COEF}$$

of the vector x, where $x_j = (-1)^j$ for $j = 1$ to N. This vector

$$\hat{x}$$

is now input into `FFTRB` with the resulting output $s = Nx$, that is, $s_j = (-1)^j N$ for $j = 1$ to N.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       COEF(N), CONST, FLOAT, SEQ(N), TWOPI, X(N)
      INTRINSIC  FLOAT
      EXTERNAL   CONST, FFTRB, FFTRF, UMACH
C
      TWOPI = 2.0*CONST('PI')
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Fill the data vector
      DO 10  I=1, N
         X(I) = FLOAT((-1)**I)
   10 CONTINUE
C                                 Compute the forward transform of X
      CALL FFTRF (N, X, COEF)
C                                 Print results
      WRITE (NOUT,99994)
      WRITE (NOUT,99995)
99994 FORMAT (9X, 'Result after forward transform')
```

```
99995 FORMAT (9X, ’INDEX’, 5X, ’X’, 8X, ’COEF’)
      WRITE (NOUT,99996) (I, X(I), COEF(I), I=1,N)
99996 FORMAT (1X, I11, 5X, F5.2, 5X, F5.2)
C                                  Compute the backward transform of
C                                  COEF
      CALL FFTRB (N, COEF, SEQ)
C                                  Print results
      WRITE (NOUT,99997)
      WRITE (NOUT,99998)
99997 FORMAT (/, 9X, ’Result after backward transform’)
99998 FORMAT (9X, ’INDEX’, 4X, ’COEF’, 6X, ’SEQ’)
      WRITE (NOUT,99999) (I, COEF(I), SEQ(I), I=1,N)
99999 FORMAT (1X, I11, 5X, F5.2, 5X, F5.2)
      END
```

Output

```
Result after forward transform
INDEX     X        COEF
  1     -1.00     -1.00
  2      1.00     -1.00
  3     -1.00     -0.48
  4      1.00     -1.00
  5     -1.00     -1.25
  6      1.00     -1.00
  7     -1.00     -4.38

Result after backward transform
INDEX    COEF      SEQ
  1     -1.00     -7.00
  2     -1.00      7.00
  3     -0.48     -7.00
  4     -1.00      7.00
  5     -1.25     -7.00
  6     -1.00      7.00
  7     -4.38     -7.00
```

FFTRI/DFFTRI (Single/Double precision)

Compute parameters needed by FFTRF and FFTRB.

Usage

CALL FFTRI (N, WFFTR)

Arguments

N — Length of the sequence to be transformed.   (Input)

WFFTR — Array of length 2N + 15 containing parameters needed by FFTRF and FFTRB.   (Output)

Comments

Different WFFTR arrays are needed for different values of N.

Algorithm

The routine FFTRI initializes the routines FFTRF (page 764) and FFTRB (page 768). An efficient way to make multiple calls for the same *N* to routine FFTRF or FFTRB, is to use routine FFTRI for initialization. (In this case, replace FFTRF or FFTRB with F2TRF or F2TRB, respectively.) The routine FFTRI is based on the routine RFFTI in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute three distinct real FFTs by calling FFTRI once and then calling F2TRF three times.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, K, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, TWOPI, WFFTR(29), SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, F2TRF, FFTRI, UMACH
C
      TWOPI = 2.0*CONST(’PI’)
C                                Get output unit number
      CALL UMACH (2, NOUT)
C                                Set the work vector
      CALL FFTRI (N, WFFTR)
C
      DO 20  K=1, 3
C                                This loop fills out the data vector
C                                with a pure exponential signal
         DO 10  I=1, N
            SEQ(I) = COS(FLOAT(K*(I-1))*TWOPI/FLOAT(N))
   10 CONTINUE
C                                Compute the Fourier transform of SEQ
         CALL F2TRF (N, SEQ, COEF, WFFTR)
C                                Print results
         WRITE (NOUT,99998)
99998    FORMAT (/, 9X, ’INDEX’, 5X, ’SEQ’, 6X, ’COEF’)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99999    FORMAT (1X, I11, 5X, F5.2, 5X, F5.2)
C
   20 CONTINUE
      END
```

Output

```
INDEX     SEQ      COEF
  1      1.00      0.00
  2      0.62      3.50
  3     -0.22      0.00
  4     -0.90      0.00
  5     -0.90      0.00
  6     -0.22      0.00
  7      0.62      0.00
```

```
INDEX     SEQ       COEF
  1      1.00       0.00
  2     -0.22       0.00
  3     -0.90       0.00
  4      0.62       3.50
  5      0.62       0.00
  6     -0.90       0.00
  7     -0.22       0.00
```

```
INDEX     SEQ       COEF
1      1.00      0.00
2     -0.90      0.00
3      0.62      0.00
4     -0.22      0.00
5     -0.22      0.00
6      0.62      3.50
7     -0.90      0.00
```

FFTCF/DFFTCF (Single/Double precision)

Compute the Fourier coefficients of a complex periodic sequence.

Usage

CALL FFTCF (N, SEQ, COEF)

Arguments

N — Length of the sequence to be transformed. (Input)

SEQ — Complex array of length N containing the periodic sequence. (Input)

COEF — Complex array of length N containing the Fourier coefficients. (Output)

Comments

1. Automatic workspace usage is

   FFTCF 6 * N + 15 units, or
   DFFTCF 12 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of F2TCF/DF2TCF. The reference is

   CALL F2TCF (N, SEQ, COEF, WFFTC, CPY)

   The additional arguments are as follows:

   WFFTC — Real array of length 4 * N + 15 initialized by FFTCI (page 777). The initialization depends on N. (Input)

   CPY — Real array of length 2 * N. (Workspace)

2. The routine FFTCF is most efficient when N is the product of small primes.

3. The arrays COEF and SEQ may be the same.

4. If FFTCF/FFTCB is used repeatedly with the same value of N, then call FFTCI followed by repeated calls to F2TCF/F2TCB. This is more efficient than repeated calls to FFTCF/FFTCB.

Algorithm

The routine FFTCF computes the discrete complex Fourier transform of a complex vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*. This considerable savings has historically led people to refer to this algorithm as the “fast Fourier transform” or FFT.

Specifically, given an *N*-vector *x*, FFTCF returns in *c* = COEF

$$c_m = \sum_{n=1}^{N} x_n e^{-2\pi i(n-1)(m-1)/N}$$

Furthermore, a vector of Euclidean norm *S* is mapped into a vector of norm

$$\sqrt{N}S$$

Finally, note that we can invert the Fourier transform as follows:

$$x_n = \frac{1}{N}\sum_{m=1}^{N} c_m e^{2\pi i(m-1)(n-1)/N}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, one has the coefficients for a trigonometric interpolating polynomial to the data. An unnormalized inverse is implemented in FFTCB (page 774). FFTCF is based on the complex FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we input a pure exponential data vector and recover its Fourier series, which is a vector with all components zero except at the appropriate frequency where it has an *N*. Notice that the norm of the input vector is

$$\sqrt{N}$$

but the norm of the output vector is *N*.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       CONST, TWOPI
      COMPLEX    C, CEXP, COEF(N), H, SEQ(N)
      INTRINSIC  CEXP
      EXTERNAL   CONST, FFTCF, UMACH
C
      C     = (0.,1.)
      TWOPI = 2.0*CONST('PI')
C                                  Here we compute (2*pi*i/N)*3.
      H = (TWOPI*C/N)*3.
C                                  This loop fills out the data vector
C                                  with a pure exponential signal of
C                                  frequency 3.
      DO 10  I=1, N
         SEQ(I) = CEXP((I-1)*H)
   10 CONTINUE
C                                  Compute the Fourier transform of SEQ
      CALL FFTCF (N, SEQ, COEF)
C                                  Get output unit number and print
C                                  results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
99998 FORMAT (9X, 'INDEX', 8X, 'SEQ', 15X, 'COEF')
      WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99999 FORMAT (1X, I11, 5X,'(',F5.2,',',F5.2,')',
     &                 5X,'(',F5.2,',',F5.2,')')
      END
```

Output

```
INDEX        SEQ                COEF
  1     ( 1.00, 0.00)     ( 0.00, 0.00)
  2     (-0.90, 0.43)     ( 0.00, 0.00)
  3     ( 0.62,-0.78)     ( 0.00, 0.00)
  4     (-0.22, 0.97)     ( 7.00, 0.00)
  5     (-0.22,-0.97)     ( 0.00, 0.00)
  6     ( 0.62, 0.78)     ( 0.00, 0.00)
  7     (-0.90,-0.43)     ( 0.00, 0.00)
```

FFTCB/DFFTCB (Single/Double precision)

Compute the complex periodic sequence from its Fourier coefficients.

Usage

```
CALL FFTCB (N, COEF, SEQ)
```

Arguments

N — Length of the sequence to be transformed.   (Input)

COEF — Complex array of length `N` containing the Fourier coefficients.   (Input)

SEQ — Complex array of length `N` containing the periodic sequence.   (Output)

Comments

1. Automatic workspace usage is

   FFTCB 6 * N + 15 units, or
   DFFTCB 12 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of F2TCB/DF2TCB. The reference is

   ```
   CALL F2TCB (N, COEF, SEQ, WFFTC, CPY)
   ```

   The additional arguments are as follows:

   WFFTC — Real array of length 4 * N + 15 initialized by FFTCI (page 777). The initialization depends on N. (Input)

   CPY — Real array of length 2 * N. (Workspace)

2. The routine FFTCB is most efficient when N is the product of small primes.

3. The arrays COEF and SEQ may be the same.

4. If FFTCF/FFTCB is used repeatedly with the same value of N; then call FFTCI followed by repeated calls to F2TCF/F2TCB. This is more efficient than repeated calls to FFTCF/FFTCB.

Algorithm

The routine FFTCB computes the inverse discrete complex Fourier transform of a complex vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*. This considerable savings has historically led people to refer to this algorithm as the “fast Fourier transform” or FFT.

Specifically, given an *N*-vector *c* = COEF, FFTCB returns in *s* = SEQ

$$s_m = \sum_{n=1}^{N} c_n e^{2\pi i(n-1)(m-1)/N}$$

Furthermore, a vector of Euclidean norm *S* is mapped into a vector of norm

$$\sqrt{N}S$$

Finally, note that we can invert the inverse Fourier transform as follows:

$$c_n = \frac{1}{N}\sum_{m=1}^{N} s_m e^{-2\pi i(n-1)(m-1)/N}$$

This formula reveals the fact that, after properly normalizing the Fourier coefficients, one has the coefficients for a trigonometric interpolating polynomial to the data. FFTCB is based on the complex inverse FFT in FFTPACK.

The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we first compute the Fourier transform of the vector x, where x_j = j for j = 1 to N. Note that the norm of x is $(N[N + 1][2N + 1]/6)^{1/2}$, and hence, the norm of the transformed vector

$$\hat{x} = c$$

is $N([N + 1][2N + 1]/6)^{1/2}$. The vector

$$\hat{x}$$

is used as input into `FFTCB` with the resulting output $s = Nx$, that is, $s_j = jN$, for j = 1 to N.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      COMPLEX    CMPLX, SEQ(N), COEF(N), X(N)
      INTRINSIC  CMPLX
      EXTERNAL   FFTCB, FFTCF, UMACH
C                                 This loop fills out the data vector
C                                 with X(I)=I, I=1,N
      DO 10  I=1, N
         X(I) = CMPLX(I,0)
   10 CONTINUE
C                                 Compute the forward transform of X
      CALL FFTCF (N, X, COEF)
C                                 Compute the backward transform of
C                                 COEF
      CALL FFTCB (N, COEF, SEQ)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print results
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (I, X(I), COEF(I), SEQ(I), I=1,N)
99998 FORMAT (5X, ’INDEX’, 9X, ’INPUT’, 9X, ’FORWARD TRANSFORM’, 3X,
     &       ’BACKWARD TRANSFORM’)
99999 FORMAT (1X, I7, 7X,’(’,F5.2,’,’,F5.2,’)’,
     &                7X,’(’,F5.2,’,’,F5.2,’)’,
     &                7X,’(’,F5.2,’,’,F5.2,’)’)
      END
```

Output

```
INDEX      INPUT         FORWARD TRANSFORM   BACKWARD TRANSFORM
 1       ( 1.00, 0.00)       (28.00, 0.00)       ( 7.00, 0.00)
 2       ( 2.00, 0.00)       (-3.50, 7.27)       (14.00, 0.00)
 3       ( 3.00, 0.00)       (-3.50, 2.79)       (21.00, 0.00)
 4       ( 4.00, 0.00)       (-3.50, 0.80)       (28.00, 0.00)
 5       ( 5.00, 0.00)       (-3.50,-0.80)       (35.00, 0.00)
 6       ( 6.00, 0.00)       (-3.50,-2.79)       (42.00, 0.00)
 7       ( 7.00, 0.00)       (-3.50,-7.27)       (49.00, 0.00)
```

FFTCI/DFFTCI (Single/Double precision)

Compute parameters needed by FFTCF and FFTCB.

Usage

CALL FFTCI (N, WFFTC)

Arguments

N — Length of the sequence to be transformed. (Input)

WFFTC — Array of length 4N + 15 containing parameters needed by FFTCF and FFTCB. (Output)

Comments

Different WFFTC arrays are needed for different values of N.

Algorithm

The routine FFTCI initializes the routines FFTCF (page 772) and FFTCB (page 774). An efficient way to make multiple calls for the same N to IMSL routine FFTCF or FFTCB is to use routine FFTCI for initialization. (In this case, replace FFTCF or FFTCB with F2TCF or F2TCB, respectively.) The routine FFTCI is based on the routine CFFTI in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute a two-dimensional complex FFT by making one call to FFTCI followed by 2N calls to F2TCF.

```
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    N
      PARAMETER  (N=4)
C
      INTEGER    I, IR, IS, J, NOUT
      REAL       CONST, FLOAT, TWOPI, WFFTC(35), CPY(2*N)
      COMPLEX    CEXP, CMPLX, COEF(N,N), H, SEQ(N,N), TEMP
      INTRINSIC  CEXP, CMPLX, FLOAT
      EXTERNAL   CONST, F2TCF, FFTCI, UMACH
C
      TWOPI = 2.0*CONST('PI')
      IR    = 3
      IS    = 1
C                                  Here we compute e**(2*pi*i/N)
      TEMP = CMPLX(0.0,TWOPI/FLOAT(N))
      H    = CEXP(TEMP)
C                                  Fill SEQ with data
      DO 20  I=1, N
         DO 10  J=1, N
            SEQ(I,J) = H**((I-1)*(IR-1)+(J-1)*(IS-1))
   10 CONTINUE
   20 CONTINUE
C                                  Print out SEQ
C                                  Get output unit number
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99997)
      DO 30  I=1, N
         WRITE (NOUT,99998) (SEQ(I,J),J=1,N)
   30 CONTINUE
C                                  Set initialization vector
      CALL FFTCI (N, WFFTC)
C                                  Transform the columns of SEQ
      DO 40  I=1, N
         CALL F2TCF (N, SEQ(1,I), COEF(1,I), WFFTC, CPY)
   40 CONTINUE
C                                  Take transpose of the result
      DO 60  I=1, N
         DO 50  J=I + 1, N
            TEMP      = COEF(I,J)
            COEF(I,J) = COEF(J,I)
            COEF(J,I) = TEMP
   50  CONTINUE
   60 CONTINUE
C                                  Transform the columns of this result
      DO 70  I=1, N
         CALL F2TCF (N, COEF(1,I), SEQ(1,I), WFFTC, CPY)
   70 CONTINUE
C                                  Take transpose of the result
      DO 90  I=1, N
         DO 80  J=I + 1, N
            TEMP     = SEQ(I,J)
            SEQ(I,J) = SEQ(J,I)
            SEQ(J,I) = TEMP
   80  CONTINUE
   90 CONTINUE
C                                  Print results
      WRITE (NOUT,99999)
      DO 100  I=1, N
         WRITE (NOUT,99998) (SEQ(I,J),J=1,N)
  100 CONTINUE
C
99997 FORMAT (1X, 'The input matrix is below')
99998 FORMAT (1X, 4(' (',F5.2,',',F5.2,')'))
99999 FORMAT (/, 1X, 'Result of two-dimensional transform')
      END
```

Output

```
The input matrix is below
 ( 1.00, 0.00) ( 1.00, 0.00) ( 1.00, 0.00) ( 1.00, 0.00)
 (-1.00, 0.00) (-1.00, 0.00) (-1.00, 0.00) (-1.00, 0.00)
 ( 1.00, 0.00) ( 1.00, 0.00) ( 1.00, 0.00) ( 1.00, 0.00)
 (-1.00, 0.00) (-1.00, 0.00) (-1.00, 0.00) (-1.00, 0.00)

Result of two-dimensional transform
 ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
 ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
 (16.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
 ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00) ( 0.00, 0.00)
```

FSINT/DFSINT (Single/Double precision)

Compute the discrete Fourier sine transformation of an odd sequence.

Usage

```
CALL FSINT (N, SEQ, COEF)
```

Arguments

N — Length of the sequence to be transformed. It must be greater than 1.   (Input)

SEQ — Array of length N containing the sequence to be transformed.   (Input)

COEF — Array of length N + 1 containing the transformed sequence.   (Output)

Comments

1. Automatic workspace usage is

   FSINT INT(2.5 * N + 15) units, or
   DFSINT 5 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of F2INT/DF2INT. The reference is

   ```
   CALL F2INT (N, SEQ, COEF, WFSIN)
   ```

   The additional argument is

   WFSIN — Array of length INT(2.5 * N + 15) initialized by FSINI. The initialization depends on N.   (Input)

2. The routine FSINT is most efficient when N + 1 is the product of small primes.

3. The routine FSINT is its own (unnormalized) inverse. Applying FSINT twice will reproduce the original sequence multiplied by 2 * (N + 1).

4. The arrays COEF and SEQ may be the same, if SEQ is also dimensioned at least N + 1.

5. COEF (N + 1) is needed as workspace.

6. If FSINT is used repeatedly with the same value of N, then call FSINI (page 780) followed by repeated calls to F2INT. This is more efficient than repeated calls to FSINT.

Algorithm

The routine FSINT computes the discrete Fourier sine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* + 1 is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an *N*-vector *s* = SEQ, FSINT returns in *c* = COEF

$$c_m = 2\sum_{n=1}^{N} s_n \sin\left(\frac{mn\pi}{N+1}\right)$$

Finally, note that the Fourier sine transform is its own (unnormalized) inverse. The routine FSINT is based on the sine FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we input a pure sine wave as a data vector and recover its Fourier sine series, which is a vector with all components zero except at the appropriate frequency it has an *N*.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       COEF(N+1), CONST, FLOAT, PI, SIN, SEQ(N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, FSINT, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Fill the data vector SEQ
C                                  with a pure sine wave
      PI = CONST('PI')
      DO 10  I=1, N
         SEQ(I) = SIN(FLOAT(I)*PI/FLOAT(N+1))
   10 CONTINUE
C                                  Compute the transform of SEQ
      CALL FSINT (N, SEQ, COEF)
C                                  Print results
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99998 FORMAT (9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ        COEF
  1       0.38       8.00
  2       0.71       0.00
  3       0.92       0.00
  4       1.00       0.00
  5       0.92       0.00
  6       0.71       0.00
  7       0.38       0.00
```

FSINI/DFSINI (Single/Double precision)

Compute parameters needed by FSINT.

Usage

CALL FSINI (N, WFSIN)

Arguments

N — Length of the sequence to be transformed. N must be greater than 1.   (Input)

WFSIN — Array of length INT(2.5 * N + 15) containing parameters needed by FSINT.   (Output)

Comments

Different WFSIN arrays are needed for different values of N.

Algorithm

The routine FSINI initializes the routine FSINT (page 779). An efficient way to make multiple calls for the same *N* to IMSL routine FSINT, is to use routine FSINI for initialization. (In this case, replace FSINT with F2INT.) The routine FSINI is based on the routine SINTI in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute three distinct sine FFTs by calling FSINI once and then calling F2INT three times.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, K, NOUT
      REAL       COEF(N+1), CONST, FLOAT, PI, SIN, WFSIN(32), SEQ(N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, F2INT, FSINI, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Initialize the work vector WFSIN
      CALL FSINI (N, WFSIN)
C                                 Different frequencies of the same
C                                 wave will be transformed
      DO 20  K=1, 3
C                                 Fill the data vector SEQ
C                                 with a pure sine wave
         PI = CONST('PI')
         DO 10  I=1, N
            SEQ(I) = SIN(FLOAT(K*I)*PI/FLOAT(N+1))
   10    CONTINUE
C                                 Compute the transform of SEQ
```

```
         CALL F2INT (N, SEQ, COEF, WFSIN)
C                                 Print results
         WRITE (NOUT,99998)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
   20 CONTINUE
99998 FORMAT (/, 9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ       COEF
  1       0.38       8.00
  2       0.71       0.00
  3       0.92       0.00
  4       1.00       0.00
  5       0.92       0.00
  6       0.71       0.00
  7       0.38       0.00

INDEX      SEQ       COEF
  1       0.71       0.00
  2       1.00       8.00
  3       0.71       0.00
  4       0.00       0.00
  5      -0.71       0.00
  6      -1.00       0.00
  7      -0.71       0.00

INDEX      SEQ       COEF
  1       0.92       0.00
  2       0.71       0.00
  3      -0.38       8.00
  4      -1.00       0.00
  5      -0.38       0.00
  6       0.71       0.00
  7       0.92       0.00
```

FCOST/DFCOST (Single/Double precision)

Compute the discrete Fourier cosine transformation of an even sequence.

Usage

`CALL FCOST (N, SEQ, COEF)`

Arguments

N — Length of the sequence to be transformed. It must be greater than 1. (Input)

SEQ — Array of length `N` containing the sequence to be transformed. (Input)

COEF — Array of length `N` containing the transformed sequence. (Output)

Comments

1. Automatic workspace usage is

   FCOST 3 * N + 15 units, or
   DFCOST 6 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of F2OST/DF2OST. The reference is

   CALL F2OST (N, SEQ, COEF, WFCOS)

   The additional argument is

   WFCOS — Array of length 3 * N + 15 initialized by FCOSI (page 784). The initialization depends on N. (Input)

2. The routine FCOST is most efficient when N – 1 is the product of small primes.

3. The routine FCOST is its own (unnormalized) inverse. Applying FCOST twice will reproduce the original sequence multiplied by 2 * (N – 1).

4. The arrays COEF and SEQ may be the same.

5. If FCOST is used repeatedly with the same value of N, then call FCOSI followed by repeated calls to F2OST. This is more efficient than repeated calls to FCOST.

Algorithm

The routine FCOST computes the discrete Fourier cosine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm , which is most efficient when *N* – 1 is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an *N*-vector *s* = SEQ, FCOST returns in *c* = COEF

$$c_m = 2\sum_{n=2}^{N-1} s_n \cos\left[\frac{(m-1)(n-1)\pi}{N-1}\right] + s_1 + s_N(-1)^{(m-1)}$$

Finally, note that the Fourier cosine transform is its own (unnormalized) inverse. Two applications of FCOST to a vector *s* produces (2*N* – 2)*s*. The routine FCOST is based on the cosine FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we input a pure cosine wave as a data vector and recover its Fourier cosine series, which is a vector with all components zero except at the appropriate frequency it has an *N* – 1.

```
INTEGER    N
PARAMETER  (N=7)
```

```
C
      INTEGER    I, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, PI, SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, FCOST, UMACH
C
      CALL UMACH (2, NOUT)
C                                  Fill the data vector SEQ
C                                  with a pure cosine wave
      PI = CONST('PI')
      DO 10  I=1, N
         SEQ(I) = COS(FLOAT(I-1)*PI/FLOAT(N-1))
   10 CONTINUE
C                                  Compute the transform of SEQ
      CALL FCOST (N, SEQ, COEF)
C                                  Print results
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99998 FORMAT (9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ        COEF
  1       1.00        0.00
  2       0.87        6.00
  3       0.50        0.00
  4       0.00        0.00
  5      -0.50        0.00
  6      -0.87        0.00
  7      -1.00        0.00
```

FCOSI/DFCOSI (Single/Double precision)

Compute parameters needed by FCOST.

Usage

CALL FCOSI (N, WFCOS)

Arguments

N — Length of the sequence to be transformed. N must be greater than 1. (Input)

WFCOS — Array of length 3N + 15 containing parameters needed by FCOST. (Output)

Comments

Different WFCOS arrays are needed for different values of N.

Algorithm

The routine `FCOSI` initializes the routine `FCOST` (page 782). An efficient way to make multiple calls for the same *N* to IMSL routine `FCOST` is to use routine `FCOSI` for initialization. (In this case, replace `FCOST` with `F2OST`.) The routine `FCOSI` is based on the routine `COSTI` in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute three distinct cosine FFTs by calling `FCOSI` once and then calling `F2OST` three times.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, K, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, PI, WFCOS(36), SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, F2OST, FCOSI, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Initialize the work vector WFCOS
      CALL FCOSI (N, WFCOS)
C                                 Different frequencies of the same
C                                 wave will be transformed
      PI = CONST('PI')
      DO 20  K=1, 3
C                                 Fill the data vector SEQ
C                                 with a pure cosine wave
         DO 10  I=1, N
            SEQ(I) = COS(FLOAT(K*(I-1))*PI/FLOAT(N-1))
   10    CONTINUE
C                                 Compute the transform of SEQ
         CALL F2OST (N, SEQ, COEF, WFCOS)
C                                 Print results
         WRITE (NOUT,99998)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
   20 CONTINUE
99998 FORMAT (/, 9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ       COEF
  1       1.00       0.00
  2       0.87       6.00
  3       0.50       0.00
  4       0.00       0.00
  5      -0.50       0.00
  6      -0.87       0.00
  7      -1.00       0.00
```

```
INDEX      SEQ        COEF
  1        1.00       0.00
  2        0.50       0.00
  3       -0.50       6.00
  4       -1.00       0.00
  5       -0.50       0.00
  6        0.50       0.00
  7        1.00       0.00

INDEX      SEQ        COEF
  1        1.00       0.00
  2        0.00       0.00
  3       -1.00       0.00
  4        0.00       6.00
  5        1.00       0.00
  6        0.00       0.00
  7       -1.00       0.00
```

QSINF/DQSINF (Single/Double precision)

Compute the coefficients of the sine Fourier transform with only odd wave numbers.

Usage

CALL QSINF (N, SEQ, COEF)

Arguments

N — Length of the sequence to be transformed. (Input)

SEQ — Array of length N containing the sequence. (Input)

COEF — Array of length N containing the Fourier coefficients. (Output)

Comments

1. Automatic workspace usage is

   QSINF 3 * N + 15 units, or
   DQSINF 6 * N + 30 units.

   Workspace may be explicitly provided, if desired, by use of Q2INF/DQ2INF. The reference is

   CALL Q2INF (N, SEQ, COEF, WQSIN)

   The additional argument is

   WQSIN — Array of length 3 * N + 15 initialized by QSINI (page 790). The initialization depends on N. (Input)

2. The routine QSINF is most efficient when N is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `QSINF`/`QSINB` is used repeatedly with the same value of `N`, then call `QSINI` followed by repeated calls to `Q2INF`/`Q2INB`. This is more efficient than repeated calls to `QSINF`/`QSINB`.

Algorithm

The routine `QSINF` computes the discrete Fourier quarter sine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an N-vector *s* = `SEQ`, `QSINF` returns in *c* = `COEF`

$$c_m = 2\sum_{n=1}^{N-1} s_n \sin\left[\frac{(2m-1)n\pi}{2N}\right] + s_N(-1)^{m-1}$$

Finally, note that the Fourier quarter sine transform has an (unnormalized) inverse, which is implemented in the IMSL routine `QSINB`. The routine `QSINF` is based on the quarter sine FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we input a pure quarter sine wave as a data vector and recover its Fourier quarter sine series.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       COEF(N), CONST, FLOAT, PI, SIN, SEQ(N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, QSINF, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Fill the data vector SEQ
C                                 with a pure sine wave
      PI = CONST('PI')
      DO 10  I=1, N
         SEQ(I) = SIN(FLOAT(I)*(PI/2.0)/FLOAT(N))
   10 CONTINUE
C                                 Compute the transform of SEQ
      CALL QSINF (N, SEQ, COEF)
C                                 Print results
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99998 FORMAT (9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ        COEF
  1       0.22        7.00
  2       0.43        0.00
  3       0.62        0.00
  4       0.78        0.00
  5       0.90        0.00
  6       0.97        0.00
  7       1.00        0.00
```

QSINB/DQSINB (Single/Double precision)

Compute a sequence from its sine Fourier coefficients with only odd wave numbers.

Usage

```
CALL QSINB (N, COEF, SEQ)
```

Arguments

N — Length of the sequence to be transformed.   (Input)

COEF — Array of length `N` containing the Fourier coefficients.   (Input)

SEQ — Array of length `N` containing the sequence.   (Output)

Comments

1. Automatic workspace usage is

   `QSINB` 3 `*` `N` + 15 units, or
   `DQSINB` 6 `*` `N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `Q2INB`/`DQ2INB`. The reference is

   `CALL Q2INB (N, SEQ, COEF, WQSIN)`

   The additional argument is

   WQSIN — ray of length 3 `*` `N` + 15 initialized by `QSINI` (page 790). The initialization depends on `N`.(Input)

2. The routine `QSINB` is most efficient when `N` is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `QSINF`/`QSINB` is used repeatedly with the same value of `N`, then call `QSINI` followed by repeated calls to `Q2INF`/`Q2INB`. This is more efficient than repeated calls to `QSINF`/`QSINB`.

Algorithm

The routine QSINB computes the discrete (unnormalized) inverse Fourier quarter sine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an *N*-vector *c* = COEF, QSINB returns in *s* = SEQ

$$s_m = 4\sum_{n=1}^{N} c_n \sin\left(\frac{(2n-1)m\pi}{2N}\right)$$

Furthermore, a vector *x* of length *N* that is first transformed by QSINF (page 786) and then by QSINB will be returned by QSINB as 4*Nx*. The routine QSINB is based on the inverse quarter sine FFT in FFTPACK which was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we first compute the quarter wave sine Fourier transform *c* of the vector *x* where $x_n = n$ for *n* = 1 to *N*. We then compute the inverse quarter wave Fourier transform of *c* which is 4*Nx* = *s*.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       FLOAT, SEQ(N), COEF(N), X(N)
      INTRINSIC  FLOAT
      EXTERNAL   QSINB, QSINF, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Fill the data vector X
C                                 with X(I) = I, I=1,N
      DO 10  I=1, N
         X(I) = FLOAT(I)
   10 CONTINUE
C                                 Compute the forward transform of X
      CALL QSINF (N, X, COEF)
C                                 Compute the backward transform of W
      CALL QSINB (N, COEF, SEQ)
C                                 Print results
      WRITE (NOUT,99998)
      WRITE (NOUT,99999) (X(I), COEF(I), SEQ(I), I=1,N)
99998 FORMAT (5X, ’INPUT’, 5X, ’FORWARD TRANSFORM’, 3X, ’BACKWARD ’,
     &       ’TRANSFORM’)
99999 FORMAT (3X, F6.2, 10X, F6.2, 15X, F6.2)
      END
```

Output

```
INPUT     FORWARD TRANSFORM   BACKWARD TRANSFORM
1.00           39.88                28.00
2.00           -4.58                56.00
3.00            1.77                84.00
4.00           -1.00               112.00
5.00            0.70               140.00
6.00           -0.56               168.00
7.00            0.51               196.00
```

QSINI/DQSINI (Single/Double precision)

Compute parameters needed by QSINF and QSINB.

Usage

CALL QSINI (N, WQSIN)

Arguments

N — Length of the sequence to be transformed. (Input)

WQSIN — Array of length 3N + 15 containing parameters needed by QSINF and QSINB. (Output)

Comments

Different WQSIN arrays are needed for different values of N.

Algorithm

The routine QSINI initializes the routines QSINF (page 786) and QSINB (page 788). An efficient way to make multiple calls for the same *N* to IMSL routine QSINF or QSINB is to use routine QSINI for initialization. (In this case, replace QSINF or QSINB with Q2INF or Q2INB, respectively.) The routine QSINI is based on the routine SINQI in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute three distinct quarter sine transforms by calling QSINI once and then calling Q2INF three times.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, K, NOUT
      REAL       COEF(N), CONST, FLOAT, PI, SIN, WQSIN(36), SEQ(N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, Q2INF, QSINI, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Initialize the work vector WQSIN
```

```
      CALL QSINI (N, WQSIN)
C                                 Different frequencies of the same
C                                 wave will be transformed
      PI = CONST('PI')
      DO 20  K=1, 3
C                                 Fill the data vector SEQ
C                                 with a pure sine wave
         DO 10  I=1, N
            SEQ(I) = SIN(FLOAT((2*K-1)*I)*(PI/2.0)/FLOAT(N))
   10    CONTINUE
C                                 Compute the transform of SEQ
         CALL Q2INF (N, SEQ, COEF, WQSIN)
C                                 Print results
         WRITE (NOUT,99998)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
   20 CONTINUE
99998 FORMAT (/, 9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ       COEF
  1       0.22       7.00
  2       0.43       0.00
  3       0.62       0.00
  4       0.78       0.00
  5       0.90       0.00
  6       0.97       0.00
  7       1.00       0.00

INDEX      SEQ       COEF
  1       0.62       0.00
  2       0.97       7.00
  3       0.90       0.00
  4       0.43       0.00
  5      -0.22       0.00
  6      -0.78       0.00
  7      -1.00       0.00

INDEX      SEQ       COEF
  1       0.90       0.00
  2       0.78       0.00
  3      -0.22       7.00
  4      -0.97       0.00
  5      -0.62       0.00
  6       0.43       0.00
  7       1.00       0.00
```

QCOSF/DQCOSF (Single/Double precision)

Compute the coefficients of the cosine Fourier transform with only odd wave numbers.

Usage

```
CALL QCOSF (N, SEQ, COEF)
```

Arguments

N — Length of the sequence to be transformed. (Input)

SEQ — Array of length `N` containing the sequence. (Input)

COEF — Array of length `N` containing the Fourier coefficients. (Output)

Comments

1. Automatic workspace usage is

   `QCOSF` 3 `*` `N` + 15 units, or
   `DQCOSF` 6 `*` `N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `Q2OSF`/`DQ2OSF`. The reference is

   ```
   CALL Q2OSF (N, SEQ, COEF, WQCOS)
   ```

   The additional argument is

   WQCOS — Array of length 3 `*` `N` + 15 initialized by `QCOSI` (page 795). The initialization depends on `N`. (Input)

2. The routine `QCOSF` is most efficient when `N` is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `QCOSF`/`QCOSB` is used repeatedly with the same value of `N`, then call `QCOSI` followed by repeated calls to `Q2OSF`/`Q2OSB`. This is more efficient than repeated calls to `QCOSF`/`QCOSB`.

Algorithm

The routine `QCOSF` computes the discrete Fourier quarter cosine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*.

Specifically, given an *N*-vector *s* = `SEQ`, `QCOSF` returns in *c* = `COEF`

$$c_m = s_1 + 2\sum_{n=2}^{N} s_n \cos\left(\frac{(2m-1)(n-1)\pi}{2N}\right)$$

Finally, note that the Fourier quarter cosine transform has an (unnormalized) inverse which is implemented in `QCOSB`. The routine `QCOSF` is based on the quarter cosine FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we input a pure quarter cosine wave as a data vector and recover its Fourier quarter cosine series.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, PI, SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, QCOSF, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Fill the data vector SEQ
C                                  with a pure cosine wave
      PI = CONST('PI')
      DO 10  I=1, N
            SEQ(I) = COS(FLOAT(I-1)*(PI/2.0)/FLOAT(N))
   10    CONTINUE

C                                  Compute the transform of SEQ
         Call QCOSF (N, SEQ, COEF)
C                                   Print results
         WRITE (NOUT,99998)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
99998 FORMAT (9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ       COEF
  1       1.00       7.00
  2       0.97       0.00
  3       0.90       0.00
  4       0.78       0.00
  5       0.62       0.00
  6       0.43       0.00
  7       0.22       0.00
```

QCOSB/DQCOSB (Single/Double precision)

Compute a sequence from its cosine Fourier coefficients with only odd wave numbers.

Usage

`CALL QCOSB (N, COEF, SEQ)`

Arguments

N — Length of the sequence to be transformed.   (Input)

COEF — Array of length `N` containing the Fourier coefficients.   (Input)

SEQ — Array of length `N` containing the sequence.   (Output)

Comments

1. Automatic workspace usage is

   `QCOSB` 3 * `N` + 15 units, or
   `DQCOSB` 6 * `N` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `Q2OSB`/`DQ2OSB`. The reference is

   `CALL Q2OSB (N, COEF, SEQ, WQCOS)`

   The additional argument is

   WQCOS — Array of length 3 * `N` + 15 initialized by `QCOSI` (page 795). The initialization depends on `N`. (Input)

2. The routine `QCOSB` is most efficient when N is the product of small primes.

3. The arrays `COEF` and `SEQ` may be the same.

4. If `QCOSF`/`QCOSB` is used repeatedly with the same value of `N`, then call `QCOSI` followed by repeated calls to `Q2OSF`/`Q2OSB`. This is more efficient than repeated calls to `QCOSF`/`QCOSB`.

Algorithm

The routine `QCOSB` computes the discrete (unnormalized) inverse Fourier quarter cosine transform of a real vector of size *N*. The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when*N* is a product of small prime factors. If *N* satisfies this condition, then the computational effort is proportional to *N* log *N*. Specifically, given an *N*-vector *c* = `COEF`, `QCOSB` returns in *s* = `SEQ`

$$s_m = 4\sum_{n=1}^{N} c_n \cos\left(\frac{(2n-1)(m-1)\pi}{2N}\right)$$

Furthermore, a vector *x* of length *N* that is first transformed by `QCOSF` (page 791) and then by `QCOSB` will be returned by `QCOSB` as 4*Nx*. The routine `QCOSB` is based on the inverse quarter cosine FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we first compute the quarter wave cosine Fourier transform *c* of the vector *x*, where $x_n = n$ for *n* = 1 to *N*. We then compute the inverse quarter wave Fourier transform of *c* which is 4*Nx* = *s*.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, NOUT
      REAL       FLOAT, SEQ(N), COEF(N), X(N)
      INTRINSIC  FLOAT
      EXTERNAL   QCOSB, QCOSF, UMACH
C                                Get output unit number
      CALL UMACH (2, NOUT)
C                                Fill the data vector X
C                                with X(I) = I, I=1,N
      DO 10  I=1, N
         X(I) = FLOAT(I)
   10 CONTINUE
C                                Compute the forward transform of X
      CALL QCOSF (N, X, COEF)
C                                Compute the backward transform of
C                                COEF
      CALL QCOSB (N, COEF, SEQ)
C                                Print results
      WRITE (NOUT,99998)
      DO 20  I=1, N
         WRITE (NOUT,99999) X(I), COEF(I), SEQ(I)
   20 CONTINUE
99998 FORMAT (5X, 'INPUT', 5X, 'FORWARD TRANSFORM', 3X, 'BACKWARD ',
     &       'TRANSFORM')
99999 FORMAT (3X, F6.2, 10X, F6.2, 15X, F6.2)
      END
```

Output

```
INPUT     FORWARD TRANSFORM   BACKWARD TRANSFORM
1.00           31.12              28.00
2.00          -27.45              56.00
3.00           10.97              84.00
4.00           -9.00             112.00
5.00            4.33             140.00
6.00           -3.36             168.00
7.00            0.40             196.00
```

QCOSI/DQCOSI (Single/Double precision)

Compute parameters needed by QCOSF and QCOSB.

Usage

CALL QCOSI (N, WQCOS)

Arguments

N — Length of the sequence to be transformed.   (Input)

WQCOS — Array of length 3N + 15 containing parameters needed by QCOSF and QCOSB.   (Output)

Comments

Different WQCOS arrays are needed for different values of N.

Algorithm

The routine QCOSI initializes the routines QCOSF (page 791) and QCOSB (page 793). An efficient way to make multiple calls for the same *N* to IMSL routine QCOSF or QCOSB is to use routine QCOSI for initialization. (In this case, replace QCOSF or QCOSB with Q2OSF or Q2OSB , respectively.) The routine QCOSI is based on the routine COSQI in FFTPACK, which was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute three distinct quarter cosine transforms by calling QCOSI once and then calling Q2OSF three times.

```
      INTEGER    N
      PARAMETER  (N=7)
C
      INTEGER    I, K, NOUT
      REAL       COEF(N), CONST, COS, FLOAT, PI, WQCOS(36), SEQ(N)
      INTRINSIC  COS, FLOAT
      EXTERNAL   CONST, Q2OSF, QCOSI, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Initialize the work vector WQCOS
      CALL QCOSI (N, WQCOS)
C                                 Different frequencies of the same
C                                 wave will be transformed
      PI = CONST('PI')
      DO 20  K=1, 3
C                                 Fill the data vector SEQ
C                                 with a pure cosine wave
         DO 10  I=1, N
            SEQ(I) = COS(FLOAT((2*K-1)*(I-1))*(PI/2.0)/FLOAT(N))
   10    CONTINUE
C                                 Compute the transform of SEQ
         CALL Q2OSF (N, SEQ, COEF, WQCOS)
C                                 Print results
         WRITE (NOUT,99998)
         WRITE (NOUT,99999) (I, SEQ(I), COEF(I), I=1,N)
   20 CONTINUE
99998 FORMAT (/, 9X, 'INDEX', 6X, 'SEQ', 7X, 'COEF')
99999 FORMAT (1X, I11, 5X, F6.2, 5X, F6.2)
      END
```

Output

```
INDEX      SEQ       COEF
  1       1.00       7.00
  2       0.97       0.00
  3       0.90       0.00
  4       0.78       0.00
  5       0.62       0.00
  6       0.43       0.00
  7       0.22       0.00

INDEX      SEQ       COEF
  1       1.00       0.00
  2       0.78       7.00
  3       0.22       0.00
  4      -0.43       0.00
  5      -0.90       0.00
  6      -0.97       0.00
  7      -0.62       0.00

INDEX      SEQ       COEF
  1       1.00       0.00
  2       0.43       0.00
  3      -0.62       7.00
  4      -0.97       0.00
  5      -0.22       0.00
  6       0.78       0.00
  7       0.90       0.00
```

FFT2D/DFFT2D (Single/Double precision)

Compute Fourier coefficients of a complex periodic two-dimensional array.

Usage

```
CALL FFT2D (NRA, NCA, A, LDA, COEF, LDCOEF)
```

Arguments

NRA — The number of rows of A. (Input)

NCA — The number of columns of A. (Input)

A — NRA by NCA complex matrix containing the periodic data to be transformed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

COEF — NRA by NCA complex matrix containing the Fourier coefficients of A. (Output)

LDCOEF — Leading dimension of COEF exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   FFT2D 4 * (NRA + NCA) + 30 + 2 * MAX(NRA, NCA) + 2 units, or
   DFFT2D 8 * (NRA + NCA) + 60 + 4 * MAX(NRA, NCA) + 4 units.

   Workspace may be explicitly provided, if desired, by use of F2T2D/DF2T2D. The reference is

   ```
   CALL F2T2D (NRA, NCA, A, LDA, COEF, LDCOEF, WFF1,
               WFF2, CWK, CPY)
   ```

The additional arguments are as follows:

WFF1 — Real array of length 4 * NRA + 15 initialized by FFTCI. The initialization depends on NRA. (Input)

WFF2 — Real array of length 4 * NCA + 15 initialized by FFTCI. The initialization depends on NCA. (Input)

CWK — Complex array of length 1. (Workspace)

CPY — Real array of length 2 * MAX(NRA, NCA). (Workspace)

2. The routine FFT2D is most efficient when NRA and NCA are the product of small primes.

3. The arrays COEF and A may be the same.

4. If FFT2D/FFT2B is used repeatedly, with the same values for NRA and NCA, then use FFTCI (page 777) to fill WFF1(N = NRA) and WFF2(N = NCA). Follow this with repeated calls to F2T2D/F2T2B. This is more efficient than repeated calls to FFT2D/FFT2B.

Algorithm

The routine FFT2D computes the discrete complex Fourier transform of a complex two dimensional array of size (NRA = N) × (NCA = M). The method used is a variant of the Cooley-Tukey algorithm , which is most efficient when N and M are each products of small prime factors. If N and M satisfy this condition, then the computational effort is proportional to $N M \log N M$. This considerable savings has historically led people to refer to this algorithm as the "fast Fourier transform" or FFT.

Specifically, given an $N \times M$ array a, FFT2D returns in c = COEF

$$c_{jk} = \sum_{n=1}^{N} \sum_{m=1}^{M} a_{nm} e^{-2\pi i(j-1)(n-1)/N} e^{-2\pi i(k-1)(m-1)/M}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$\sqrt{NM}S$$

Finally, note that an unnormalized inverse is implemented in FFT2B (page 800). The routine FFT2D is based on the complex FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute the Fourier transform of the pure frequency input for a 5 × 4 array

$$a_{nm} = e^{2\pi i(n-1)2/N} e^{2\pi i(m-1)3/M}$$

for $1 \le n \le 5$ and $1 \le m \le 4$ using the IMSL routine FFT2D. The result

$$\hat{a} = c$$

has all zeros except in the (3, 4) position.

```
      INTEGER    I, IR, IS, J, LDA, LDCOEF, NCA, NRA
      REAL       CONST, FLOAT, TWOPI
      COMPLEX    A(5,4), C, CEXP, CMPLX, COEF(5,4), H
      CHARACTER  TITLE1*26, TITLE2*26
      INTRINSIC  CEXP, CMPLX, FLOAT
      EXTERNAL   CONST, FFT2D, WRCRN
C
      TITLE1 = 'The input matrix is below '
      TITLE2 = 'The output matrix is below'
      NRA    = 5
      NCA    = 4
      LDA    = 5
      LDCOEF = 5
      IR     = 3
      IS     = 4
C                                  Fill A with initial data
      TWOPI = 2.0*CONST('PI')
      C     = CMPLX(0.0,1.0)
      H     = CEXP(TWOPI*C)
      DO 10  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = CEXP(TWOPI*C*((FLOAT((I-1)*(IR-1))/FLOAT(NRA)+
     &               FLOAT((J-1)*(IS-1))/FLOAT(NCA))))
   10 CONTINUE
C
      CALL WRCRN (TITLE1, NRA, NCA, A, LDA, 0)
C
      CALL FFT2D (NRA, NCA, A, LDA, COEF, LDCOEF)
C
      CALL WRCRN (TITLE2, NRA, NCA, COEF, LDCOEF, 0)
C
      END
```

Output

```
                The input matrix is below
                1                 2                 3                 4
1  ( 1.000, 0.000)  ( 0.000,-1.000)  (-1.000, 0.000)  ( 0.000, 1.000)
2  (-0.809, 0.588)  ( 0.588, 0.809)  ( 0.809,-0.588)  (-0.588,-0.809)
3  ( 0.309,-0.951)  (-0.951,-0.309)  (-0.309, 0.951)  ( 0.951, 0.309)
4  ( 0.309, 0.951)  ( 0.951,-0.309)  (-0.309,-0.951)  (-0.951, 0.309)
5  (-0.809,-0.588)  (-0.588, 0.809)  ( 0.809, 0.588)  ( 0.588,-0.809)

                 The Output matrix is below
                1                 2                 3                 4
1  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)
2  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)
3  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)  ( 20.00,  0.00)
4  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)
5  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)  (  0.00,  0.00)
```

FFT2B/DFFT2B (Single/Double precision)

Compute the inverse Fourier transform of a complex periodic two-dimensional array.

Usage

```
CALL FFT2B (NRCOEF, NCCOEF, COEF, LDCOEF, A, LDA)
```

Arguments

NRCOEF — The number of rows of COEF. (Input)

NCCOEF — The number of columns of COEF. (Input)

COEF — NRCOEF by NCCOEF complex array containing the Fourier coefficients to be transformed. (Input)

LDCOEF — Leading dimension of COEF exactly as specified in the dimension statement of the calling program. (Input)

A — NRCOEF by NCCOEF complex array containing the Inverse Fourier coefficients of COEF. (Output)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   FFT2B 4 * (NRCOEF + NCCOEF) + 32 + 2 * MAX(NRCOEF, NCCOEF) units, or

   DFFT2B 8 * (NRCOEF + NCCOEF) + 64 + 4 * MAX(NRCOEF, NCCOEF) units.

   Workspace may be explicitly provided, if desired, by use of F2T2B/DF2T2B. The reference is

   ```
   CALL F2T2B (NRCOEF, NCCOEF, A, LDA, COEF, LDCOEF,
                WFF1, WFF2, CWK, CPY)
   ```

   The additional arguments are as follows:

   WFF1 — Real array of length 4 * NRCOEF + 15 initialized by FFTCI (page 777). The initialization depends on NRCOEF. (Input)

   WFF2 — Real array of length 4 * NCCOEF + 15 initialized by FFTCI. The initialization depends on NCCOEF. (Input)

   CWK — Complex array of length 1. (Workspace)

   CPY — Real array of length 2 * MAX(NRCOEF, NCCOEF). (Workspace)

2. The routine FFT2B is most efficient when NRCOEF and NCCOEF are the product of small primes.

3. The arrays COEF and A may be the same.

4. If FFT2D/FFT2B is used repeatedly, with the same values for NRCOEF and NCCOEF, then use FFTCI to fill WFF1(N = NRCOEF) and WFF2(N = NCCOEF). Follow this with repeated calls to F2T2D/F2T2B. This is more efficient than repeated calls to FFT2D/FFT2B.

Algorithm

The routine FFT2B computes the inverse discrete complex Fourier transform of a complex two-dimensional array of size (NRCOEF = N) × (NCCOEF = M). The method used is a variant of the Cooley-Tukey algorithm , which is most efficient when N and M are both products of small prime factors. If N and M satisfy this condition, then the computational effort is proportional to $N M \log N M$. This considerable savings has historically led people to refer to this algorithm as the “fast Fourier transform” or FFT.

Specifically, given an $N \times M$ array c = COEF, FFT2B returns in a

$$a_{jk} = \sum_{n=1}^{N} \sum_{m=1}^{M} c_{nm} e^{2\pi i(j-1)(n-1)/N} e^{2\pi i(k-1)(m-1)/M}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$S\sqrt{NM}$$

Finally, note that an unnormalized inverse is implemented in FFT2D (page 797). The routine FFT2B is based on the complex FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we first compute the Fourier transform of the 5×4 array

$$x_{nm} = n + 5(m-1)$$

for $1 \le n \le 5$ and $1 \le m \le 4$ using the IMSL routine FFT2D. The result

$$\hat{x} = c$$

is then inverted by a call to FFT2B. Note that the result is an array a satisfying $a = (5)(4)x = 20x$. In general, FFT2B is an unnormalized inverse with expansion factor $N M$.

```
      INTEGER    LDA, LDCOEF, M, N, NCA, NRA
      COMPLEX    CMPLX, X(5,4), A(5,4), COEF(5,4)
      CHARACTER  TITLE1*26, TITLE2*26, TITLE3*26
      INTRINSIC  CMPLX
      EXTERNAL   FFT2B, FFT2D, WRCRN
C
      TITLE1 = 'The input matrix is below '
      TITLE2 = 'After FFT2D                ’
      TITLE3 = 'After FFT2B                ’
      NRA    = 5
      NCA    = 4
      LDA    = 5
      LDCOEF = 5
C                                 Fill X with initial data
      DO 20  N=1, NRA
         DO 10  M=1, NCA
            X(N,M) = CMPLX(FLOAT(N+5*M-5),0.0)
   10    CONTINUE
   20 CONTINUE
C
      CALL WRCRN (TITLE1, NRA, NCA, X, LDA, 0)
C
      CALL FFT2D (NRA, NCA, X, LDA, COEF, LDCOEF)
C
      CALL WRCRN (TITLE2, NRA, NCA, COEF, LDCOEF, 0)
C
      CALL FFT2B (NRA, NCA, COEF, LDCOEF, A, LDA)
C
      CALL WRCRN (TITLE3, NRA, NCA, A, LDA, 0)
C
      END
```

Output

```
                The input matrix is below
                1                 2                 3                 4
1  (  1.00,  0.00)  (  6.00,  0.00)  ( 11.00,  0.00)  ( 16.00,  0.00)
2  (  2.00,  0.00)  (  7.00,  0.00)  ( 12.00,  0.00)  ( 17.00,  0.00)
3  (  3.00,  0.00)  (  8.00,  0.00)  ( 13.00,  0.00)  ( 18.00,  0.00)
4  (  4.00,  0.00)  (  9.00,  0.00)  ( 14.00,  0.00)  ( 19.00,  0.00)
5  (  5.00,  0.00)  ( 10.00,  0.00)  ( 15.00,  0.00)  ( 20.00,  0.00)

                             After FFT2D
                1                 2                 3                 4
1  ( 210.0,   0.0)  ( -50.0,  50.0)  ( -50.0,   0.0)  ( -50.0, -50.0)
2  ( -10.0,  13.8)  (   0.0,   0.0)  (   0.0,   0.0)  (   0.0,   0.0)
3  ( -10.0,   3.2)  (   0.0,   0.0)  (   0.0,   0.0)  (   0.0,   0.0)
4  ( -10.0,  -3.2)  (   0.0,   0.0)  (   0.0,   0.0)  (   0.0,   0.0)
5  ( -10.0, -13.8)  (   0.0,   0.0)  (   0.0,   0.0)  (   0.0,   0.0)

                             After FFT2B
                1                 2                 3                 4
1  (  20.0,   0.0)  ( 120.0,   0.0)  ( 220.0,   0.0)  ( 320.0,   0.0)
2  (  40.0,   0.0)  ( 140.0,   0.0)  ( 240.0,   0.0)  ( 340.0,   0.0)
3  (  60.0,   0.0)  ( 160.0,   0.0)  ( 260.0,   0.0)  ( 360.0,   0.0)
4  (  80.0,   0.0)  ( 180.0,   0.0)  ( 280.0,   0.0)  ( 380.0,   0.0)
5  ( 100.0,   0.0)  ( 200.0,   0.0)  ( 300.0,   0.0)  ( 400.0,   0.0)
```

FFT3F/DFFT3F (Single/Double precision)

Compute Fourier coefficients of a complex periodic three-dimensional array.

Usage

```
CALL FFT3F (N1, N2, N3, A, LDA, MDA, B, LDB, MDB)
```

Arguments

N1 — Limit on the first subscript of matrices A and B.   (Input)

N2 — Limit on the second subscript of matrices A and B.   (Input)

N3 — Limit on the third subscript of matrices A and B.   (Input)

A — Three-dimensional complex matrix containing the data to be transformed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

MDA — Middle dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

B — Three-dimensional complex matrix containing the Fourier coefficients of A. (Output)
The matrices A and B may be the same.

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

MDB — Middle dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   FFT3F 4 * (N1 + N2 + N3) + 2 * MAX(N1, N2, N3) + 45 units, or
   DFFT3F 8 * (N1 + N2 + N3) + 4 * MAX(N1, N2, N3) + 90 units.

   Workspace may be explicitly provided, if desired, by use of F2T3F/DF2T3F. The reference is

   ```
   CALL F2T3F (N1, N2, N3, A, LDA, MDA, B, LDB, MDB,
               WFF1, WFF2, WFF3, CPY)
   ```

   The additional arguments are as follows:

   WFF1 — Real array of length 4 * N1 + 15 initialized by FFTCI (page 777). The initialization depends on N1.   (Input)

   WFF2 — Real array of length 4 * N2 + 15 initialized by FFTCI. The initialization depends on N2.   (Input)

WFF3 — Real array of length 4 * N3 + 15 initialized by FFTCI. The initialization depends on N3.   (Input)

CPY — Real array of size 2 * MAX(N1, N2, N3).   (Workspace)

2. The routine FFT3F is most efficient when N1, N2, and N3 are the product of small primes.

3. If FFT3F/FFT3B is used repeatedly with the same values for N1, N2 and N3, then use FFTCI to fill WFF1(N = N1), WFF2(N = N2), and WFF3(N = N3). Follow this with repeated calls to F2T3F/F2T3B. This is more efficient than repeated calls to FFT3F/FFT3B.

Algorithm

The routine FFT3F computes the forward discrete complex Fourier transform of a complex three-dimensional array of size (N1 = N) × (N2 = M) × (N3 = L). The method used is a variant of the Cooley-Tukey algorithm , which is most efficient when N, M, and L are each products of small prime factors. If N, M, and L satisfy this condition, then the computational effort is proportional to $N\,M\,L \log N\,M\,L$. This considerable savings has historically led people to refer to this algorithm as the “fast Fourier transform” or FFT.

Specifically, given an $N \times M \times L$ array a, FFT3F returns in c = COEF

$$c_{jkl} = \sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{l=1}^{L} a_{nml} e^{-2\pi i(j-1)(n-1)/N} e^{-2\pi i(k-1)(m-1)/M} e^{-2\pi i(k-1)(l-1)/L}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$\sqrt{NML}S$$

Finally, note that an unnormalized inverse is implemented in FFT3B. The routine FFT3F is based on the complex FFT in FFTPACK. The package FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute the Fourier transform of the pure frequency input for a 2 × 3 × 4 array

$$a_{nml} = e^{2\pi i(n-1)1/2} e^{2\pi i(m-1)2/3} e^{2\pi i(l-1)2/4}$$

for $1 \le n \le 2$, $1 \le m \le 3$, and $1 \le l \le 4$ using the IMSL routine FFT3F. The result

$$\hat{a} = c$$

has all zeros except in the (2, 3, 3) position.

```
      INTEGER    LDA, LDB, MDA, MDB, NDA, NDB
      PARAMETER  (LDA=2, LDB=2, MDA=3, MDB=3, NDA=4, NDB=4)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, J, K, L, M, N, N1, N2, N3, NOUT
      REAL       PI
      COMPLEX    A(LDA,MDA,NDA), B(LDB,MDB,NDB), C, H
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  CEXP, CMPLX
      COMPLEX    CEXP, CMPLX
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   FFT3F, UMACH
C                                 SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   CONST
      REAL       CONST
C                                 Get output unit number
      CALL UMACH (2, NOUT)
      N1 = 2
      N2 = 3
      N3 = 4
      PI = CONST('PI')
      C  = CMPLX(0.0,2.0*PI)
C                                 Set array A
      DO 30  N=1, 2
         DO 20  M=1, 3
            DO 10  L=1, 4
               H        = C*(N-1)*1/2 + C*(M-1)*2/3 + C*(L-1)*2/4
               A(N,M,L) = CEXP(H)
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
C
      CALL FFT3F (N1, N2, N3, A, LDA, MDA, B, LDB, MDB)
C
      WRITE (NOUT,99996)
      DO 50  I=1, 2
         WRITE (NOUT,99998) I
         DO 40  J=1, 3
            WRITE (NOUT,99999) (A(I,J,K),K=1,4)
   40    CONTINUE
   50 CONTINUE
C
      WRITE (NOUT,99997)
      DO 70  I=1, 2
         WRITE (NOUT,99998) I
         DO 60  J=1, 3
            WRITE (NOUT,99999) (B(I,J,K),K=1,4)
   60    CONTINUE
   70 CONTINUE
C
99996 FORMAT (13X, 'The input for FFT3F is')
99997 FORMAT (/, 13X, 'The results from FFT3F are')
99998 FORMAT (/, ' Face no. ', I1)
99999 FORMAT (1X, 4('(',F6.2,',',F6.2,')',3X))
      END
```

Output

```
             The input for FFT3F is

Face no. 1
(  1.00,  0.00)   ( -1.00,  0.00)   (  1.00,  0.00)   ( -1.00,  0.00)
( -0.50, -0.87)   (  0.50,  0.87)   ( -0.50, -0.87)   (  0.50,  0.87)
( -0.50,  0.87)   (  0.50, -0.87)   ( -0.50,  0.87)   (  0.50, -0.87)
```

```
Face no. 2
( -1.00,  0.00)   (  1.00,  0.00)   ( -1.00,  0.00)   (  1.00,  0.00)
(  0.50,  0.87)   ( -0.50, -0.87)   (  0.50,  0.87)   ( -0.50, -0.87)
(  0.50, -0.87)   ( -0.50,  0.87)   (  0.50, -0.87)   ( -0.50,  0.87)

The results from FFT3F are

Face no. 1
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)

Face no. 2
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   ( 24.00,  0.00)   (  0.00,  0.00)
```

FFT3B/DFFT3B (Single/Double precision)

Compute the inverse Fourier transform of a complex periodic three-dimensional array.

Usage

```
CALL FFT3B (N1, N2, N3, A, LDA, MDA, B, LDB, MDB)
```

Arguments

N1 — Limit on the first subscript of matrices A and B.   (Input)

N2 — Limit on the second subscript of matrices A and B.   (Input)

N3 — Limit on the third subscript of matrices A and B.   (Input)

A — Three-dimensional complex matrix containing the data to be transformed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

MDA — Middle dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

B — Three-dimensional complex matrix containing the inverse Fourier coefficients of A.   (Output)
The matrices A and B may be the same.

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

MDB — Middle dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   FFT3B 4 * (N1 + N2 + N3) + 2 * MAX(N1, N2, N3) + 45 units, or
   DFFT3B 8 * (N1 + N2 + N3) + 4 * MAX(N1, N2, N3) + 90 units.

   Workspace may be explicitly provided, if desired, by use of F2T3B/DF2T3B. The reference is

   ```
   CALL F2T3B (N1, N2, N3, A, LDA, MDA, B, LDB, MDB,
               WFF1, WFF2, WFF3, CPY)
   ```

   The additional arguments are as follows:

   WFF1 — Real array of length 4 * N1 + 15 initialized by FFTCI (page 777). The initialization depends on N1. (Input)

   WFF2 — Real array of length 4 * N2 + 15 initialized by FFTCI. The initialization depends on N2. (Input)

   WFF3 — Real array of length 4 * N3 + 15 initialized by FFTCI. The initialization depends on N3. (Input)

   CPY — Real array of size 2 * MAX(N1, N2, N3). (Workspace)

2. The routine FFT3B is most efficient when N1, N2, and N3 are the product of small primes.

3. If FFT3F/FFT3B is used repeatedly with the same values for N1, N2 and N3, then use FFTCI to fill WFF1(N = N1), WFF2(N = N2), and WFF3(N = N3). Follow this with repeated calls to F2T3F/F2T3B. This is more efficient than repeated calls to FFT3F/FFT3B.

Algorithm

The routine FFT3B computes the inverse discrete complex Fourier transform of a complex three-dimensional array of size (N1 = *N*) × (N2 = *M*) × (N3 = *L*). The method used is a variant of the Cooley-Tukey algorithm, which is most efficient when *N*, *M*, and *L* are each products of small prime factors. If *N*, *M*, and *L* satisfy this condition, then the computational effort is proportional to *N M L* log *N M L*. This considerable savings has historically led people to refer to this algorithm as the "fast Fourier transform" or FFT.

Specifically, given an $N \times M \times L$ array a, FFT3B returns in b

$$b_{jkl} \sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{l=1}^{L} a_{nml} e^{2\pi i(j-1)(n-1)/N} e^{2\pi i(k-1)(m-1)/M} e^{2\pi i(k-1)(l-1)/L}$$

Furthermore, a vector of Euclidean norm S is mapped into a vector of norm

$$\sqrt{NML}S$$

Finally, note that an unnormalized inverse is implemented in FFT3F. The routine FFT3B is based on the complex FFT in FFTPACK. The package

FFTPACK was developed by Paul Swarztrauber at the National Center for Atmospheric Research.

Example

In this example, we compute the Fourier transform of the $2 \times 3 \times 4$ array

$$x_{nml} = n + 2(m-1) + 2(3)(l-1)$$

for $1 \le n \le 2$, $1 \le m \le 3$, and $1 \le l \le 4$ using the IMSL routine FFT3F. The result

$$a = \hat{x}$$

is then inverted using FFT3B. Note that the result is an array b satisfying $b = 2(3)(4)x = 24x$. In general, FFT3B is an unnormalized inverse with expansion factor $N\,M\,L$.

```
      INTEGER    LDA, LDB, MDA, MDB, NDA, NDB
      PARAMETER  (LDA=2, LDB=2, MDA=3, MDB=3, NDA=4, NDB=4)
C                                 SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I, J, K, L, M, N, N1, N2, N3, NOUT
      COMPLEX    A(LDA,MDA,NDA), B(LDB,MDB,NDB), X(LDB,MDB,NDB)
C                                 SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  CEXP, CMPLX
      COMPLEX    CEXP, CMPLX
C                                 SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   FFT3B, FFT3F, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
      N1 = 2
      N2 = 3
      N3 = 4
C                                 Set array X
      DO 30  N=1, 2
         DO 20  M=1, 3
            DO 10  L=1, 4
               X(N,M,L) = N + 2*(M-1) + 2*3*(L-1)
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
C
      CALL FFT3F (N1, N2, N3, X, LDA, MDA, A, LDA, MDA)
      CALL FFT3B (N1, N2, N3, A, LDA, MDA, B, LDB, MDB)
C
      WRITE (NOUT,99996)
      DO 50  I=1, 2
         WRITE (NOUT,99998) I
         DO 40  J=1, 3
            WRITE (NOUT,99999) (X(I,J,K),K=1,4)
   40    CONTINUE
   50 CONTINUE
C
      WRITE (NOUT,99997)
      DO 70  I=1, 2
         WRITE (NOUT,99998) I
         DO 60  J=1, 3
            WRITE (NOUT,99999) (A(I,J,K),K=1,4)
   60    CONTINUE
   70 CONTINUE
C
      WRITE (NOUT, 99995)
      DO 90  I=1, 2
         WRITE (NOUT,99998) I
         DO 80  J=1, 3
            WRITE (NOUT,99999) (B(I,J,K),K=1,4)
   80    CONTINUE
   90 CONTINUE
99995 FORMAT (13X, ’The unnormalized inverse is’)
99996 FORMAT (13X, ’The input for FFT3F is’)
99997 FORMAT (/, 13X, ’The results from FFT3F are’)
99998 FORMAT (/, ’ Face no. ’, I1)
99999 FORMAT (1X, 4(’(’,F6.2,’,’,F6.2,’)’,3X))
      END
```

Output

```
            The input for FFT3F is

Face no. 1
(  1.00,  0.00)   (  7.00,  0.00)   ( 13.00,  0.00)   ( 19.00,  0.00)
(  3.00,  0.00)   (  9.00,  0.00)   ( 15.00,  0.00)   ( 21.00,  0.00)
(  5.00,  0.00)   ( 11.00,  0.00)   ( 17.00,  0.00)   ( 23.00,  0.00)

Face no. 2
(  2.00,  0.00)   (  8.00,  0.00)   ( 14.00,  0.00)   ( 20.00,  0.00)
(  4.00,  0.00)   ( 10.00,  0.00)   ( 16.00,  0.00)   ( 22.00,  0.00)
(  6.00,  0.00)   ( 12.00,  0.00)   ( 18.00,  0.00)   ( 24.00,  0.00)

The results from FFT3F are

Face no. 1
(300.00,  0.00)   (-72.00, 72.00)   (-72.00,  0.00)   (-72.00,-72.00)
(-24.00, 13.86)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(-24.00,-13.86)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)

Face no. 2
(-12.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)
(  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)   (  0.00,  0.00)

The unnormalized inverse is

Face no. 1
( 24.00,  0.00)   (168.00,  0.00)   (312.00,  0.00)   (456.00,  0.00)
( 72.00,  0.00)   (216.00,  0.00)   (360.00,  0.00)   (504.00,  0.00)
(120.00,  0.00)   (264.00,  0.00)   (408.00,  0.00)   (552.00,  0.00)

Face no. 2
( 48.00,  0.00)   (192.00,  0.00)   (336.00,  0.00)   (480.00,  0.00)
( 96.00,  0.00)   (240.00,  0.00)   (384.00,  0.00)   (528.00,  0.00)
(144.00,  0.00)   (288.00,  0.00)   (432.00,  0.00)   (576.00,  0.00)
```

RCONV/DRCONV (Single/Double precision)

Compute the convolution of two real vectors.

Usage

```
CALL RCONV (IDO, NX, X, NY, Y, IPAD, NZ, Z, ZHAT)
```

Arguments

IDO — Flag indicating the usage of RCONV. (Input)

IDO	**Usage**
0	If this is the only call to RCONV.

If RCONV is called multiple times in sequence with the same NX, NY, and IPAD, IDO should be set to

1	on the first call
2	on the intermediate calls
3	on the final call.

NX — Length of the vector X. (Input)

X — Real vector of length NX. (Input)

NY — Length of the vector Y. (Input)

Y — Real vector of length NY. (Input)

IPAD — IPAD should be set to zero for periodic data or to one for nonperiodic data. (Input)

NZ — Length of the vector Z. (Input/Output)
Upon input: When IPAD is zero, NZ must be at least MAX(NX, NY). When IPAD is one, NZ must be greater than or equal to the smallest integer greater than or equal to (NX + NY −1) of the form (2^{α}) * (3^{β}) * (5^{γ}) where alpha, beta, and gamma are nonnegative integers. Upon output, the value for NZ that was used by RCONV.

Z — Real vector of length NZ ontaining the convolution of X and Y. (Output)

ZHAT — Real vector of length NZ containing the discrete Fourier transform of Z. (Output)

Comments

1. Automatic workspace usage is

   RCONV 4 * NZ + 15 units, or
   DRCONV 8 * NZ + 30 units.

   Workspace may be explicitly provided, if desired, by use of R2ONV/DR2ONV. The reference is

```
CALL R2ONV (IDO, NX, X, NY, Y, IPAD, NZ, Z, ZHAT
               XWK, YWK, WK)
```

The additional arguments are as follows:

XWK — Real work array of length `NZ`.

YWK — Real work array of length `NZ`.

WK — Real work arrary of length 2 `*` `NZ` + 15.

2. Informational error

Type	Code	
4	1	The length of the vector `Z` must be large enough to hold the results. An acceptable length is returned in `NZ`.

Algorithm

The routine `RCONV` computes the discrete convolution of two sequences *x* and *y*. More precisely, let n_x be the length of *x* and n_y denote the length of *y*. If a circular convolution is desired, then `IPAD` must be set to zero. We set

$$n_z := \max\{n_x, n_y\}$$

and we pad out the shorter vector with zeroes. Then, we compute

$$z_i = \sum_{j=1}^{n_z} x_{i-j+1} y_j$$

where the index on *x* is interpreted as a positive number between 1 and n_z, modulo n_z.

The technique used to compute the z_i's is based on the fact that the (complex discrete) Fourier transform maps convolution into multiplication. Thus, the Fourier transform of *z* is given by

$$\hat{z}(n) = \hat{x}(n)\hat{y}(n)$$

where

$$\hat{z}(n) = \sum_{m=1}^{n_z} z_m e^{-2\pi i(m-1)(n-1)/n_z}$$

The technique used here to compute the convolution is to take the discrete Fourier transform of *x* and *y*, multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is a product of small primes if `IPAD` is set to zero. If n_z is a product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If `IPAD` is one, then a good value is chosen for n_z so that the Fourier transforms are efficient and $n_z \ge n_x + n_y - 1$. This will mean that both vectors will be padded with zeroes.

We point out that no complex transforms of *x* or *y* are taken since both sequences are real, we can take real transforms and simulate the complex transform above. This can produce a savings of a factor of six in time as well as save space over using the complex transform.

Example

In this example, we compute both a periodic and a non-periodic convolution. The idea here is that one can compute a moving average of the type found in digital filtering using this routine. The averaging operator in this case is especially simple and is given by averaging five consecutive points in the sequence. The periodic case tries to recover a noisy sin function by averaging five nearby values. The nonperiodic case tries to recover the values of an exponential function contaminated by noise. The large error for the last value printed has to do with the fact that the convolution is averaging the zeroes in the “pad” rather than function values. Notice that the signal size is 100, but we only report the errors at ten points.

```
      INTEGER    NFLTR, NY
      PARAMETER  (NFLTR=5, NY=100)
C
      INTEGER    I, K, MOD, NOUT, NZ
      REAL       ABS, CONST, EXP, F1, F2, FLOAT, FLTR(NFLTR),
     &           FLTRER, ORIGER, RNUNF, SIN, TOTAL1, TOTAL2, TWOPI, X,
     &           Y(NY), Z(2*(NFLTR+NY-1)), ZHAT(2*(NFLTR+NY-1))
      INTRINSIC  ABS, EXP, FLOAT, MOD, SIN
      EXTERNAL   CONST, RCONV, RNSET, RNUNF, UMACH
C                                DEFINE FUNCTIONS
      F1(X) = SIN(X)
      F2(X) = EXP(X)
C
      CALL RNSET (1234579)
      CALL UMACH (2, NOUT)
      TWOPI = 2.0*CONST('PI')
C                                 SET UP THE FILTER
      DO 10  I=1, 5
         FLTR(I) = 0.2
   10 CONTINUE
C                                 SET UP Y-VECTOR FOR THE PERIODIC
C                                 CASE.
      DO 20  I=1, NY
         X    = TWOPI*FLOAT(I-1)/FLOAT(NY-1)
         Y(I) = F1(X) + 0.5*RNUNF() - 0.25
   20 CONTINUE
C                                 CALL THE CONVOLUTION ROUTINE FOR THE
C                                 PERIODIC CASE.
      NZ = 2*(NFLTR+NY-1)
      CALL RCONV (0, NFLTR, FLTR, NY, Y, 0, NZ, Z, ZHAT)
C                                 PRINT RESULTS
      WRITE (NOUT,99993)
      WRITE (NOUT,99995)
      TOTAL1 = 0.0
      TOTAL2 = 0.0
      DO 30  I=1, NY
C                                 COMPUTE THE OFFSET FOR THE Z-VECTOR
         IF (I .GE. NY-1) THEN
```

```
            K = I - NY + 2
         ELSE
            K = I + 2
         END IF
C
         X      = TWOPI*FLOAT(I-1)/FLOAT(NY-1)
         ORIGER = ABS(Y(I)-F1(X))
         FLTRER = ABS(Z(K)-F1(X))
         IF (MOD(I,11) .EQ. 1) WRITE (NOUT,99997) X, F1(X), ORIGER,
     &       FLTRER
         TOTAL1 = TOTAL1 + ORIGER
         TOTAL2 = TOTAL2 + FLTRER
   30 CONTINUE
      WRITE (NOUT,99998) TOTAL1/FLOAT(NY)
      WRITE (NOUT,99999) TOTAL2/FLOAT(NY)
C                                 SET UP Y-VECTOR FOR THE NONPERIODIC
C                                 CASE.
      DO 40  I=1, NY
         A    = FLOAT(I-1)/FLOAT(NY-1)
         Y(I) = F2(A) + 0.5*RNUNF() - 0.25
   40 CONTINUE
C                                 CALL THE CONVOLUTION ROUTINE FOR THE
C                                 NONPERIODIC CASE.
      NZ = 2*(NFLTR+NY-1)
      CALL RCONV (0, NFLTR, FLTR, NY, Y, 1, NZ, Z, ZHAT)
C                                 PRINT RESULTS
      WRITE (NOUT,99994)
      WRITE (NOUT,99996)
      TOTAL1 = 0.0
      TOTAL2 = 0.0
      DO 50  I=1, NY
         X      = FLOAT(I-1)/FLOAT(NY-1)
         ORIGER = ABS(Y(I)-F2(X))
         FLTRER = ABS(Z(I+2)-F2(X))
         IF (MOD(I,11) .EQ. 1) WRITE (NOUT,99997) X, F2(X), ORIGER,
     &       FLTRER
         TOTAL1 = TOTAL1 + ORIGER
         TOTAL2 = TOTAL2 + FLTRER
   50 CONTINUE
      WRITE (NOUT,99998) TOTAL1/FLOAT(NY)
      WRITE (NOUT,99999) TOTAL2/FLOAT(NY)
99993 FORMAT (’ Periodic Case’)
99994 FORMAT (/,’ Nonperiodic Case’)
99995 FORMAT (8X, ’x’, 9X, ’sin(x)’, 6X, ’Original Error’, 5X,
     &       ’Filtered Error’)
99996 FORMAT (8X, ’x’, 9X, ’exp(x)’, 6X, ’Original Error’, 5X,
     &       ’Filtered Error’)
99997 FORMAT (1X, F10.4, F13.4, 2F18.4)
99998 FORMAT (’ Average absolute error before filter:’, F10.5)
99999 FORMAT (’ Average absolute error after filter:’, F11.5)
      END
```

Output

```
Periodic Case
    x         sin(x)      Original Error     Filtered Error
 0.0000       0.0000            0.0811             0.0587
 0.6981       0.6428            0.0226             0.0781
 1.3963       0.9848            0.1526             0.0529
 2.0944       0.8660           0.0959           0.0125
 2.7925       0.3420           0.1747           0.0292
 3.4907      -0.3420           0.1035           0.0238
 4.1888      -0.8660           0.0402           0.0595
 4.8869      -0.9848           0.0673           0.0798
 5.5851      -0.6428           0.1044           0.0074
 6.2832       0.0000           0.0154           0.0018
 Average absolute error before filter:   0.12481
 Average absolute error after filter:    0.04778

Nonperiodic Case
    x         exp(x)      Original Error     Filtered Error
 0.0000       1.0000           0.1476           0.3915
 0.1111       1.1175           0.0537           0.0326
 0.2222       1.2488           0.1278           0.0932
 0.3333       1.3956           0.1136           0.0987
 0.4444       1.5596           0.1617           0.0964
 0.5556       1.7429           0.0071           0.0662
 0.6667       1.9477           0.1248           0.0713
 0.7778       2.1766           0.1556           0.0158
 0.8889       2.4324           0.1529           0.0696
 1.0000       2.7183           0.2124           1.0562
 Average absolute error before filter:   0.12538
 Average absolute error after filter:    0.07764
```

CCONV/DCCONV (Single/Double precision)

Compute the convolution of two complex vectors.

Usage

```
CALL CCONV (IDO, NX, X, NY, Y, IPAD, NZ, Z, ZHAT)
```

Arguments

IDO — Flag indicating the usage of CCONV. (Input)

IDO	**Usage**
0	If this is the only call to CCONV.

If CCONV is called multiple times in sequence with the same NX, NY, and IPAD, IDO should be set to

1	on the first call
2	on the intermediate calls
3	on the final call.

NX — Length of the vector X. (Input)

X — Complex vector of length NX. (Input)

NY — Length of the vector Y. (Input)

Y — Complex vector of length NY. (Input)

IPAD — IPAD should be set to zero for periodic data or to one for nonperiodic data. (Input)

NZ — Length of the vector `Z`. (Input/Output)
Upon input: When `IPAD` is zero, `NZ` must be at least `MAX(NX, NY)`. When `IPAD` is one, `NZ` must be greater than or equal to the smallest integer greater than or equal to (`NX` + `NY` – 1) of the form (2^{α}) `*` (3^{β}) `*` (5^{γ}) where alpha, beta, and gamma are nonnegative integers. Upon output, the value for `NZ` that was used by `CCONV`.

Z — Complex vector of length `NZ` containing the convolution of `X` and `Y`. (Output)

ZHAT — Complex vector of length `NZ` containing the discrete complex Fourier transform of `Z`. (Output)

Comments

1. Automatic workspace usage is

   `CCONV` 10 `*` `NZ` + 15 units, or
   `DCCONV` 20 `*` `NZ` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `C2ONV`/`DC2ONV`. The reference is

   ```
   CALL C2ONV (IDO, NX, X, NY, Y, IPAD, NZ, Z, ZHAT,
               XWK, YWK, WK)
   ```

   The additional arguments are as follows:

   XWK — Complex work array of length `NZ`.

   YWK — Complex work array of length `NZ`.

   WK — Real work array of length 6 `*` `NZ` + 15.

2. Informational error

Type	Code	
4	1	The length of the vector `Z` must be large enough to hold the results. An acceptable length is returned in `NZ`.

Algorithm

The subroutine `CCONV` computes the discrete convolution of two complex sequences *x* and *y*. More precisely, let n_x be the length of *x* and n_y denote the length of *y*. If a circular convolution is desired, then `IPAD` must be set to zero. We set

$$n_z := \max\{n_x, n_y\}$$

and we pad out the shorter vector with zeroes. Then, we compute

$$z_i = \sum_{j=1}^{n_z} x_{i-j+1}\, y_j$$

where the index on x is interpreted as a positive number between 1 and n_z, modulo n_z.

The technique used to compute the z_i's is based on the fact that the (complex discrete) Fourier transform maps convolution into multiplication. Thus, the Fourier transform of z is given by

$$\hat{z}(n) = \hat{x}(n)\hat{y}(n)$$

where

$$\hat{z}(n) = \sum_{m=1}^{n_z} z_m e^{-2\pi i(m-1)(n-1)/n_z}$$

The technique used here to compute the convolution is to take the discrete Fourier transform of x and y, multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is a product of small primes if IPAD is set to zero. If n_z is a product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If IPAD is one, then a a good value is chosen for n_z so that the Fourier transforms are efficient and $n_z \ge n_x + n_y - 1$. This will mean that both vectors will be padded with zeroes.

Example

In this example, we compute both a periodic and a non-periodic convolution. The idea here is that one can compute a moving average of the type found in digital filtering using this routine. The averaging operator in this case is especially simple and is given by averaging five consecutive points in the sequence. The periodic case tries to recover a noisy function $f_1(x) = \cos(x) + i \sin(x)$ by averaging five nearby values. The nonperiodic case tries to recover the values of the function $f_2(x) = e^x f_1(x)$ contaminated by noise. The large error for the first and last value printed has to do with the fact that the convolution is averaging the zeroes in the “pad” rather than function values. Notice that the signal size is 100, but we only report the errors at ten points.

```
      INTEGER    NFLTR, NY
      PARAMETER  (NFLTR=5, NY=100)
C
      INTEGER    I, K, MOD, NOUT, NZ
      REAL       CABS, CONST, COS, EXP, FLOAT, FLTRER, ORIGER, RNUNF,
     &           SIN, TOTAL1, TOTAL2, TWOPI, X
      COMPLEX    CMPLX, F1, F2, FLTR(NFLTR), Y(NY), Z(2*(NFLTR+NY-1)),
     &           ZHAT(2*(NFLTR+NY-1))
      INTRINSIC  CABS, CMPLX, COS, EXP, FLOAT, MOD, SIN
      EXTERNAL   CCONV, CONST, RNSET, RNUNF, UMACH
C                                DEFINE FUNCTIONS
      F1(X) = CMPLX(COS(X),SIN(X))
      F2(X) = EXP(X)*CMPLX(COS(X),SIN(X))
C
      CALL RNSET (1234579)
      CALL UMACH (2, NOUT)
      TWOPI = 2.0*CONST('PI')
C                                 SET UP THE FILTER
      CALL CSET(NFLTR,(0.2,0.0),FLTR,1)
C                                 SET UP Y-VECTOR FOR THE PERIODIC
C                                 CASE.
      DO 20  I=1, NY
         X    = TWOPI*FLOAT(I-1)/FLOAT(NY-1)
         Y(I) = F1(X) + CMPLX(0.5*RNUNF()-0.25,0.5*RNUNF()-0.25)
   20 CONTINUE
C                                 CALL THE CONVOLUTION ROUTINE FOR THE
C                                 PERIODIC CASE.
      NZ = 2*(NFLTR+NY-1)
      CALL CCONV (0, NFLTR, FLTR, NY, Y, 0, NZ, Z, ZHAT)
C                                 PRINT RESULTS
      WRITE (NOUT,99993)
      WRITE (NOUT,99995)
      TOTAL1 = 0.0
      TOTAL2 = 0.0
      DO 30  I=1, NY
C                                 COMPUTE THE OFFSET FOR THE Z-VECTOR
         IF (I .GE. NY-1) THEN
            K = I - NY + 2
         ELSE
            K = I + 2
         END IF
C
         X      = TWOPI*FLOAT(I-1)/FLOAT(NY-1)
         ORIGER = CABS(Y(I)-F1(X))
         FLTRER = CABS(Z(K)-F1(X))
         IF (MOD(I,11) .EQ. 1) WRITE (NOUT,99997) X, F1(X), ORIGER,
     &       FLTRER
         TOTAL1 = TOTAL1 + ORIGER
         TOTAL2 = TOTAL2 + FLTRER
   30 CONTINUE
      WRITE (NOUT,99998) TOTAL1/FLOAT(NY)
      WRITE (NOUT,99999) TOTAL2/FLOAT(NY)
C                                 SET UP Y-VECTOR FOR THE NONPERIODIC
C                                 CASE.
      DO 40  I=1, NY
         X    = FLOAT(I-1)/FLOAT(NY-1)
         Y(I) = F2(X) + CMPLX(0.5*RNUNF()-0.25,0.5*RNUNF()-0.25)
   40 CONTINUE
C                                 CALL THE CONVOLUTION ROUTINE FOR THE
C                                 NONPERIODIC CASE.
      NZ = 2*(NFLTR+NY-1)
      CALL CCONV (0, NFLTR, FLTR, NY, Y, 1, NZ, Z, ZHAT)
C                                 PRINT RESULTS
      WRITE (NOUT,99994)
      WRITE (NOUT,99996)
      TOTAL1 = 0.0
      TOTAL2 = 0.0
      DO 50  I=1, NY
         X      = FLOAT(I-1)/FLOAT(NY-1)
         ORIGER = CABS(Y(I)-F2(X))
         FLTRER = CABS(Z(I+2)-F2(X))
         IF (MOD(I,11) .EQ. 1) WRITE (NOUT,99997) X, F2(X), ORIGER,
     &       FLTRER
         TOTAL1 = TOTAL1 + ORIGER
         TOTAL2 = TOTAL2 + FLTRER
   50 CONTINUE
      WRITE (NOUT,99998) TOTAL1/FLOAT(NY)
      WRITE (NOUT,99999) TOTAL2/FLOAT(NY)
99993 FORMAT (' Periodic Case')
99994 FORMAT (/, ' Nonperiodic Case')
99995 FORMAT (8X, 'x', 15X, 'f1(x)', 8X, 'Original Error', 5X,
     &       'Filtered Error')
99996 FORMAT (8X, 'x', 15X, 'f2(x)', 8X, 'Original Error', 5X,
     &       'Filtered Error')
99997 FORMAT (1X, F10.4, 5X, '(', F7.4, ',', F8.4, ' )', 5X, F8.4,
     &       10X, F8.4)
99998 FORMAT (' Average absolute error before filter:', F11.5)
99999 FORMAT (' Average absolute error after filter:', F12.5)
      END
```

Output

```
Periodic Case
 x               f1(x)        Original Error     Filtered Error
 0.0000     ( 1.0000,  0.0000 )       0.1666            0.0773
 0.6981     ( 0.7660,  0.6428 )       0.1685            0.1399
 1.3963     ( 0.1736,  0.9848 )       0.1756            0.0368
 2.0944     (-0.5000,  0.8660 )       0.2171            0.0142
 2.7925     (-0.9397,  0.3420 )       0.1147            0.0200
 3.4907     (-0.9397, -0.3420 )       0.0998            0.0331
 4.1888     (-0.5000, -0.8660 )       0.1137            0.0586
 4.8869     ( 0.1736, -0.9848 )       0.2217            0.0843
 5.5851     ( 0.7660, -0.6428 )       0.1831            0.0744
 6.2832     ( 1.0000,  0.0000 )       0.3234            0.0893
 Average absolute error before filter:    0.19315
 Average absolute error after filter:     0.08296

Nonperiodic Case
 x               f2(x)        Original Error     Filtered Error
 0.0000     ( 1.0000,  0.0000 )       0.0783            0.4336
 0.1111     ( 1.1106,  0.1239 )       0.2434            0.0477
 0.2222     ( 1.2181,  0.2752 )       0.1819            0.0584
 0.3333     ( 1.3188,  0.4566 )       0.0703            0.1267
 0.4444     ( 1.4081,  0.6706 )       0.1458            0.0868
 0.5556     ( 1.4808,  0.9192 )       0.1946            0.0930
 0.6667     ( 1.5307,  1.2044 )       0.1458            0.0734
 0.7778     ( 1.5508,  1.5273 )       0.1815            0.0690
 0.8889     ( 1.5331,  1.8885 )       0.0805            0.0193
 1.0000     ( 1.4687,  2.2874 )       0.2396            1.1708
 Average absolute error before filter:    0.18549
 Average absolute error after filter:     0.09636
```

RCORL/DRCORL (Single/Double precision)

Compute the correlation of two real vectors.

Usage

```
CALL RCORL (IDO, N, X, Y, IPAD, NZ, Z, ZHAT)
```

Arguments

IDO — Flag indicating the usage of RCORL. (Input)

IDO	Usage
0	If this is the only call to RCORL.

If RCORL is called multiple times in sequence with the same NX, NY, and IPAD, IDO should be set to

1	on the first call
2	on the intermediate calls
3	on the final call.

N — Length of the X and Y vectors. (Input)

X — Real vector of length N. (Input)

Y — Real vector of length N. (Input)

IPAD — IPAD should be set as follows. (Input)

IPAD	Value
IPAD	0 for periodic data with X and Y different.
IPAD	1 for nonperiodic data with X and Y different.
IPAD	2 for periodic data with X and Y identical.
IPAD	3 for nonperiodic data with X and Y identical.

NZ — Length of the vector Z. (Input/Output)
Upon input: When IPAD is zero or two, NZ must be at least (2 * N – 1). When IPAD is one or three, NZ must be greater than or equal to the smallest integer greater than or equal to (2 * N – 1) of the form $(2^\alpha) * (3^\beta) * (5^\gamma)$ where alpha, beta, and gamma are nonnegative integers. Upon output, the value for NZ that was used by RCORL.

Z — Real vector of length NZ containing the correlation of X and Y. (Output)

ZHAT — Real vector of length NZ containing the discrete Fourier transform of Z. (Output)

Comments

1. Automatic workspace usage is

   RCORL 4 * NZ + 15 units, or
   DRCORL 8 * NZ + 30 units.

   Workspace may be explicitly provided, if desired, by use of R2ORL/DR2ORL. The reference is

   ```
   CALL R2ORL (IDO, N, X, Y, IPAD, NZ, Z, ZHAT, XWK,
               YWK, WK)
   ```

   The additional arguments are as follows:

   XWK — Real work array of length NZ.

YWK — Real work array of length `NZ`.

WK — Real work array of length 2 * `NZ` + 15.

2. Informational error

Type	Code	
4	1	The length of the vector `Z` must be large enough to hold the results. An acceptable length is returned in `NZ`.

Algorithm

The subroutine `RCORL` computes the discrete correlation of two sequences *x* and *y*. More precisely, let *n* be the length of *x* and *y*. If a circular correlation is desired, then `IPAD` must be set to zero (for *x* and *y* distinct) and two (for *x* = *y*). We set (on output)

$$n_z = n \qquad \text{if IPAD } = \ 0, \ 2$$

$$n_z = 2^\alpha 3^\beta 5^\gamma \geq 2n - 1 \qquad \text{if IPAD } = \ 1, \ 3$$

where α, β, γ are nonnegative integers yielding the smallest number of the type $2^\alpha 3^\beta 5^\gamma$ satisfying the inequality. Once n_z is determined, we pad out the vectors with zeroes. Then, we compute

$$z_i = \sum_{j=1}^{n_z} x_{i+j-1} y_j$$

where the index on *x* is interpreted as a positive number between one and n_z, modulo n_z. Note that this means that

$$z_{n_z - k}$$

contains the correlation of $x(\cdot - \mathrm{k} - 1)$ with *y* as *k* = 0, 1, …, n_z/2. Thus, if $x(k - 1) = y(k)$ for all *k*, then we would expect

$$z_{n_z}$$

to be the largest component of *z*.

The technique used to compute the z_i's is based on the fact that the (complex discrete) Fourier transform maps correlation into multiplication. Thus, the Fourier transform of *z* is given by

$$\hat{z}_j = \hat{x}_j \overline{\hat{y}}_j$$

where

$$\hat{z}_j = \sum_{m=1}^{n_z} z_m e^{-2\pi i(m-1)(j-1)/n_z}$$

Thus, the technique used here to compute the correlation is to take the discrete Fourier transform of *x* and the conjugate of the discrete Fourier transform of *y*, multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is a product of small primes if `IPAD` is set to zero or two. If n_z is a product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If `IPAD` is one or three, then a good value is chosen for n_z so that the Fourier transforms are efficient and $n_z \geq 2n - 1$. This will mean that both vectors will be padded with zeroes.

We point out that no complex transforms of *x* or *y* are taken since both sequences are real, and we can take real transforms and simulate the complex transform above. This can produce a savings of a factor of six in time as well as save space over using the complex transform.

Example

In this example, we compute both a periodic and a non-periodic correlation between two distinct signals *x* and *y*. In the first case we have 100 equally spaced points on the interval [0, 2π] and $f_1(x) = \sin(x)$. We define *x* and *y* as follows

$$\begin{aligned} x_i &= f_1(2\pi \frac{i-1}{n-1}) \quad i = 1, \ldots, n \\ y_i &= f_1(2\pi \frac{i-1}{n-1} + \frac{\pi}{2}) \qquad i = 1, \ldots, n \end{aligned}$$

Note that the maximum value of *z* (the correlation of *x* with *y*) occurs at $i = 26$, which corresponds to the offset.

The nonperiodic case uses the function $f_2(x) = \sin(x^2)$. The two input signals are on the interval [0, 4π].

$$\begin{aligned} x_i &= f_2(4\pi \frac{i-1}{n-1}) \qquad i = 1, \ldots, n \\ y_i &= f_2(4\pi \frac{i-1}{n-1} + \pi) \qquad i = 1, \ldots, n \end{aligned}$$

The offset of *x* to *y* is again (roughly) 26 and this is where *z* has its maximum value.

```
      INTEGER    N
      PARAMETER  (N=100)
C
      INTEGER    I, ISMAX, K, NOUT, NZ
      REAL       A, CONST, F1, F2, FLOAT, PI, SIN, SNRM2, X(N), XNORM,
     &           Y(N), YNORM, Z(4*N), ZHAT(4*N)
      INTRINSIC  FLOAT, SIN
      EXTERNAL   CONST, ISMAX, RCORL, SNRM2, UMACH
C                                 Define functions
```

```
      F1(A) = SIN(A)
      F2(A) = SIN(A*A)
C
      CALL UMACH (2, NOUT)
      PI = CONST('pi')
C                                 Set up the vectors for the
C                                 periodic case.
      DO 10  I=1, N
         X(I) = F1(2.0*PI*FLOAT(I-1)/FLOAT(N-1))
         Y(I) = F1(2.0*PI*FLOAT(I-1)/FLOAT(N-1)+PI/2.0)
   10 CONTINUE
C                                 Call the correlation routine for the
C                                 periodic case.
      NZ = 2*N
      CALL RCORL (0, N, X, Y, 0, NZ, Z, ZHAT)
C                                 Find the element of Z with the
C                                 largest normalized value.
      XNORM = SNRM2(N,X,1)
      YNORM = SNRM2(N,Y,1)
      DO 20  I=1, N
         Z(I) = Z(I)/(XNORM*YNORM)
   20 CONTINUE
      K = ISMAX(N,Z,1)
C                                 Print results for the periodic
C                                 case.
      WRITE (NOUT,99995)
      WRITE (NOUT,99994)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) K
      WRITE (NOUT,99999) K, Z(K)
C                                 Set up the vectors for the
C                                 nonperiodic case.
      DO 30  I=1, N
         X(I) = F2(4.0*PI*FLOAT(I-1)/FLOAT(N-1))
         Y(I) = F2(4.0*PI*FLOAT(I-1)/FLOAT(N-1)+PI)
   30 CONTINUE
C                                 Call the correlation routine for the
C                                 nonperiodic case.
      NZ = 4*N
      CALL RCORL (0, N, X, Y, 1, NZ, Z, ZHAT)
C                                 Find the element of Z with the
C                                 largest normalized value.
      XNORM = SNRM2(N,X,1)
      YNORM = SNRM2(N,Y,1)
      DO 40  I=1, N
         Z(I) = Z(I)/(XNORM*YNORM)
   40 CONTINUE
      K = ISMAX(N,Z,1)
C                                 Print results for the nonperiodic
C                                 case.
      WRITE (NOUT,99996)
      WRITE (NOUT,99994)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) K
      WRITE (NOUT,99999) K, Z(K)
99994 FORMAT (1X, 28('-'))
99995 FORMAT (' Case #1: Periodic data')
99996 FORMAT (/, ' Case #2: Nonperiodic data')
99997 FORMAT (' The element of Z with the largest normalized ')
```

```
99998 FORMAT (’ value is Z(’, I2, ’).’)
99999 FORMAT (’ The normalized value of Z(’, I2, ’) is’, F6.3)
      END
```

Output

```
Example #1: Periodic case
---------------------------
The element of Z with the largest normalized value is Z(26).
The normalized value of Z(26) is 1.000

Example #2: Nonperiodic case
---------------------------
The element of Z with the largest normalized value is Z(26).
The normalized value of Z(26) is 0.661
```

CCORL/DCCORL (Single/Double precision)

Compute the correlation of two complex vectors.

Usage

`CALL CCORL (IDO, N, X, Y, IPAD, NZ, Z, ZHAT)`

Arguments

IDO — Flag indicating the usage of `CCORL`. (Input)

IDO	**Usage**
0	If this is the only call to `CCORL`.

If `CCORL` is called multiple times in sequence with the same `NX`, `NY`, and `IPAD`, `IDO` should be set to

1	on the first call
2	on the intermediate calls
3	on the final call.

N — Length of the `X` and `Y` vectors. (Input)

X — Complex vector of length `N`. (Input)

Y — Complex vector of length `N`. (Input)

IPAD — `IPAD` should be set as follows. (Input)
`IPAD` = 0 for periodic data with `X` and `Y` different. `IPAD` = 1 for nonperiodic data with `X` and `Y` different. `IPAD` = 2 for periodic data with `X` and `Y` identical. `IPAD` = 3 for nonperiodic data with `X` and `Y` identical.

NZ — Length of the vector `Z`. (Input/Output)
Upon input: When `IPAD` is zero or two, `NZ` must be at least (2 * `N` – 1). When `IPAD` is one or three, `NZ` must be greater than or equal to the smallest integer greater than or equal to (2 * `N` – 1) of the form $(2^\alpha) * (3^\beta) * (5^\gamma)$ where alpha, beta, and gamma are nonnegative integers. Upon output, the value for `NZ` that was used by `CCORL`.

Z — Complex vector of length `NZ` containing the correlation of `X` and `Y`. (Output)

ZHAT — Complex vector of length `NZ` containing the inverse discrete complex Fourier transform of `Z`. (Output)

Comments

1. Automatic workspace usage is

   `CCORL` 10 `*` `NZ` + 15 units, or
   `DCCORL` 20 `*` `NZ` + 30 units.

   Workspace may be explicitly provided, if desired, by use of `C2ORL`/`DC2ORL`. The reference is

   ```
   CALL C2ORL (IDO, N, X, Y, IPAD, NZ, Z, ZHAT, XWK,
               YWK, WK)
   ```

   The additional arguments are as follows:

   XWK — Complex work array of length `NZ`.

   YWK — Complex work array of length `NZ`.

   WK — Real work arrary of length 6 `*` `NZ` + 15.

2. Informational error

Type	Code	
4	1	The length of the vector `Z` must be large enough to hold the results. An acceptable length is returned in `NZ`.

Algorithm

The subroutine `CCORL` computes the discrete correlation of two complex sequences *x* and *y*. More precisely, let *n* be the length of *x* and *y*. If a circular correlation is desired, then `IPAD` must be set to zero (for *x* and *y* distinct) and two (for *x* = *y*). We set (on output)

$$n_z = n \qquad \text{if IPAD } = \ 0,\ 2$$

$$n_z = 2^{\alpha} 3^{\beta} 5^{\gamma} \geq 2n - 1 \qquad \text{if IPAD } = \ 1,\ 3$$

where α, β, γ are nonnegative integers yielding the smallest number of the type $2^{\alpha}3^{\beta}5^{\gamma}$ satisfying the inequality. Once n_z is determined, we pad out the vectors with zeroes. Then, we compute

$$z_i = \sum_{j=1}^{n_z} x_{i+j-1} \bar{y}_j$$

where the index on *x* is interpreted as a positive number between one and n_z, modulo n_z. Note that this means that

$$z_{n_z - k}$$

contains the correlation of $x(\cdot - k - 1)$ with y as $k = 0, 1, \ldots, n_z/2$. Thus, if $x(k - 1) = y(k)$ for all k, then we would expect

$$\Re z_{n_z}$$

to be the largest component of $\Re z$.

The technique used to compute the z_i's is based on the fact that the (complex discrete) Fourier transform maps correlation into multiplication. Thus, the Fourier transform of z is given by

$$\hat{z}_j = \hat{x}_j \bar{\hat{y}}_j$$

where

$$\hat{z}_j = \sum_{m=1}^{n_z} z_m e^{-2\pi i(m-1)(j-1)/n_z}$$

Thus, the technique used here to compute the correlation is to take the discrete Fourier transform of x and the conjugate of the discrete Fourier transform of y, multiply the results together component-wise, and then take the inverse transform of this product. It is very important to make sure that n_z is a product of small primes if `IPAD` is set to zero or two. If n_z is a product of small primes, then the computational effort will be proportional to $n_z \log(n_z)$. If `IPAD` is one or three, then a good value is chosen for n_z so that the Fourier transforms are efficient and $n_z \geq 2n - 1$. This will mean that both vectors will be padded with zeroes.

Example

In this example, we compute both a periodic and a non-periodic correlation between two distinct signals x and y. In the first case, we have 100 equally spaced points on the interval $[0, 2\pi]$ and $f_1(x) = \cos(x) + i \sin(x)$. We define x and y as follows

$$\begin{aligned} x_i &= f_1\left(2\pi \frac{i-1}{n-1}\right) \quad i = 1, \ldots, n \\ y_i &= f_1\left(2\pi \frac{i-1}{n-1} + \frac{\pi}{2}\right) \qquad i = 1, \ldots, n \end{aligned}$$

Note that the maximum value of z (the correlation of x with y) occurs at $i = 26$, which corresponds to the offset.

The nonperiodic case uses the function $f_2(x) = \cos(x^2) + i \sin(x^2)$. The two input signals are on the interval $[0, 4\pi]$.

$$x_i = f_2(4\pi\frac{i-1}{n-1}) \qquad i = 1, \ldots, n$$

$$y_i = f_2(4\pi\frac{i-1}{n-1} + \pi) \qquad i = 1, \ldots, n$$

The offset of *x* to *y* is again (roughly) 26 and this is where *z* has its maximum value.

```
      INTEGER    N
      PARAMETER  (N=100)
C
      INTEGER    I, ISMAX, K, NOUT, NZ
      REAL       A, CONST, COS, F1, F2, FLOAT, PI, SCNRM2, SIN,
     &           XNORM, YNORM, ZREAL(4*N)
      COMPLEX    CMPLX, X(N), Y(N), Z(4*N), ZHAT(4*N)
      INTRINSIC  CMPLX, COS, FLOAT, SIN
      EXTERNAL   CCORL, CONST, ISMAX, RNSET, SCNRM2, UMACH
C                                Define functions
      F1(A) = CMPLX(COS(A),SIN(A))
      F2(A) = CMPLX(COS(A*A),SIN(A*A))
C
      CALL RNSET (1234579)
      CALL UMACH (2, NOUT)
      PI = CONST('pi')
C                                 Set up the vectors for the
C                                 periodic case.
      DO 10  I=1, N
         X(I) = F1(2.0*PI*FLOAT(I-1)/FLOAT(N-1))
         Y(I) = F1(2.0*PI*FLOAT(I-1)/FLOAT(N-1)+PI/2.0)
   10 CONTINUE
C                                 Call the correlation routine for the
C                                 periodic case.
      NZ = 2*N
      CALL CCORL (0, N, X, Y, 0, NZ, Z, ZHAT)
C                                 Find the element of Z with the
C                                 largest normalized real part.
      XNORM = SCNRM2(N,X,1)
      YNORM = SCNRM2(N,Y,1)
      DO 20  I=1, N
         ZREAL(I) = REAL(Z(I))/(XNORM*YNORM)
   20 CONTINUE
      K = ISMAX(N,ZREAL,1)
C                                 Print results for the periodic
C                                 case.
      WRITE (NOUT,99995)
      WRITE (NOUT,99994)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) K
      WRITE (NOUT,99999) K, ZREAL(K)
C                                 Set up the vectors for the
C                                 nonperioddic case.
      DO 30  I=1, N
         X(I) = F2(4.0*PI*FLOAT(I-1)/FLOAT(N-1))
         Y(I) = F2(4.0*PI*FLOAT(I-1)/FLOAT(N-1)+PI)
   30 CONTINUE
C                                 Call the correlation routine for the
C                                 nonperiodic case.
```

```
      NZ = 4*N
      CALL CCORL (0, N, X, Y, 1, NZ, Z, ZHAT)
C                                  Find the element of z with the
C                                  largest normalized real part.
      XNORM = SCNRM2(N,X,1)
      YNORM = SCNRM2(N,Y,1)
      DO 40  I=1, N
         ZREAL(I) = REAL(Z(I))/(XNORM*YNORM)
   40 CONTINUE
      K = ISMAX(N,ZREAL,1)
C                                  Print results for the nonperiodic
C                                  case.
      WRITE (NOUT,99996)
      WRITE (NOUT,99994)
      WRITE (NOUT,99997)
      WRITE (NOUT,99998) K
      WRITE (NOUT,99999) K, ZREAL(K)
99994 FORMAT (1X, 28('-'))
99995 FORMAT (' Case #1: periodic data')
99996 FORMAT (/, ' Case #2: nonperiodic data')
99997 FORMAT (' The element of Z with the largest normalized ')
99998 FORMAT (' real part is Z(', I2, ').')
99999 FORMAT (' The normalized value of real(Z(', I2, ')) is', F6.3)
      END
```

Output

```
Example #1: periodic case
----------------------------
The element of Z with the largest normalized real part is Z(26).
The normalized value of real(Z(26)) is 1.000

Example #2: nonperiodic case
----------------------------
The element of Z with the largest normalized real part is Z(26).
The normalized value of real(Z(26)) is 0.638
```

INLAP/DINLAP (Single/Double precision)

Compute the inverse Laplace transform of a complex function.

Usage

`CALL INLAP (F, N, T, ALPHA, RELERR, KMAX, FINV)`

Arguments

F — User-supplied `FUNCTION` to which the inverse Laplace transform will be computed. The form is `F(Z)`, where

`Z` – Complex argument. (Input)
`F` – The complex function value. (Output)

`F` must be declared `EXTERNAL` in the calling program. `F` should also be declared `COMPLEX`.

N — Number of points at which the inverse Laplace transform is desired. (Input)

T — Array of length `N` containing the points at which the inverse Laplace transform is desired. (Input)
`T(I)` must be greater than zero for all `I`.

ALPHA — An estimate for the maximum of the real parts of the singularities of `F`. If unknown, set `ALPHA` = 0. (Input)

RELERR — The relative accuracy desired. (Input)

KMAX — The number of function evaluations allowed for each `T(I)`. (Input)

FINV — Array of length `N` whose `I`-th component contains the approximate value of the Laplace transform at the point `T(I)`. (Output)

Comments

Informational errors

Type	Code	
4	1	The algorithm was not able to achieve the accuracy requested within `KMAX` function evaluations for some `T(I)`.
4	2	Overflow is occurring for a particular value of `T`.

Algorithm

The routine `INLAP` computes the inverse Laplace transform of a complex-valued function. Recall that if *f* is a function that vanishes on the negative real axis, then we can define the Laplace transform of *f* by

$$L[f](s) := \int_0^\infty e^{-sx} f(x)dx$$

It is assumed that for some value of *s* the integrand is absolutely integrable.

The computation of the inverse Laplace transform is based on applying the epsilon algorithm to the complex Fourier series obtained as a discrete approximation to the inversion integral. The initial algorithm was proposed by K.S. Crump (1976) but was significantly improved by de Hoog et al. (1982). Given a complex-valued transform $F(s) = L[f](s)$, the trapezoidal rule gives the approximation to the inverse transform

$$g(t) = \left(e^{\alpha t} / T\right)\Re\left\{\frac{1}{2}F(\alpha) + \sum_{k=1}^{\infty} F(\alpha + \frac{ik\pi}{T})\exp(\frac{ik\pi t}{T})\right\}$$

This is the real part of the sum of a complex power series in $z = \exp(i\pi t/T)$, and the algorithm accelerates the convergence of the partial sums of this power series by using the epsilon algorithm to compute the corresponding diagonal Pade approximants. The algorithm attempts to choose the order of the Pade approximant to obtain the specified relative accuracy while not exceeding the maximum number of function evaluations allowed. The parameter α is an estimate for the maximum of the real parts of the singularities of F, and an incorrect choice of α may give false convergence. Even in cases where the correct value of α is unknown, the algorithm will attempt to estimate an acceptable value. Assuming satisfactory convergence, the discretization error $E := g - f$ satisfies

$$E = \sum_{n=1}^{\infty} e^{-2n\alpha T} f(2nT + t)$$

It follows that if $|f(t)| \le Me^{\beta t}$, then we can estimate the expression above to obtain (for $0 \le t \le 2T$)

$$E \le Me^{\alpha t} \,/\, \left(e^{2T(\alpha-\beta)} - 1\right)$$

Example

We invert the Laplace transform of the simple function $(s - 1)^{-2}$ and print the computed answer, the true solution and the difference at five different points. The correct inverse transform is xe^{x}.

```
      INTEGER    I, KMAX, N, NOUT
      REAL       ALPHA, DIFF(5), EXP, FINV(5), FLOAT, RELERR, T(5),
     &           TRUE(5)
      COMPLEX    F
      INTRINSIC  EXP, FLOAT
      EXTERNAL   F, INLAP, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C
      DO 10  I=1, 5
         T(I) = FLOAT(I) - 0.5
   10 CONTINUE
      N      = 5
      ALPHA  = 1.0E0
      KMAX   = 500
      RELERR = 5.0E-4
      CALL INLAP (F, N, T, ALPHA, RELERR, KMAX, FINV)
C                                 Evaluate the true solution and the
C                                 difference
      DO 20  I=1, 5
         TRUE(I) = T(I)*EXP(T(I))
         DIFF(I) = TRUE(I) - FINV(I)
   20 CONTINUE
C
      WRITE (NOUT,99999) (T(I),FINV(I),TRUE(I),DIFF(I),I=1,5)
99999 FORMAT (7X, ’T’, 8X, ’FINV’, 9X, ’TRUE’, 9X, ’DIFF’, /,
     &       5(1X,E9.1,3X,1PE10.3,3X,1PE10.3,3X,1PE10.3,/))
      END
C
      COMPLEX FUNCTION F (S)
      COMPLEX    S
      F = 1./(S-1.)**2
      RETURN
      END
```

Output

```
     T         FINV         TRUE         DIFF
0.5E+00    8.244E-01    8.244E-01   -4.768E-06
1.5E+00    6.723E+00    6.723E+00   -3.481E-05
2.5E+00    3.046E+01    3.046E+01   -1.678E-04
3.5E+00    1.159E+02    1.159E+02   -6.027E-04
4.5E+00    4.051E+02    4.051E+02   -2.106E-03
```

SINLP/DSINLP (Single/Double precision)

Compute the inverse Laplace transform of a complex function.

Usage

```
CALL SINLP (F, N, T, SIGMA0, EPSTOL, ERRVEC, FINV)
```

Arguments

F — User-supplied `FUNCTION` to which the inverse Laplace transform will be computed. The form is `F(Z)`, where

> `Z` — Complex argument. (Input)
> `F` — The complex function value. (Output)

`F` must be declared `EXTERNAL` in the calling program. `F` must also be declared `COMPLEX`.

N — The number of points at which the inverse Laplace transform is desired. (Input)

T — Vector of length `N` containing points at which the inverse Laplace transform is desired. (Input)
`T(I)` must be greater than zero for all `I`.

SIGMA0 — An estimate for the maximum of the real parts of the singularities of `F`. (Input)
If unknown, set `SIGMA0` = 0.0.

EPSTOL — The required absolute uniform pseudo accuracy for the coefficients and inverse Laplace transform values. (Input)

ERRVEC — Vector of length eight containing diagnostic information. (Output)
All components depend on the intermediately generated Laguerre coefficients. See Comments.

FINV — Vector of length `N` whose `I`-th component contains the approximate value of the inverse Laplace transform at the point `T(I)`. (Output)

Comments

1. Automatic workspace usage is

   SINLP 9 * MTOP/4 + N units, or
   DSINLP 9 * MTOP/2 + N units.

   Workspace may be explicitly provided, if desired, by use of S2NLP/DS2NLP. The reference is

   ```
   CALL S2NLP (F, N, T, SIGMA0, EPSTOL, ERRVEC, FINV,
               SIGMA, BVALUE, MTOP, WK, IFLOVC)
   ```

   The additional arguments are as follows:

   SIGMA — The first parameter of the Laguerre expansion. If SIGMA is not greater than SIGMA0, it is reset to SIGMA0 + 0.7. (Input)

   BVALUE — The second parameter of the Laguerre expansion. If BVALUE is less than 2.0 * (SIGMA – SIGMA0), it is reset to 2.5 * (SIGMA – SIGMA0). (Input)

   MTOP — An upper limit on the number of coefficients to be computed in the Laguerre expansion. MTOP must be a multiple of four. Note that the maximum number of Laplace transform evaluations is MTOP/2 + 2. (Default: 1024.) (Input)

   WK — Real work vector of length 9 * MTOP/4.

   IFLOVC — Integer vector of length N, the I-th component of which contains the overflow/underflow indicator for the computed value of FINV(I). (Output)
   See Comment 3.

2. Informational errors

Type	Code	
1	1	Normal termination, but with estimated error bounds slightly larger than EPSTOL. Note, however, that the actual errors on the final results may be smaller than EPSTOL as bounds independent of T are pessimistic.
3	2	Normal calculation, terminated early at the roundoff error level estimate because this estimate exceeds the required accuracy (usually due to overly optimistic expectation by the user about attainable accuracy).
4	3	The decay rate of the coefficients is too small. It may improve results to use S2NLP and increase MTOP.
4	4	The decay rate of the coefficients is too small. In addition, the roundoff error level is such that required accuracy cannot be reached.
4	5	No error bounds are returned as the behavior of the coefficients does not enable reasonable prediction. Results are probably wrong. Check the value of

SIGMA0. In this case, each of ERRVEC(J), J = 1, …, 5, is set to – 1.0.

3. The following are descriptions of the vectors ERRVEC and IFLOVC.

ERRVEC — Real vector of length eight.

ERRVEC(1) = Overall estimate of the pseudo error, ERRVEC(2) + ERRVEC(3) + ERRVEC(4). Pseudo error = absolute error / exp(sigma * tvalue).

ERRVEC(2) = Estimate of the pseudo discretization error.

ERRVEC(3) = Estimate of the pseudo truncation error.

ERRVEC(4) = Estimate of the pseudo condition error on the basis of minimal noise levels in the function values.

ERRVEC(5) = K, the coefficient of the decay function for ACOEF, the coefficients of the Laguerre expansion.

ERRVEC(6) = R, the base of the decay function for ACOEF. Here abs(ACOEF (J + 1)).LE.K/R**J for J.GE.MACT/2, where MACT is the number of Laguerre coefficients actually computed.

ERRVEC(7) = ALPHA, the logarithm of the largest ACOEF.

ERRVEC(8) = BETA, the logarithm of the smallest nonzero ACOEF.

IFLOVC — Integer vector of length N containing the overflow/underflow indicators for FINV. For each I, the value of IFLOVC(I) signifies the following.

0 = Normal termination.

1 = The value of the inverse Laplace transform is found to be too large to be representable; FINV(I) is set to AMACH(6).

–1 = The value of the inverse Laplace transform is found to be too small to be representable; FINV(I) is set to 0.0.

2 = The value of the inverse Laplace transform is estimated to be too large, even before the series expansion, to be representable; FINV(I) is set to AMACH(6).

–2 = The value of the inverse Laplace transform is estimated to be too small, even before the series expansion, to be representable; FINV(I) is set to 0.0.

Algorithm

The routine SINLP computes the inverse Laplace transform of a complex-valued function. Recall that if *f* is a function that vanishes on the negative real axis, then we can define the Laplace transform of *f* by

$$L[f](s) := \int_0^\infty e^{-sx} f(x)dx$$

It is assumed that for some value of *s* the integrand is absolutely integrable.

The computation of the inverse Laplace transform is based on a modification of Weeks' method (see W.T. Weeks (1966)) due to B.S. Garbow et. al. (1988). This method is suitable when *f* has continuous derivatives of all orders on [0, ∞). In this situation, this routine should be used in place of the IMSL routine `INLAP` (page 827). It is especially efficient when multiple function values are desired. In particular, given a complex-valued function *F*(*s*) = *L*[*f*](*s*), we can expand *f* in a Laguerre series whose coefficients are determined by *F*. This is fully described in B.S. Garbow et. al. (1988) and Lyness and Giunta (1986).

The algorithm attempts to return approximations *g*(*t*) to *f*(*t*) satisfying

$$\left|\frac{g(t) - f(t)}{e^{\sigma t}}\right| < \varepsilon$$

where ε := `EPSTOL` and σ := `SIGMA` > `SIGMA0`. The expression on the left is called the pseudo error. An estimate of the pseudo error is available in `ERRVEC`(1).

The first step in the method is to transform *F* to ϕ where

$$\phi(z) = \frac{b}{1-z} F\left(\frac{b}{1-z} - \frac{b}{2} + \sigma\right)$$

Then, if *f* is smooth, it is known that ϕ is analytic in the unit disc of the complex plane and hence has a Taylor series expansion

$$\phi(z) = \sum_{s=0}^{\infty} a_s z^s$$

which converges for all *z* whose absolute value is less than the radius of convergence R_c. This number is estimated in `ERRVEC`(6). In `ERRVEC`(5), we estimate the smallest number *K* which satisfies

$$|a_s| < \frac{K}{R^s}$$

for all $R < R_c$.

The coefficients of the Taylor series for ϕ can be used to expand *f* in a Laguerre series

$$f(t) = e^{\sigma t} \sum_{s=0}^{\infty} a_s e^{-bt/2} L_s(bt)$$

Example

We invert the Laplace transform of the simple function $(s-1)^{-2}$ and print the computed answer, the true solution, and the difference at five different points.

The correct inverse transform is xe^{x}.

```
      INTEGER    I, N, NOUT
      REAL       DIFF(5), ERRVEC(8), EXP, FINV(5), FLOAT, RELERR,
     &           SIGMA0, T(5), TRUE(5)
      COMPLEX    F
      INTRINSIC  EXP, FLOAT
      EXTERNAL   F, SINLP, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C
      DO 10  I=1, 5
         T(I) = FLOAT(I) - 0.5
   10 CONTINUE
      N      = 5
      SIGMA0 = 1.0E0
      RELERR = 5.0E-4
      CALL SINLP (F, N, T, SIGMA0, RELERR, ERRVEC, FINV)
C                                  Evaluate the true solution and the
C                                  difference
      DO 20  I=1, 5
         TRUE(I) = T(I)*EXP(T(I))
         DIFF(I) = TRUE(I) - FINV(I)
   20 CONTINUE
C
      WRITE (NOUT,99999) (T(I),FINV(I),TRUE(I),DIFF(I),I=1,5)
99999 FORMAT (7X, ’T’, 8X, ’FINV’, 9X, ’TRUE’, 9X, ’DIFF’, /,
     &       5(1X,E9.1,3X,1PE10.3,3X,1PE10.3,3X,1PE10.3,/))
      END
C
      COMPLEX FUNCTION F (S)
      COMPLEX    S
C
      F = 1./(S-1.)**2
      RETURN
      END
```

Output

```
    T        FINV         TRUE         DIFF
0.5E+00    8.244E-01    8.244E-01   -2.086E-06
1.5E+00    6.723E+00    6.723E+00   -8.583E-06
2.5E+00    3.046E+01    3.046E+01    0.000E+00
3.5E+00    1.159E+02    1.159E+02    2.289E-05
4.5E+00    4.051E+02    4.051E+02   -2.136E-04
```

Chapter 7: Nonlinear Equations

Routines

Usage Notes

Zeros of a Polynomial

A polynomial function of degree *n* can be expressed as follows:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0$$

where $a_n \neq 0$.

There are three routines for zeros of a polynomial. The routines ZPLRC (page 836) and ZPORC (page 838) find zeros of the polynomial with real coefficients while the routine ZPOCC (page 839) finds zeros of the polynomial with complex coefficients.

The Jenkins-Traub method is used for the routines ZPORC and ZPOCC; whereas ZPLRC uses the Laguerre method. Both methods perform well in comparison with other methods. The Jenkins-Traub algorithm usually runs faster than the

Laguerre method. Furthermore, the routine `ZANLY` (page 841) in the next section can also be used for the complex polynomial.

Zero(s) of a Function

The routines `ZANLY` (page 841) and `ZREAL` (page 846) use Müller's method to find the zeros of a complex analytic function and real zeros of a real function, respectively. The routine `ZBREN` (page 843) finds a zero of a real function, using an algorithm that is a combination of interpolation and bisection. This algorithm requires the user to supply two points such that the function values at these two points have opposite sign. For functions where it is difficult to obtain two such points, `ZREAL` can be used.

Root of System of Equations

A system of equations can be stated as follows:

$$f_i(x) = 0, \text{ for } i = 1, 2, \ldots, n$$

where $x \, \varepsilon \, \mathbf{R}^n$.

The routines `NEQNF` (page 848) and `NEQNJ` (page 851) use a modified Powell hybrid method to find the zero of a system of nonlinear equations. The difference between these two routines is that the Jacobian is estimated by a finite-difference method in `NEQNF`, whereas the user has to provide the Jacobian for `NEQNJ`. It is advised that the Jacobian-checking routine, `CHJAC` (page 952), be used to ensure the accuracy of the user-supplied Jacobian.

The routines `NEQBF` (page 854) and `NEQBJ` (page 860) use a secant method with Broyden's update to find the zero of a system of nonlinear equations. The difference between these two routines is that the Jacobian is estimated by a finite-difference method in `NEQBF`; whereas the user has to provide the Jacobian for `NEQBJ`. For more details, see Dennis and Schnabel (1983, Chapter 8).

ZPLRC/DZPLRC (Single/Double precision)

Find the zeros of a polynomial with real coefficients using Laguerre's method.

Usage

```
CALL ZPLRC (NDEG, COEFF, ROOT)
```

Arguments

NDEG — Degree of the polynomial. $1 \leq$ `NDEG` ≤ 100 (Input)

COEFF — Vector of length `NDEG` + 1 containing the coefficients of the polynomial in increasing order by degree. (Input)

The polynomial is COEFF(NDEG + 1) * Z**NDEG + COEFF(NDEG) * Z**(NDEG – 1) + … + COEFF(1).

ROOT — Complex vector of length NDEG containing the zeros of the polynomial. (Output)

Comments

Informational errors

Type	Code	
3	1	The first several coefficients of the polynomial are equal to zero. Several of the last roots will be set to machine infinity to compensate for this problem.
3	2	Fewer than NDEG zeros were found. The ROOT vector will contain the value for machine infinity in the locations that do not contain zeros.

Algorithm

Routine ZPLRC computes the n zeros of the polynomial

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0$$

where the coefficients a_i for $i = 0, 1, \ldots, n$ are real and n is the degree of the polynomial.

The routine ZPLRC is a modification of B.T. Smith’s routine ZERPOL (Smith 1967) that uses Laguerre’s method. Laguerre’s method is cubically convergent for isolated zeros and linearly convergent for multiple zeros. The maximum length of the step between successive iterates is restricted so that each new iterate lies inside a region about the previous iterate known to contain a zero of the polynomial. An iterate is accepted as a zero when the polynomial value at that iterate is smaller than a computed bound for the rounding error in the polynomial value at that iterate. The original polynomial is deflated after each real zero or pair of complex zeros is found. Subsequent zeros are found using the deflated polynomial.

Example

This example finds the zeros of the third-degree polynomial

$$p(z) = z^3 - 3e^2 + 4z - 2$$

where z is a complex variable.

```
C                                 Declare variables
      INTEGER    NDEG
      PARAMETER  (NDEG=3)
C
      REAL       COEFF(NDEG+1)
      COMPLEX    ZERO(NDEG)
      EXTERNAL   WRCRN, ZPLRC
C                                 Set values of COEFF
```

```
C                                 COEFF = (-2.0  4.0 -3.0  1.0)
C
      DATA COEFF/-2.0, 4.0, -3.0, 1.0/
C
      CALL ZPLRC (NDEG, COEFF, ZERO)
C
      CALL WRCRN ('The zeros found are', 1, NDEG, ZERO, 1, 0)
C
      END
```

Output

```
              The zeros found are
             1                 2                 3
( 1.000, 1.000)  ( 1.000,-1.000)  ( 1.000, 0.000)
```

ZPORC/DZPORC (Single/Double precision)

Find the zeros of a polynomial with real coefficients using the Jenkins-Traub three-stage algorithm.

Usage

CALL ZPORC (NDEG, COEFF, ROOT)

Arguments

NDEG — Degree of the polynomial. $1 \le$ NDEG ≤ 100 (Input)

COEFF — Vector of length NDEG + 1 containing the coefficients of the polynomial in increasing order by degree.   (Input)
The polynomial is COEFF(NDEG + 1)*Z**NDEG + COEFF(NDEG) * Z**(NDEG −1) + … + COEFF(1).

ROOT — Complex vector of length NDEG containing the zeros of the polynomial. (Output)

Comments

Informational errors

Type	Code	
3	1	The first several coefficients of the polynomial are equal to zero. Several of the last roots will be set to machine infinity to compensate for this problem.
3	2	Fewer than NDEG zeros were found. The ROOT vector will contain the value for machine infinity in the locations that do not contain zeros.

Algorithm

Routine ZPORC computes the n zeros of the polynomial

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0$$

where the coefficients a_i for $i = 0, 1, \ldots, n$ are real and n is the degree of the polynomial.

The routine ZPORC uses the Jenkins-Traub three-stage algorithm (Jenkins and Traub 1970; Jenkins 1975). The zeros are computed one at a time for real zeros or two at a time for complex conjugate pairs. As the zeros are found, the real zero or quadratic factor is removed by polynomial deflation.

Example

This example finds the zeros of the third-degree polynomial

$$p(z) = z^3 - 3e^2 + 4z - 2$$

where z is a complex variable.

```
C                                 Declare variables
      INTEGER    NDEG
      PARAMETER  (NDEG=3)
C
      REAL       COEFF(NDEG+1)
      COMPLEX    ZERO(NDEG)
      EXTERNAL   WRCRN, ZPORC
C                                 Set values of COEFF
C                                 COEFF = (-2.0  4.0 -3.0  1.0)
C
      DATA COEFF/-2.0, 4.0, -3.0, 1.0/
C
      CALL ZPORC (NDEG, COEFF, ZERO)
C
      CALL WRCRN ('The zeros found are', 1, NDEG, ZERO, 1, 0)
C
      END
```

Output

```
              The zeros found are
             1                2                3
( 1.000, 0.000)  ( 1.000, 1.000)  ( 1.000,-1.000)
```

ZPOCC/DZPOCC (Single/Double precision)

Find the zeros of a polynomial with complex coefficients using the Jenkins-Traub three-stage algorithm.

Usage

CALL ZPOCC (NDEG, COEFF, ROOT)

Arguments

NDEG – Degree of the polynomial. $1 \le$ NDEG < 50 (Input)

COEFF — Complex vector of length `NDEG` + 1 containing the coefficients of the polynomial in increasing order by degree.   (Input)
The polynomial is `COEFF(NDEG` + 1) `* Z**NDEG` + `COEFF(NDEG) * Z**(NDEG` – 1) + … + `COEFF`(1).

ROOT — Complex vector of length `NDEG` containing the zeros of the polynomial.   (Output)

Comments

Informational errors

Type	Code	
3	1	The first several coefficients of the polynomial are equal to zero. Several of the last roots will be set to machine infinity to compensate for this problem.
3	2	Fewer than `NDEG` zeros were found. The `ROOT` vector will contain the value for machine infinity in the locations that do not contain zeros.

Algorithm

Routine `ZPOCC` computes the n zeros of the polynomial

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0$$

where the coefficients a_i for $i = 0, 1, \ldots, n$ are real and n is the degree of the polynomial.

The routine `ZPOCC` uses the Jenkins-Traub three-stage complex algorithm (Jenkins and Traub 1970, 1972). The zeros are computed one at a time in roughly increasing order of modulus. As each zero is found, the polynomial is deflated to one of lower degree.

Example

This example finds the zeros of the third-degree polynomial

$$p(z) = z^3 - (3 + 6i)z^2 - (8 - 12i)z + 10$$

where z is a complex variable.

```
C                                 Declare variables
      INTEGER    NDEG
      PARAMETER  (NDEG=3)
C
      COMPLEX    COEFF(NDEG+1), ZERO(NDEG)
      EXTERNAL   WRCRN, ZPOCC
C                                 Set values of COEFF
C                                 COEFF = ( 10.0 +  0.0i )
C                                         ( -8.0 + 12.0i )
C                                         ( -3.0 -  6.0i )
C                                         (  1.0 +  0.0i )
C
```

```
      DATA COEFF/(10.0,0.0), (-8.0,12.0), (-3.0,-6.0), (1.0,0.0)/
C
      CALL ZPOCC (NDEG, COEFF, ZERO)
C
      CALL WRCRN ('The zeros found are', 1, NDEG, ZERO, 1, 0)
C
      END
```

Output

```
              The zeros found are
             1                 2                 3
( 1.000, 1.000)  ( 1.000, 2.000)  ( 1.000, 3.000)
```

ZANLY/DZANLY (Single/Double precision)

Find the zeros of a univariate complex function using Müller's method.

Usage

```
CALL ZANLY (F, ERRABS, ERRREL, NKNOWN, NNEW, NGUESS, ZINIT,
            ITMAX, Z, INFO)
```

Arguments

F — User-supplied `COMPLEX FUNCTION` to compute the value of the function of which the zeros will be found. The form is `F`(`Z`), where

`Z` — The complex value at which the function is evaluated.   (Input)
`Z` should not be changed by `F`.
`F` — The computed complex function value at the point `Z`.   (Output)
`F` must be declared `EXTERNAL` in the calling program.

ERRABS — First stopping criterion.   (Input)
Let `FP`(`Z`) = `F`(`Z`)/`P` where `P` = (`Z` – `Z`(1)) `*` (`Z` – `Z`(2)) `*`…`*` (`Z` – `Z`(`K` – 1)) and `Z`(1), …, `Z`(`K` – 1) are previously found zeros. If (`CABS`(`F`(`Z`))`.LE.ERRABS.AND.CABS`(`FP`(`Z`))`.LE.ERRABS`), then `Z` is accepted as a zero.

ERRREL — Second stopping criterion is the relative error.   (Input)
A zero is accepted if the difference in two successive approximations to this zero is within `ERRREL`. `ERRREL` must be less than 0.01; otherwise, 0.01 will be used.

NKNOWN — The number of previously known zeros, if any, that must be stored in `ZINIT`(1), …, `ZINIT`(`NKNOWN`) prior to entry to `ZANLY`.   (Input)
`NKNOWN` must be set equal to zero if no zeros are known.

NNEW — The number of new zeros to be found by `ZANLY`.   (Input)

NGUESS — The number of initial guesses provided.   (Input)
These guesses must be stored in `ZINIT`(`NKNOWN` + 1), …, `ZINIT`(`NKNOWN` + `NGUESS`). `NGUESS` must be set equal to zero if no guesses are provided.

ZINIT — A complex vector of length NKNOWN + NNEW. (Input)
ZINIT(1), …, ZINIT(NKNOWN) must contain the known zeros. ZINIT(NKNOWN + 1), …, ZINIT(NKNOWN + NNEW) may, on user option, contain initial guesses for the NNEW new zeros that are to be computed. If the user does not provide an initial guess, zero is used.

ITMAX — The maximum allowable number of iterations per zero. (Input)

Z — A complex vector of length NKNOWN + NNEW. (Output)
Z(1), …, Z(NKNOWN) contain the known zeros. Z(NKNOWN + 1), …, Z(NKNOWN + NNEW) contain the new zeros found by ZANLY. If ZINIT is not needed, ZINIT and Z can share the same storage locations.

INFO — An integer vector of length NKNOWN + NNEW. (Output)
INFO(J) contains the number of iterations used in finding the J-th zero when convergence was achieved. If convergence was not obtained in ITMAX iterations, INFO(J) will be greater than ITMAX.

Comments

1. Informational error

Type	Code	
3	1	Failure to converge within ITMAX iterations for at least one of the NNEW new roots.

2. Routine ZANLY always returns the last approximation for zero J in Z(J). If the convergence criterion is satisfied, then INFO(J) is less than or equal to ITMAX. If the convergence criterion is not satisfied, then INFO(J) is set to either ITMAX + 1 or ITMAX + K, with K greater than 1. INFO(J) = ITMAX + 1 indicates that ZANLY did not obtain convergence in the allowed number of iterations. In this case, the user may wish to set ITMAX to a larger value. INFO(J) = ITMAX + K means that convergence was obtained (on iteration K) for the deflated function FP(Z) = F(Z)/((Z – Z(1) … (Z – Z(J – 1))) but failed for F(Z). In this case, better initial guesses might help or it might be necessary to relax the convergence criterion.

Algorithm

Müller’s method with deflation is used. It assumes that the complex function $f(z)$ has at least two continuous derivatives. For more details, see Müller (1965).

Example

This example finds the zeros of the equation $f(z) = z^3 + 5z^2 + 9z + 45$, where z is a complex variable.

```
C                                 Declare variables
      INTEGER    INFO(3), ITMAX, NGUESS, NKNOWN, NNEW
      REAL       ERRABS, ERRREL
      COMPLEX    F, Z(3), ZINIT(3)
      EXTERNAL   F, WRCRN, ZANLY
C                                  Set the guessed zero values in ZINIT
C
C                                  ZINIT = (1.0+1.0i 1.0+1.0i 1.0+1.0i)
      DATA ZINIT/3*(1.0,1.0)/
C                                  Set values for all input parameters
      ERRABS = 0.0001
      ERRREL = 0.0001
      NKNOWN = 0
      NNEW   = 3
      NGUESS = 3
      ITMAX  = 100
C                                  Find the zeros of F
      CALL ZANLY (F, ERRABS, ERRREL, NKNOWN, NNEW, NGUESS, ZINIT,
     &            ITMAX, Z, INFO)
C                                  Print results
      CALL WRCRN ('The zeros are', 1, 3, Z, 1, 0)
      END
C                                  External complex function
      COMPLEX FUNCTION F (Z)
      COMPLEX    Z
C
      F = Z**3 + 5.0*Z**2 + 9.0*Z + 45.0
      RETURN
      END
```

Output

```
              The zeros are
             1                 2                 3
( 0.000, 3.000)  ( 0.000,-3.000)  (-5.000, 0.000)
```

ZBREN/DZBREN (Single/Double precision)

Find a zero of a real function that changes sign in a given interval.

Usage

CALL ZBREN (F, ERRABS, ERRREL, A, B, MAXFN)

Arguments

F — User-supplied FUNCTION to compute the value of the function of which a zero will be found. The form is F(X), where

X — The point at which the function is evaluated. (Input)
X should not be changed by F.
F — The computed function value at the point X. (Output)
F must be declared EXTERNAL in the calling program.

ERRABS — First stopping criterion. (Input)
A zero, B, is accepted if ABS(F(B)) is less than or equal to ERRABS. ERRABS may be set to zero.

ERRREL — Second stopping criterion is the relative error. (Input)
A zero is accepted if the change between two successive approximations to this zero is within `ERRREL`.

A — See `B`. (Input/Output)

B — On input, the user must supply two points, `A` and `B`, such that `F(A)` and `F(B)` are opposite in sign. (Input/Output)
On output, both `A` and `B` are altered. `B` will contain the best approximation to the zero of `F`.

MAXFN — On input, `MAXFN` specifies an upper bound on the number of function evaluations required for convergence. (Input/Output)
On output, `MAXFN` will contain the actual number of function evaluations used.

Comments

1. Informational error

Type	Code	
4	1	Failure to converge in `MAXFN` function evaluations.

2. On exit from `ZBREN` without any error message, `A` and `B` satisfy the following:
   $F(A)F(B) \le 0.0$
   $|F(B)| \le |F(A)|$, and
   either $|F(B)| \le ERRABS$ or
   $|A - B| \le \max(|B|, 0.1) * ERRREL$.

   The presence of 0.1 in the stopping criterion causes leading zeros to the right of the decimal point to be counted as significant digits. Scaling may be required in order to accurately determine a zero of small magnitude.

3. `ZBREN` is guaranteed to convergence within `K` function evaluations, where $K = (\ln((B - A)/D) + 1.0)^2$, and

$$\left(D = \min_{x \in (A,B)} \left(\max\left(|x|, 0.1\right) * ERRREL \right) \right)$$

   This is an upper bound on the number of evaluations. Rarely does the actual number of evaluations used by `ZBREN` exceed

$$\sqrt{K}$$

   `D` can be computed as follows:
   `P` = `AMAX1`(0.1, `AMIN1`(|`A`|, |`B`|))
   `IF`((`A` – 0.1) `*` (`B` – 0.1) < 0.0) `P` = 0.1,
   `D` = `P` `*` `ERRREL`

Algorithm

The algorithm used by ZBREN is a combination of linear interpolation, inverse quadratic interpolation, and bisection. Convergence is usually superlinear and is never much slower than the rate for the bisection method. See Brent (1971) for a more detailed account of this algorithm.

Example

This example finds a zero of the function

$$f(x) = x^2 + x - 2$$

in the interval (– 10.0, 0.0).

```
C                                  Declare variables
      REAL       ERRABS, ERRREL
C
      INTEGER    MAXFN, NOUT
      REAL       A, B, F
      EXTERNAL   F, UMACH, ZBREN
C                                  Set values of A, B, ERRABS,
C                                  ERRREL, MAXFN
      A      = -10.0
      B      = 0.0
      ERRABS = 0.0
      ERRREL = 0.001
      MAXFN  = 100
C
      CALL UMACH (2, NOUT)
C                                  Find zero of F
      CALL ZBREN (F, ERRABS, ERRREL, A, B, MAXFN)
C
      WRITE (NOUT,99999) B, MAXFN
99999 FORMAT (’  The best approximation to the zero of F is equal to’,
     &       F5.1, ’.’, /, ’  The number of function evaluations’
     &       ’ required was ’, I2, ’.’, //)
C
      END
C
      REAL FUNCTION F (X)
      REAL       X
C
      F = X**2 + X - 2.0
      RETURN
      END
```

Output

```
The best approximation to the zero of F is equal to -2.0. The number of
function evaluations required was 12.
```

ZREAL/DZREAL (Single/Double precision)

Find the real zeros of a real function using Müller's method.

Usage

```
CALL ZREAL (F, ERRABS, ERRREL, EPS, ETA, NROOT, ITMAX,
            XGUESS, X, INFO)
```

Arguments

F — User-supplied `FUNCTION` to compute the value of the function of which a zero will be found. The form is `F(X)`, where

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `F`.
`F` – The computed function value at the point `X`. (Output)
`F` must be declared `EXTERNAL` in the calling program.

ERRABS — First stopping criterion. (Input)
A zero `X(I)` is accepted if `ABS(F(X(I)).LT. ERRABS`.

ERRREL — Second stopping criterion is the relative error. (Input)
A zero `X(I)` is accepted if the relative change of two successive approximations to `X(I)` is less than `ERRREL`.

EPS — See `ETA`. (Input)

ETA — Spread criteria for multiple zeros. (Input)
If the zero `X(I)` has been computed and `ABS(X(I) – X(J)).LT.EPS`, where `X(J)` is a previously computed zero, then the computation is restarted with a guess equal to `X(I)` + `ETA`.

NROOT — The number of zeros to be found by `ZREAL`. (Input)

ITMAX — The maximum allowable number of iterations per zero. (Input)

XGUESS — A vector of length `NROOT`. (Input)
`XGUESS` contains the initial guesses for the zeros.

X — A vector of length `NROOT`. (Output)
`X` contains the computed zeros.

INFO — An integer vector of length `NROOT`. (Output)
`INFO(J)` contains the number of iterations used in finding the `J`-th zero when convergence was achieved. If convergence was not obtained in `ITMAX` iterations, `INFO(J)` will be greater than `ITMAX`.

Comments

1. Informational error

Type	Code	
3	1	Failure to converge within `ITMAX` iterations for at least one of the `NROOT` roots.

2. Routine `ZREAL` always returns the last approximation for zero `J` in `X(J)`. If the convergence criterion is satisfied, then `INFO(J)` is less than or equal to `ITMAX`. If the convergence criterion is not satisfied, then `INFO(J)` is set to `ITMAX` + 1.

3. The routine `ZREAL` assumes that there exist `NROOT` distinct real zeros for the function `F` and that they can be reached from the initial guesses supplied. The routine is designed so that convergence to any single zero cannot be obtained from two different initial guesses.

4. Scaling the `X` vector in the function `F` may be required, if any of the zeros are known to be less than one.

Algorithm

Routine `ZREAL` computes *n* real zeros of a real function *f*. Given a user-supplied function *f*(*x*) and an *n*-vector of initial guesses x_1, x_2, …, x_n, the routine uses Müller’s method to locate *n* real zeros of *f*, that is, *n* real values of *x* for which *f*(*x*) = 0. The routine has two convergence criteria: the first requires that

$$\left| f\left(x_i^m\right)\right|$$

be less than `ERRABS`; the second requires that the relative change of any two successive approximations to an x_i be less than `ERRREL`. Here,

$$x_i^m$$

is the *m*-th approximation to x_i. Let `ERRABS` be ε_1, and `ERRREL` be ε_2.The criteria may be stated mathematically as follows:

Criterion 1:

$$\left| f\left(x_i^m\right)\right| < \varepsilon_1$$

Criterion 2:

$$\left| \frac{x_i^{m+1} - x_i^m}{x_i^m} \right| < \varepsilon_2$$

“Convergence” is the satisfaction of either criterion.

Example

This example finds the real zeros of the second-degree polynomial

$$f(x) = x^2 + 2x - 6$$

with the initial guess (4.6, –193.3).

```
C                                  Declare variables
      INTEGER    ITMAX, NROOT
      REAL       EPS, ERRABS, ERRREL, ETA
      PARAMETER  (NROOT=2)
C
      INTEGER    INFO(NROOT)
      REAL       F, X(NROOT), XGUESS(NROOT)
      EXTERNAL   F, WRRRN, ZREAL
C                                  Set values of initial guess
C                                  XGUESS = (  4.6 -193.3)
C
      DATA XGUESS/4.6, -193.3/
C
      EPS    = 1.0E-5
      ERRABS = 1.0E-5
      ERRREL = 1.0E-5
      ETA    = 1.0E-2

      ITMAX  = 100
C                                  Find the zeros
      CALL ZREAL (F, ERRABS, ERRREL, EPS, ETA, NROOT, ITMAX, XGUESS,
     &            X, INFO)
C
      CALL WRRRN (’The zeros are’, 1, NROOT, X, 1, 0)
C
      END
C
      REAL FUNCTION F (X)
      REAL       X
C
      F = X*X + 2.0*X - 6.0
      RETURN
      END
```

Output

```
The zeros are
    1       2
1.646  -3.646
```

NEQNF/DNEQNF (Single/Double precision)

Solve a system of nonlinear equations using a modified Powell hybrid algorithm and a finite-difference approximation to the Jacobian.

Usage

```
CALL NEQNF (FCN, ERRREL, N, ITMAX, XGUESS, X, FNORM)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the system of equations to be solved. The usage is CALL FCN (X, F, N), where

X – The point at which the functions are evaluated. (Input)
X should not be changed by FCN.
F – The computed function values at the point X. (Output)
N – Length of X and F. (Input)

FCN must be declared EXTERNAL in the calling program.

ERRREL — Stopping criterion. (Input)
The root is accepted if the relative error between two successive approximations to this root is less than ERRREL.

N — The number of equations to be solved and the number of unknowns. (Input)

ITMAX — The maximum allowable number of iterations. (Input)
The maximum number of calls to FCN is ITMAX * (N + 1). Suggested value ITMAX = 200.

XGUESS — A vector of length N. (Input)
XGUESS contains the initial estimate of the root.

X — A vector of length N. (Output)
X contains the best estimate of the root found by NEQNF.

FNORM — A scalar that has the value $F(1)^2 + \ldots + F(N)^2$ at the point X. (Output)

Comments

1. Automatic workspace usage is

   NEQNF 1.5 * N^2 + 7.5 * N units, or
   DNEQNF 3 * N^2 + 15 * N units.

   Workspace may be explicitly provided, if desired, by use of N2QNF/DN2QNF. The reference is

   ```
   CALL N2QNF (FCN, ERRREL, N, ITMAX, XGUESS, X, FNORM,
               FVEC, FJAC, R, QTF, WK)
   ```

   The additional arguments are as follows:

   FVEC — A vector of length N. FVEC contains the functions evaluated at the point X.

   FJAC — An N by N matrix. FJAC contains the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

R — A vector of length `N * (N + 1)/2`. `R` contains the upper triangular matrix produced by the `QR` factorization of the final approximate Jacobian. `R` is stored row-wise.

QTF — A vector of length `N`. `QTF` contains the vector `TRANS(Q) * FVEC`.

WK — A work vector of length `5 * N`.

2. Informational errors

Type	Code	
4	1	The number of calls to `FCN` has exceeded `ITMAX * (N + 1)`. A new initial guess may be tried.
4	2	`ERRREL` is too small. No further improvement in the approximate solution is possible.
4	3	The iteration has not made good progress. A new initial guess may be tried.

Algorithm

Routine `NEQNF` is based on the MINPACK subroutine `HYBRD1`, which uses a modification of M.J.D. Powell's hybrid algorithm. This algorithm is a variation of Newton's method, which uses a finite-difference approximation to the Jacobian and takes precautions to avoid large step sizes or increasing residuals. For further description, see More et al. (1980).

Since a finite-difference method is used to estimate the Jacobian, for single precision calculation, the Jacobian may be so incorrect that the algorithm terminates far from a root. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, IMSL routine `NEQNJ` (page 851) should be used instead.

Example

The following 3 × 3 system of nonlinear equations

$$f_1(x) = x_1 + e^{x_1 - 1} + \left(x_2 + x_3\right)^2 - 27 = 0$$

$$f_2(x) = e^{x_2 - 2} / x_1 + x_3^2 - 10 = 0$$

$$f_3(x) = x_3 + \sin\left(x_2 - 2\right) + x_2^2 - 7 = 0$$

is solved with the initial guess (4.0, 4.0, 4.0).

```
C                                 Declare variables
      INTEGER    ITMAX, N
      REAL       ERRREL
      PARAMETER  (N=3)
C
      INTEGER    K, NOUT
      REAL       FNORM, X(N), XGUESS(N)
      EXTERNAL   FCN, NEQNF, UMACH
```

```
C                                 Set values of initial guess
C                                 XGUESS = (  4.0  4.0  4.0 )
C
      DATA XGUESS/4.0, 4.0, 4.0/
C
      ERRREL = 0.0001
      ITMAX  = 100
C
      CALL UMACH (2, NOUT)
C                                 Find the solution
      CALL NEQNF (FCN, ERRREL, N, ITMAX, XGUESS, X, FNORM)
C                                 Output
      WRITE (NOUT,99999) (X(K),K=1,N), FNORM
99999 FORMAT ('  The solution to the system is', /, '  X = (', 3F5.1,
     &       ')', /, '  with FNORM =', F5.4, //)
C
      END
C                                 User-defined subroutine
      SUBROUTINE FCN (X, F, N)
      INTEGER    N
      REAL       X(N), F(N)
C
      REAL       EXP, SIN
      INTRINSIC  EXP, SIN
C
      F(1) = X(1) + EXP(X(1)-1.0) + (X(2)+X(3))*(X(2)+X(3)) - 27.0
      F(2) = EXP(X(2)-2.0)/X(1) + X(3)*X(3) - 10.0
      F(3) = X(3) + SIN(X(2)-2.0) + X(2)*X(2) - 7.0
      RETURN
      END
```

Output

```
The solution to the system is
X = (  1.0  2.0  3.0)
with FNORM =.0000
```

NEQNJ/DNEQNJ (Single/Double precision)

Solve a system of nonlinear equations using a modified Powell hybrid algorithm with a user-supplied Jacobian.

Usage

CALL NEQNJ (FCN, LSJAC, ERRREL, N, ITMAX, XGUESS, X, FNORM)

Arguments

FCN — User-supplied SUBROUTINE to evaluate the system of equations to be solved. The usage is CALL FCN (X, F, N), where

X – The point at which the functions are evaluated. (Input)
X should not be changed by FCN.
F – The computed function values at the point X. (Output)
N – Length of X, F. (Input)

FCN must be declared EXTERNAL in the calling program.

LSJAC — User-supplied SUBROUTINE to evaluate the Jacobian at a point X. The usage is CALL LSJAC (N, X, FJAC), where

N – Length of X. (Input)
X – The point at which the function is evaluated. (Input)
X should not be changed by LSJAC.
FJAC — The computed N by N Jacobian at the point X. (Output)

LSJAC must be declared EXTERNAL in the calling program.

ERRREL — Stopping criterion. (Input)
The root is accepted if the relative error between two successive approximations to this root is less than ERRREL.

N — The number of equations to be solved and the number of unknowns. (Input)

ITMAX — The maximum allowable number of iterations. (Input)
Suggested value = 200.

XGUESS — A vector of length N. (Input)
XGUESS contains the initial estimate of the root.

X — A vector of length N. (Output)
X contains the best estimate of the root found by NEQNJ.

FNORM — A scalar that has the value F(1)^2 + … + F(N)^2 at the point X. (Output)

Comments

1. Automatic workspace usage is

NEQNJ 1.5 * N**2 + 7.5 * N units, or
DNEQNJ 3 * N**2 + 15 * N units.

Workspace may be explicitly provided, if desired, by use of N2QNJ/DN2QNJ. The reference is

```
CALL N2QNJ (FCN, LSJAC, ERRREL, N, ITMAX, XGUESS, X,
            FNORM, FVEC, FJAC, R, QTF, WK)
```

The additional arguments are as follows:

FVEC — A vector of length N. FVEC contains the functions evaluated at the point X.

FJAC — An N by N matrix. FJAC contains the orthogonal matrix Q produced by the QR factorization of the final approximate Jacobian.

R — A vector of length N * (N + 1)/2. R contains the upper triangular matrix produced by the QR factorization of the final approximate Jacobian. R is stored row-wise.

QTF — A vector of length N. QTF contains the vector TRANS(Q) * FVEC.

WK — A work vector of length 5 * N.

2. Informational errors

Type	Code	
4	1	The number of calls to FCN has exceeded ITMAX. A new initial guess may be tried.
4	2	ERRREL is too small. No further improvement in the approximate solution is possible.
4	3	The iteration has not made good progress. A new initial guess may be tried.

Algorithm

Routine NEQNJ is based on the MINPACK subroutine HYBRDJ, which uses a modification of M.J.D. Powell's hybrid algorithm. This algorithm is a variation of Newton's method, which takes precautions to avoid large step sizes or increasing residuals. For further description, see More et al. (1980).

Example

The following 3 × 3 system of nonlinear equations

$$f_1(x) = x_1 + e^{x_1 - 1} + \left(x_2 + x_3\right)^2 - 27 = 0$$

$$f_2(x) = e^{x_2 - 2} / x_1 + x_3^2 - 10 = 0$$

$$f_3(x) = x_3 + \sin\left(x_2 - 2\right) + x_2^2 - 7 = 0$$

is solved with the initial guess (4.0, 4.0, 4.0).

```
C                                 Declare variables
      INTEGER    ITMAX, N
      REAL       ERRREL
      PARAMETER  (N=3)
C
      INTEGER    K, NOUT
      REAL       FNORM, X(N), XGUESS(N)
      EXTERNAL   FCN, LSJAC, NEQNJ, UMACH
C                                 Set values of initial guess
C                                 XGUESS = (  4.0  4.0  4.0  )
C
      DATA XGUESS/4.0, 4.0, 4.0/
C
      ERRREL = 0.0001
      ITMAX  = 100
C
      CALL UMACH (2, NOUT)
C                                 Find the solution
      CALL NEQNJ (FCN, LSJAC, ERRREL, N, ITMAX, XGUESS, X, FNORM)
C                                 Output
      WRITE (NOUT,99999) (X(K),K=1,N), FNORM
```

```
99999 FORMAT ('  The roots found are', /, '  X = (', 3F5.1,
     &       ')', /, '  with FNORM =',F5.4, //)
C
      END
C                                  User-supplied subroutine
      SUBROUTINE FCN (X, F, N)
      INTEGER    N
      REAL       X(N), F(N)
C
      REAL       EXP, SIN
      INTRINSIC  EXP, SIN
C
      F(1) = X(1) + EXP(X(1)-1.0) + (X(2)+X(3))*(X(2)+X(3)) - 27.0
      F(2) = EXP(X(2)-2.0)/X(1) + X(3)*X(3) - 10.0
      F(3) = X(3) + SIN(X(2)-2.0) + X(2)*X(2) - 7.0
      RETURN
      END
C                                  User-supplied subroutine to
C                                  compute Jacobian
      SUBROUTINE LSJAC (N, X, FJAC)
      INTEGER    N
      REAL       X(N), FJAC(N,N)
C
      REAL       COS, EXP
      INTRINSIC  COS, EXP
C
      FJAC(1,1) = 1.0 + EXP(X(1)-1.0)
      FJAC(1,2) = 2.0*(X(2)+X(3))
      FJAC(1,3) = 2.0*(X(2)+X(3))
      FJAC(2,1) = -EXP(X(2)-2.0)*(1.0/X(1)**2)
      FJAC(2,2) = EXP(X(2)-2.0)*(1.0/X(1))
      FJAC(2,3) = 2.0*X(3)
      FJAC(3,1) = 0.0
      FJAC(3,2) = COS(X(2)-2.0) + 2.0*X(2)
      FJAC(3,3) = 1.0
      RETURN
      END
```

Output

```
The roots found are
X = (  1.0  2.0  3.0)
with FNORM =.0000
```

NEQBF/DNEQBF (Single/Double precision)

Solve a system of nonlinear equations using factored secant update with a finite-difference approximation to the Jacobian.

Usage

```
CALL NEQBF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM,
            X, FVEC)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the system of equations to be solved. The usage is CALL FCN (N, X, F), where

N – Length of X and F.   (Input)
X – The point at which the functions are evaluated.   (Input)
X should not be changed by FCN.
F – The computed function values at the point X.   (Output)

FCN must be declared EXTERNAL in the calling program.

N — Dimension of the problem.   (Input)

XGUESS — Vector of length N containing initial guess of the root.   (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables.   (Input)
XSCALE is used mainly in scaling the distance between two points. In the absence of other information, set all entries to 1.0. If internal scaling is desired for XSCALE, set IPARAM (6) to 1.

FSCALE — Vector of length N containing the diagonal scaling matrix for the functions.   (Input)
FSCALE is used mainly in scaling the function residuals. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6.   (Input/Output)
Set IPARAM (1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 5.   (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution.   (Output)

FVEC — Vector of length N containing the values of the functions at the approximate solution.   (Output)

Comments

1. Automatic workspace usage is

   NEQBF $2N^2 + 11 * N$ units, or
   DNEQBF $4N^2 + 22 * N$ units.

   Workspace may be explicitly provided, if desired, by use of N2QBF/DN2QBF. The reference is

   ```
   CALL N2QBF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM,
                RPARAM, X, FVEC, WK, LWK)
   ```

   The additional arguments are as follows:

WK — A work vector of length LWK. On output WK contains the following information:

The third N locations contain the last step taken.

The fourth N locations contain the last Newton step.

The final N^2 locations contain an estimate of the Jacobian at the solution.

LWK — Length of WK, which must be at least 2 * N^2 + 11 * N. (Input)

2. Informational errors

Type	Code	
3	1	The last global step failed to decrease the 2-norm of F(X) sufficiently; either the current point is close to a root of F(X) and no more accuracy is possible, or the secant approximation to the Jacobian is inaccurate, or the step tolerance is too large.
3	3	The scaled distance between the last two steps is less than the step tolerance; the current point is probably an approximate root of F(X) (unless STEPTL is too large).
3	4	Maximum number of iterations exceeded.
3	5	Maximum number of function evaluations exceeded.
3	7	Five consecutive steps of length STEPMX have been taken; either the 2-norm of F(X) asymptotes from above to a finite value in some direction or the maximum allowable step size STEPMX is too small.

3. The stopping criterion for NEQBF occurs when the scaled norm of the functions is less than the scaled function tolerance (RPARAM(1)).

4. If the default parameters are desired for NEQBF, then set IPARAM(1) to zero and call routine NEQBF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling NEQBF:

   CALL N4QBJ (IPARAM, RPARAM)
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to N4QBJ will set IPARAM and RPARAM to their default values, so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

   IPARAM — Integer vector of length 6.

   IPARAM(1) = Initialization flag.

   IPARAM(2) = Number of good digits in the function.
   Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of Jacobian evaluations.
Default: not used in NEQBF.

IPARAM(6) = Internal variable scaling flag.
If IPARAM(6) = 1, then the values of XSCALE are set internally.
Default: 0.

RPARAM — Real vector of length 5.

RPARAM(1) = Scaled function tolerance.
The scaled norm of the functions is computed as

$$\max_i \left(\left| f_i \right| * fs_i \right)$$

where f_i is the *i*-th component of the function vector F, and fs_i is the *i*-th component of FSCALE.
Default:

$$\sqrt{\varepsilon}$$

where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The scaled norm of the step between two points *x* and *y* is computed as

$$\max_i \left\{ \frac{\left| x_i - y_i \right|}{\max\left(\left| x_i \right|, 1/s_i \right)} \right\}$$

where s_i is the *i*-th component of XSCALE.
Default: $\varepsilon^{2/3}$, where ε is the machine precision.

RPARAM(3) = False convergence tolerance.
Default: not used in NEQBF.

RPARAM(4) = Maximum allowable step size. (STEPMX)

Default: 1000 * $\max(\varepsilon_1, \varepsilon_2)$, where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i \right)^2}$$

$\varepsilon_2 = \|s\|_2$, s = XSCALE, and t = XGUESS.

RPARAM(5) = Size of initial trust region.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DN4QBJ` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

Routine `NEQBF` uses a secant algorithm to solve a system of nonlinear equations, i.e.,

$$F(x) = 0$$

where $F : \mathbf{R}^n \to \mathbf{R}^n$, and $x \in \mathbf{R}^n$.

From a current point, the algorithm uses a double dogleg method to solve the following subproblem approximately:

$$\min_{s \in \mathbf{R}^n} \left\| F(x_c) + J(x_c)s \right\|_2$$

$$\text{subject to } \| s \|_2 \le \delta_c$$

to get a direction s_c, where $F(x_c)$ and $J(x_c)$ are the function values and the approximate Jacobian respectively evaluated at the current point x_c. Then, the function values at the point $x_n = x_c + s_c$ are evaluated and used to decide whether the new point x_n should be accepted.

When the point x_n is rejected, this routine reduces the trust region δ_c and goes back to solve the subproblem again. This procedure is repeated until a better point is found.

The algorithm terminates if the new point satisfies the stopping criterion. Otherwise, δ_c is adjusted, and the approximate Jacobian is updated by Broyden’s formula,

$$J_n = J_c + \frac{(y - J_c s_c) s_c^T}{s_c^T s_c}$$

where $J_n = J(x_n)$, $J_c = J(x_c)$, and $y = F(x_n) - F(x_c)$. The algorithm then continues using the new point as the current point, i.e. $x_c \leftarrow x_n$.

For more details, see Dennis and Schnabel (1983, Chapter 8).

Since a finite-difference method is used to estimate the initial Jacobian, for single precision calculation, the Jacobian may be so incorrect that the algorithm terminates far from a root. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, IMSL routine `NEQBJ` (page 860) should be used instead.

Example

The following 3 × 3 system of nonlinear equations:

$$f_1(x) = x_1 + e^{x_1 - 1} + (x_2 + x_3)^2 - 27 = 0$$

$$f_2(x) = e^{x_2 - 2} / x_1 + x_3^2 - 10 = 0$$

$$f_3(x) = x_3 + \sin(x_2 - 2) + x_2^2 - 7 = 0$$

is solved with the initial guess (4.0, 4.0, 4.0).

```
C                                 Declare variables
      INTEGER    N
      PARAMETER  (N=3)
C
      INTEGER    IPARAM(6), K, NOUT
      REAL       FCN, FSCALE(N), FVEC(N), RPARAM(5), X(N), XGUESS(N),
     &           XSCALE(N)
      EXTERNAL   FCN, NEQBF, UMACH
C                                 Set values of initial guess
C                                 XGUESS = (  4.0  4.0  4.0 )
C
      DATA XGUESS/3*4.0/, XSCALE/3*1.0/, FSCALE/3*1.0/
C
C                                 Use the default setting
C
      IPARAM(1) = 0
C                                 Find the solution
      CALL NEQBF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM, X,
     &           FVEC)
C                                 Output
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N)
99999 FORMAT (’  The solution to the system is’, /, ’  X = (’, 3F8.3,
     &       ’)’)
C
      END
C                                 User-defined subroutine
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F(N)
C
      REAL       EXP, SIN
      INTRINSIC  EXP, SIN
C
      F(1) = X(1) + EXP(X(1)-1.0) + (X(2)+X(3))*(X(2)+X(3)) - 27.0
      F(2) = EXP(X(2)-2.0)/X(1) + X(3)*X(3) - 10.0
      F(3) = X(3) + SIN(X(2)-2.0) + X(2)*X(2) - 7.0
      RETURN
      END
```

Output

```
The solution to the system is
X = (   1.000   2.000   3.000)
```

NEQBJ/DNEQBJ (Single/Double precision)

Solve a system of nonlinear equations using factored secant update with a user-supplied Jacobian.

Usage

```
CALL NEQBJ (FCN, JAC, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X, FVEC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the system of equations to be solved. The usage is `CALL FCN (N, X, F)`, where
`N` – Length of `X` and `F`. (Input)
`X` – The point at which the functions are evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function values at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (N, X, FJAC, LDFJAC)`, where
`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the Jacobian is evaluated. (Input)
`X` should not be changed by `JAC`.
`FJAC` – The computed `N` by `N` Jacobian at the point `X`. (Output)
`LDFJAC` – Leading dimension of `FJAC`. (Input)

`JAC` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length `N` containing initial guess of the root. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the distance between two points. In the absence of other information, set all entries to 1.0. If internal scaling is desired for `XSCALE`, set `IPARAM`(6) to 1.

FSCALE — Vector of length `N` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the function residuals. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set `IPARAM` (1) to zero for default values of `IPARAM` and `RPARAM`.
See Comment 4.

RPARAM — Parameter vector of length 5. (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution. (Output)

FVEC — Vector of length N containing the values of the functions at the approximate solution. (Output)

Comments

1. Automatic workspace usage is

   NEQBJ 2 * N^2 + 11 * N units, or
   DNEQBJ 4 * N^2 + 22 * N units.

   Workspace may be explicitly provided, if desired, by use of N2QBJ/DN2QBJ. The reference is

```
CALL N2QBJ (FCN, JAC, N, XGUESS, XSCALE, FSCALE,
            IPARAM, RPARAM, X, FVEC, WK, LWK)
```

   The additional arguments are as follows:

   WK — A work vector of length LWK. On output WK contains the following information: The third N locations contain the last step taken. The fourth N locations contain the last Newton step. The final N^2 locations contain an estimate of the Jacobian at the solution.

   LWK — Length of WK, which must be at least 2 * N^2 + 11 * N. (Input)

2. Informational errors

Type	Code	
3	1	The last global step failed to decrease the 2-norm of F(X) sufficiently; either the current point is close to a root of F(X) and no more accuracy is possible, or the secant approximation to the Jacobian is inaccurate, or the step tolerance is too large.
3	3	The scaled distance between the last two steps is less than the step tolerance; the current point is probably an approximate root of F(X) (unless STEPTL is too large).
3	4	Maximum number of iterations exceeded.
3	5	Maximum number of function evaluations exceeded.
3	7	Five consecutive steps of length STEPMX have been taken; either the 2-norm of F(X) asymptotes from above to a finite value in some direction or the maximum allowable stepsize STEPMX is too small.

3. The stopping criterion for NEQBJ occurs when the scaled norm of the functions is less than the scaled function tolerance (RPARAM(1)).

4. If the default parameters are desired for NEQBJ, then set IPARAM(1) to zero and call routine NEQBJ. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling NEQBJ:

```
CALL N4QBJ (IPARAM, RPARAM)
```
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to N4QBJ will set IPARAM and RPARAM to their default values, so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.

IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of Jacobian evaluations.
Default: not used in NEQBJ.

IPARAM(6) = Internal variable scaling flag.
If IPARAM(6) = 1, then the values of XSCALE are set internally.
Default: 0.

RPARAM — Real vector of length 5.

RPARAM(1) = Scaled function tolerance.
The scaled norm of the functions is computed as

$$\max_i \left(\left|f_i\right| * fs_i\right)$$

where f_i is the i-th component of the function vector F, and fs_i is the i-th component of FSCALE.
Default:

$$\sqrt{\varepsilon}$$

where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The scaled norm of the step between two points x and y is computed as

$$\max_i \{\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}\}$$

where s_i is the i-th component of `XSCALE`.

Default: $\varepsilon^{2/3}$, where ε is the machine precision.

`RPARAM`(3) = False convergence tolerance.
Default: not used in `NEQBJ`.

`RPARAM`(4) = Maximum allowable step size. (`STEPMX`)

Default: 1000 `*` max(ε_1, ε_2), where

$$\varepsilon_1 = \sqrt{\sum\nolimits_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \|s\|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(5) = Size of initial trust region.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DN4QBJ` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to "Error Handling" in the Introduction.

Algorithm

Routine `NEQBJ` uses a secant algorithm to solve a system of nonlinear equations, i. e.,

$$F\,(x) = 0$$

where $F : \mathbf{R}^n \rightarrow \mathbf{R}^n$, and $x \in \mathbf{R}^n$.

From a current point, the algorithm uses a double dogleg method to solve the following subproblem approximately:

$$\min_{s \in \mathbf{R}^n} \left\| F\left(x_c\right) + J\left(x_c\right)s \right\|_2$$

$$\text{subject to } \|s\|_2 \leq \delta_c$$

to get a direction s_c, where $F(x_c)$ and $J(x_c)$ are the function values and the approximate Jacobian respectively evaluated at the current point x_c. Then, the function values at the point $x_n = x_c + s_c$ are evaluated and used to decide whether the new point x_n should be accepted.

When the point x_n is rejected, this routine reduces the trust region δ_c and goes back to solve the subproblem again. This procedure is repeated until a better point is found.

The algorithm terminates if the new point satisfies the stopping criterion. Otherwise, δ_c is adjusted, and the approximate Jacobian is updated by Broyden's formula,

$$J_n = J_c + \frac{(y - J_c s_c) s_c^T}{s_c^T s_c}$$

where $J_n = J(x_n)$, $J_c = J(x_c)$, and $y = F(x_n) - F(x_c)$. The algorithm then continues using the new point as the current point, i.e. $x_c \leftarrow x_n$.

For more details, see Dennis and Schnabel (1983, Chapter 8).

Example

The following 3 × 3 system of nonlinear equations

$$f_1(x) = x_1 + e^{x_1 - 1} + (x_2 + x_3)^2 - 27 = 0$$
$$f_2(x) = e^{x_2 - 2} / x_1 + x_3^2 - 10 = 0$$
$$f_3(x) = x_3 + \sin(x_2 - 2) + x_2^2 - 7 = 0$$

is solved with the initial guess (4.0, 4.0, 4.0).

```
C                                 Declare variables
      INTEGER    N
      PARAMETER  (N=3)
C
      INTEGER    IPARAM(6), K, NOUT
      REAL       FCN, FSCALE(N), FVEC(N), JAC, RPARAM(5), X(N),
     &           XGUESS(N), XSCALE(N)
      EXTERNAL   FCN, JAC, NEQBJ, UMACH
C                                 Set values of initial guess
C                                 XGUESS = (  4.0  4.0  4.0 )
C
      DATA XGUESS/3*4.0/, XSCALE/3*1.0/, FSCALE/3*1.0/
C
C                                 Use the default setting
C
      IPARAM(1) = 0
C                                 Find the solution
      CALL NEQBJ (FCN, JAC, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM,
     &           X, FVEC)
C                                 Output
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N)
99999 FORMAT ('  The solution to the system is', /, '  X = (', 3F8.3,
     &       ')')
C
      END
C                                 User-defined subroutine
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F(N)
```

```
C
      REAL       EXP, SIN
      INTRINSIC  EXP, SIN
C
      F(1) = X(1) + EXP(X(1)-1.0) + (X(2)+X(3))*(X(2)+X(3)) - 27.0
      F(2) = EXP(X(2)-2.0)/X(1) + X(3)*X(3) - 10.0
      F(3) = X(3) + SIN(X(2)-2.0) + X(2)*X(2) - 7.0
      RETURN
      END
C                                  User-supplied subroutine to
C                                  compute Jacobian
      SUBROUTINE JAC (N, X, FJAC, LDFJAC)
      INTEGER    N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,*)
C
      REAL       COS, EXP
      INTRINSIC  COS, EXP
C
      FJAC(1,1) = 1.0 + EXP(X(1)-1.0)
      FJAC(1,2) = 2.0*(X(2)+X(3))
      FJAC(1,3) = 2.0*(X(2)+X(3))
      FJAC(2,1) = -EXP(X(2)-2.0)*(1.0/X(1)**2)
      FJAC(2,2) = EXP(X(2)-2.0)*(1.0/X(1))
      FJAC(2,3) = 2.0*X(3)
      FJAC(3,1) = 0.0
      FJAC(3,2) = COS(X(2)-2.0) + 2.0*X(2)
      FJAC(3,3) = 1.0
      RETURN
      END
```

Output

```
The solution to the system is
X = (   1.000   2.000   3.000)
```

Chapter 8: Optimization

Routines

Usage Notes

Unconstrained Minimization

The unconstrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is at least continuous. The routines for unconstrained minimization are grouped into three categories: univariate functions (UV***), multivariate functions (UM***), and nonlinear least squares (UNLS*).

For the univariate function routines, it is assumed that the function is unimodal within the specified interval. Otherwise, only a local minimum can be expected. For further discussion on unimodality, see Brent (1973).

A quasi-Newton method is used for the multivariate function routines UMINF (page 881) and UMING (page 886), whereas UMIDH (page 891) and UMIAH (page 896) use a modified Newton algorithm. The routines UMCGF (page 902) and UMCGG (page 905) make use of a conjugate gradient approach, and UMPOL (page 909) uses a polytope method. For more details on these algorithms, see the documentation for the corresponding routines.

The nonlinear least squares routines use a modified Levenberg-Marquardt algorithm. If the nonlinear least squares problem is a nonlinear data-fitting problem, then software that is designed to deliver better statistical output may be useful; see IMSL (1991).

These routines are designed to find only a local minimum point. However, a function may have many local minima. It is often possible to obtain a better local solution by trying different initial points and intervals.

High precision arithmetic is recommended for the routines that use only function values. Also it is advised that the derivative-checking routines CH*** be used to ensure the accuracy of the user-supplied derivative evaluation subroutines.

Minimization with Simple Bounds

The minimization with simple bounds problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l_i \le x_i \le u_i \text{, for } i = 1, 2, \ldots, n$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$, and all the variables are not necessarily bounded.

The routines `BCO**` use the same algorithms as the routines `UMI**`, and the routines `BCLS*` are the corresponding routines of `UNLS*`. The only difference is that an active set strategy is used to ensure that each variable stays within its bounds. The routine `BCPOL` (page 948) uses a function comparison method similar to the one used by `UMPOL` (page 909). Convergence for these polytope methods is not guaranteed; therefore, these routines should be used as a last alternative.

Linearly Constrained Minimization

The linearly constrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } Ax = b$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$, A is an $m \times n$ coefficient matrix, and b is a vector of length m. If $f(x)$ is linear, then the problem is a linear programming problem; if $f(x)$ is quadratic, the problem is a quadratic programming problem.

The routine `DLPRS` (page 973) uses a revised simplex method to solve small- to medium-sized linear programming problems. No sparsity is assumed since the coefficients are stored in full matrix form.

The routine `QPROG` (page 982) is designed to solve convex quadratic programming problems using a dual quadratic programming algorithm. If the given Hessian is not positive definite, then `QPROG` modifies it to be positive definite. In this case, output should be interpreted with care.

The routines `LCONF` (page 984) and `LCONG` (page 990) use an iterative method to solve the linearly constrained problem with a general objective function. For a detailed description of the algorithm, see Powell (1988, 1989).

Nonlinearly Constrained Minimization

The nonlinearly constrained minimization problem can be stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } g_i(x) = 0, \text{ for } \quad i = 1, 2, \ldots, m_1$$

$$g_i(x) \geq 0, \text{ for } \quad i = m_1 + 1, \ldots, m$$

where $f : \mathbf{R}^n \rightarrow \mathbf{R}$ and $g_i : \mathbf{R}^n \rightarrow \mathbf{R}$, for $i = 1, 2, \ldots, m$

The routines NCONF (page 996) and NCONG (page 1003) use a successive quadratic programming algorithm to solve this problem. A more complete discussion of this algorithm can be found in the documentation.

Selection of Routines

The following general guidelines are provided to aid in the selection of the appropriate routine.

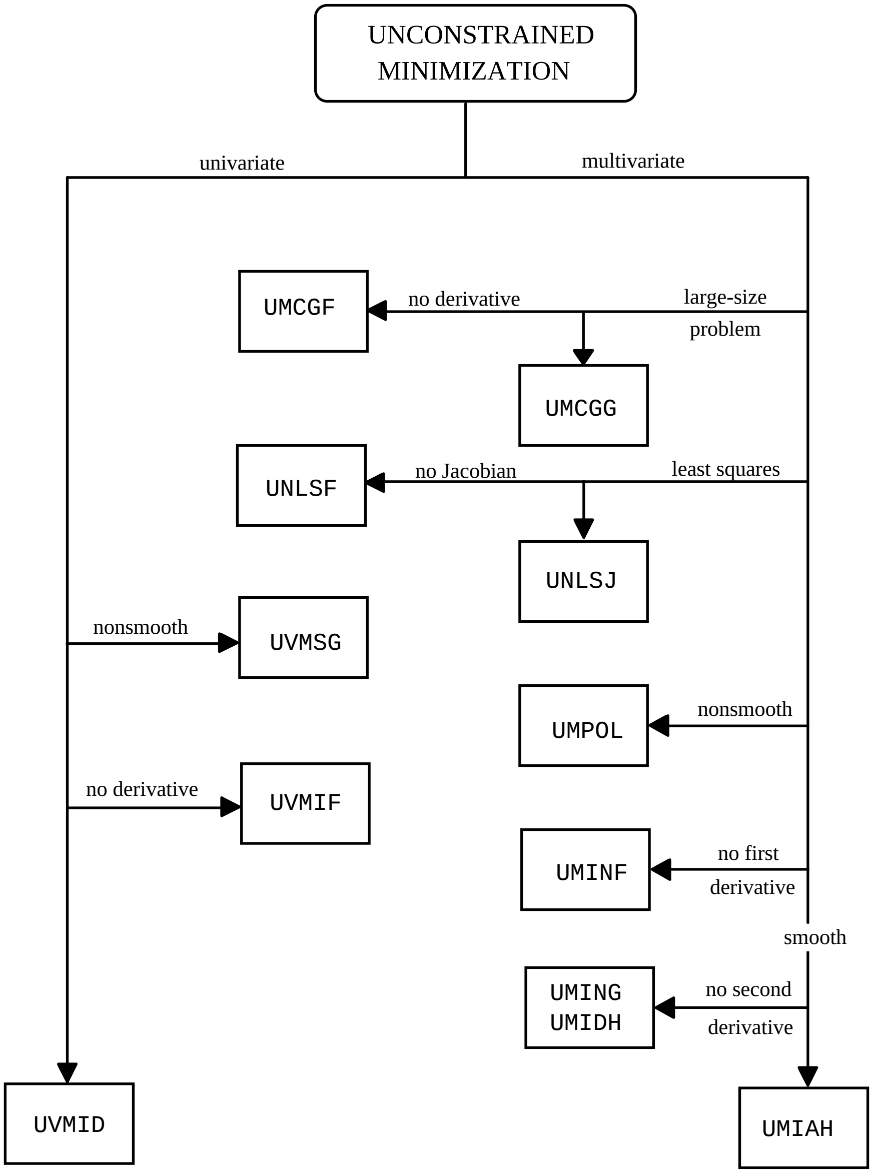
Unconstrained Minimization

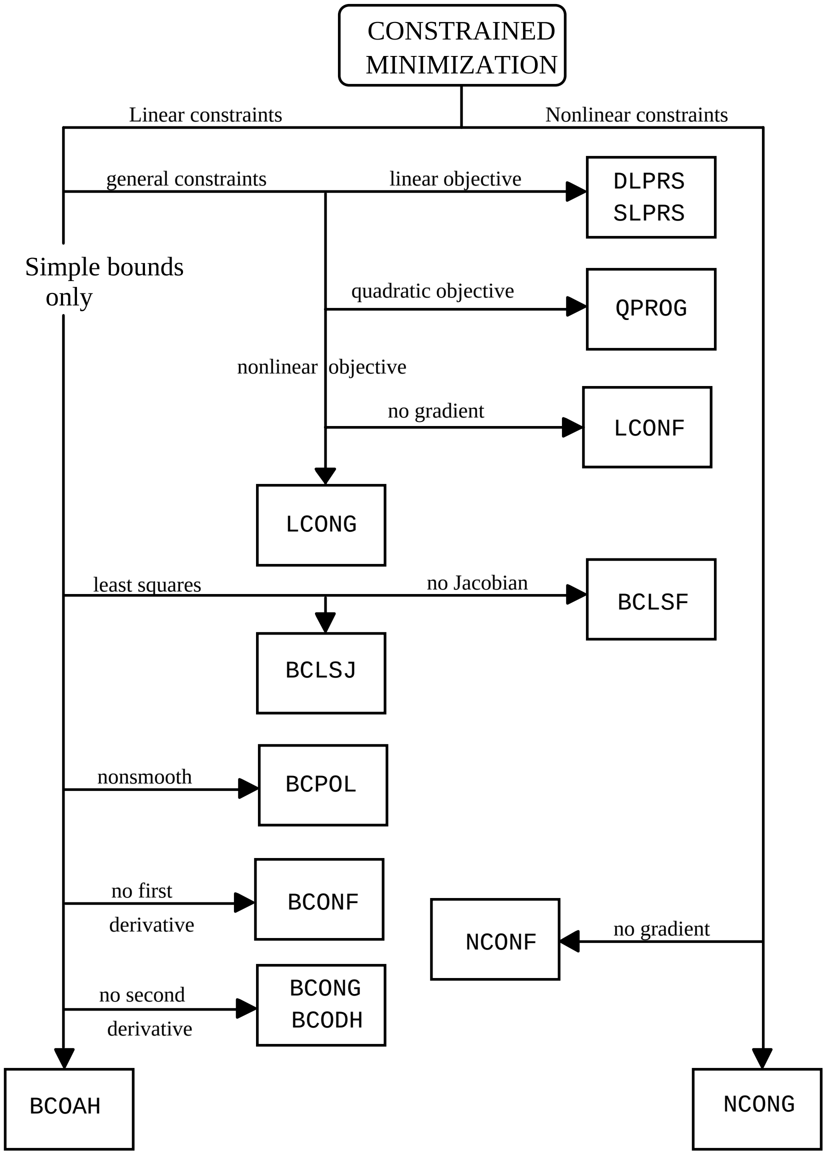
1. For the univariate case, use UVMID (page 875) when the gradient is available, and use UVMIF (page 872) when it is not. If discontinuities exist, then use UVMGS (page 878).
2. For the multivariate case, use UMCG* when storage is a problem, and use UMPOL (page 909) when the function is nonsmooth. Otherwise, use UMI** depending on the availability of the gradient and the Hessian.
3. For least squares problems, use UNLSJ (page 918) when the Jacobian is available, and use UNLSF (page 912) when it is not.

Minimization with Simple Bounds

1. Use BCONF (page 923) when only function values are available. When first derivatives are available, use either BCONG (page 930) or BCODH (page 936). If first and second derivatives are available, then use BCOAH (page 942).
2. For least squares, use BCLSF (page 952) or BCLSJ (page 958) depending on the availability of the Jacobian.
3. Use BCPOL (page 948) for nonsmooth functions that could not be solved satisfactorily by the other routines.

The following charts provide a quick reference to routines in this chapter:


UNCONSTRAINED
MINIMIZATION
univariate
multivariate
UMCGF
no derivative
large-size
problem
UMCGG
UNLSF
no Jacobian
least squares
UNLSJ
nonsmooth
UVMSG
UMPOL
nonsmooth
no derivative
UVMIF
UMINF
no first
derivative
smooth
UMING
UMIDH
no second
derivative
UVMID
UMIAH



UVMIF/DUVMIF (Single/Double precision)

Find the minimum point of a smooth function of a single variable using only function evaluations.

Usage

```
CALL UVMIF (F, XGUESS, STEP, BOUND, XACC, MAXFN, X)
```

Arguments

F — User-supplied `FUNCTION` to compute the value of the function to be minimized. The form is `F(X)`, where

`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `F`.
`F` – The computed function value at the point `X`. (Output)

`F` must be declared `EXTERNAL` in the calling program.

XGUESS — An initial guess of the minimum point of `F`. (Input)

STEP — An order of magnitude estimate of the required change in `X`. (Input)

BOUND — A positive number that limits the amount by which `X` may be changed from its initial value. (Input)

XACC — The required absolute accuracy in the final value of `X`. (Input)
On a normal return there are points on either side of `X` within a distance `XACC` at which `F` is no less than `F(X)`.

MAXFN — Maximum number of function evaluations allowed. (Input)

X — The point at which a minimum value of `F` is found. (Output)

Comments

Informational errors

Type	Code	
3	1	Computer rounding errors prevent further refinement of `X`.
3	2	The final value of `X` is at a bound. The minimum is probably beyond the bound.
4	3	The number of function evaluations has exceeded `MAXFN`.

Algorithm

The routine `UVMIF` uses a safeguarded quadratic interpolation method to find a minimum point of a univariate function. Both the code and the underlying algorithm are based on the routine `ZXLSF` written by M.J.D. Powell at the University of Cambridge.

The routine `UVMIF` finds the least value of a univariate function, f, that is specified by the function subroutine `F`. Other required data include an initial estimate of the solution, `XGUESS` , and a positive number `BOUND`. Let x_0 = `XGUESS` and b = `BOUND`, then x is restricted to the interval $[x_0 - b, x_0 + b]$. Usually, the algorithm begins the search by moving from x_0 to $x = x_0 + s$, where s = `STEP` is also provided by the user and may be positive or negative. The first two function evaluations indicate the direction to the minimum point, and the search strides out along this direction until a bracket on a minimum point is found or until x reaches one of the bounds $x_0 \pm b$. During this stage, the step length increases by a factor of between two and nine per function evaluation; the factor depends on the position of the minimum point that is predicted by quadratic interpolation of the three most recent function values.

When an interval containing a solution has been found, we will have three points, x_1, x_2, and x_3, with $x_1 < x_2 < x_3$ and $f(x_2) \le f(x_1)$ and $f(x_2) \le f(x_3)$. There are three main ingredients in the technique for choosing the new x from these three points. They are (i) the estimate of the minimum point that is given by quadratic interpolation of the three function values, (ii) a tolerance parameter ε, that depends on the closeness of f to a quadratic, and (iii) whether x_2 is near the center of the range between x_1 and x_3 or is relatively close to an end of this range. In outline, the new value of x is as near as possible to the predicted minimum point, subject to being at least ε from x_2, and subject to being in the longer interval between x_1 and x_2 or x_2 and x_3 when x_2 is particularly close to x_1 or x_3. There is some elaboration, however, when the distance between these points is close to the required accuracy; when the distance is close to the machine precision; or when ε is relatively large.

The algorithm is intended to provide fast convergence when f has a positive and continuous second derivative at the minimum and to avoid gross inefficiencies in pathological cases, such as

$$f(x) = x + 1.001|x|$$

The algorithm can make ε large automatically in the pathological cases. In this case, it is usual for a new value of x to be at the midpoint of the longer interval that is adjacent to the least calculated function value. The midpoint strategy is used frequently when changes to f are dominated by computer rounding errors, which will almost certainly happen if the user requests an accuracy that is less than the square root of the machine precision. In such cases, the routine claims to have achieved the required accuracy if it knows that there is a local minimum point within distance δ of x, where δ = XACC, even though the rounding errors in f may cause the existence of other local minimum points nearby. This difficulty is inevitable in minimization routines that use only function values, so high precision arithmetic is recommended.

Example

A minimum point of $e^x - 5x$ is found.

```
C                                  Declare variables
      INTEGER    MAXFN, NOUT
      REAL       BOUND, F, FX, STEP, X, XACC, XGUESS
      EXTERNAL   F, UMACH, UVMIF
C                                  Initialize variables
      XGUESS = 0.0
      XACC   = 0.001
      BOUND  = 100.0
      STEP   = 0.1
      MAXFN  = 50
C
C                                 Find minimum for F = EXP(X) - 5X
      CALL UVMIF (F, XGUESS, STEP, BOUND, XACC, MAXFN, X)
      FX = F(X)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FX
C
99999 FORMAT ('   The minimum is at ', 7X, F7.3, //, '   The function '
     &       , 'value is ', F7.3)
C
      END
C                                 Real function: F = EXP(X) - 5.0*X
      REAL FUNCTION F (X)
      REAL       X
C
      REAL       EXP
      INTRINSIC  EXP
C
      F = EXP(X) - 5.0E0*X
C
      RETURN
      END
```

Output

```
The minimum is at          1.609

The function value is  -3.047
```

UVMID/DUVMID (Single/Double precision)

Find the minimum point of a smooth function of a single variable using both function evaluations and first derivative evaluations.

Usage

```
CALL UVMID (F, G, XGUESS, ERRREL, GTOL, MAXFN, A, B, X, FX,
            GX)
```

Arguments

F — User-supplied `FUNCTION` to define the function to be minimized. The form is `F(X)`, where

`X` – The point at which the function is to be evaluated.   (Input)
`F` – The computed value of the function at `X`.   (Output)

`F` must be declared `EXTERNAL` in the calling program.

G — User-supplied `FUNCTION` to compute the derivative of the function. The form is `G(X)`, where

`X` – The point at which the derivative is to be computed.   (Input)
`G` – The computed value of the derivative at `X`.   (Output)

`G` must be declared `EXTERNAL` in the calling program.

XGUESS — An initial guess of the minimum point of `F`.   (Input)

ERRREL — The required relative accuracy in the final value of X. (Input)
This is the first stopping criterion. On a normal return, the solution X is in an interval that contains a local minimum and is less than or equal to MAX(1.0, ABS(X)) * ERRREL. When the given ERRREL is less than machine epsilon, SQRT(machine epsilon) is used as ERRREL.

GTOL — The derivative tolerance used to decide if the current point is a local minimum. (Input)
This is the second stopping criterion. X is returned as a solution when GX is less than or equal to GTOL. GTOL should be nonnegative, otherwise zero would be used.

MAXFN — Maximum number of function evaluations allowed. (Input)

A — A is the lower endpoint of the interval in which the minimum point of F is to be located. (Input)

B — B is the upper endpoint of the interval in which the minimum point of F is to be located. (Input)

X — The point at which a minimum value of F is found. (Output)

FX — The function value at point X. (Output)

GX — The derivative value at point X. (Output)

Comments

Informational errors

Type	Code	
3	1	The final value of X is at the lower bound. The minimum is probably beyond the bound.
3	2	The final value of X is at the upper bound. The minimum is probably beyond the bound.
4	3	The maximum number of function evaluations has been exceeded.

Algorithm

The routine UVMID uses a descent method with either the secant method or cubic interpolation to find a minimum point of a univariate function. It starts with an initial guess and two endpoints. If any of the three points is a local minimum point and has least function value, the routine terminates with a solution. Otherwise, the point with least function value will be used as the starting point.

From the starting point, say x_c, the function value $f_c = f(x_c)$, the derivative value $g_c = g(x_c)$, and a new point x_n defined by $x_n = x_c - g_c$ are computed. The function $f_n = f(x_n)$, and the derivative $g_n = g(x_n)$ are then evaluated. If either $f_n \geq f_c$ or g_n has the opposite sign of g_c, then there exists a minimum point between x_c and x_n; and an initial interval is obtained. Otherwise, since x_c is kept as the point that has lowest function value, an interchange between x_n and x_c is performed. The secant method is then used to get a new point

$$x_s = x_c - g_c(\frac{g_n - g_c}{x_n - x_c})$$

Let $x_n \leftarrow x_s$ and repeat this process until an interval containing a minimum is found or one of the convergence criteria is satisfied. The convergence criteria are as follows: Criterion 1:

$$|x_c - x_n| \le \varepsilon_c$$

Criterion 2:

$$|g_c| \le \varepsilon_g$$

where $\varepsilon_c = \max\{1.0, |x_c|\}\varepsilon$, ε is a relative error tolerance and ε_g is a gradient tolerance.

When convergence is not achieved, a cubic interpolation is performed to obtain a new point. Function and derivative are then evaluated at that point; and accordingly, a smaller interval that contains a minimum point is chosen. A safeguarded method is used to ensure that the interval reduces by at least a fraction of the previous interval. Another cubic interpolation is then performed, and this procedure is repeated until one of the stopping criteria is met.

Example

A minimum point of $e^x - 5x$ is found.

```
C                                  Declare variables
      INTEGER    MAXFN, NOUT
      REAL       A, B, ERRREL, F, FX, G, GTOL, GX, X, XGUESS
      EXTERNAL   F, G, UMACH, UVMID
C                                  Initialize variables
      XGUESS = 0.0
C                                 Set ERRREL to zero in order
C                                 to use SQRT(machine epsilon)
C                                 as relative error
      ERRREL = 0.0
      GTOL   = 0.0
      A      = -10.0
      B      = 10.0
      MAXFN  = 50
C
C                                 Find minimum for F = EXP(X) - 5X
      CALL UVMID (F, G, XGUESS, ERRREL, GTOL, MAXFN, A, B, X, FX, GX)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FX, GX
C
99999 FORMAT ('   The minimum is at ', 7X, F7.3, //, '   The function '
     &       , 'value is ', F7.3, //, '   The derivative is ', F7.3)
C
      END
```

```
C                                 Real function: F = EXP(X) - 5.0*X
      REAL FUNCTION F (X)
      REAL       X
C
      REAL       EXP
      INTRINSIC  EXP
C
      F = EXP(X) - 5.0E0*X
C
      RETURN
      END
C
      REAL FUNCTION G (X)
      REAL       X
C
      REAL       EXP
      INTRINSIC  EXP
C
      G = EXP(X) - 5.0E0
      RETURN
      END
```

Output

```
The minimum is at       1.609

The function value is  -3.047

The derivative is  -0.001
```

UVMGS/DUVMGS (Single/Double precision)

Find the minimum point of a nonsmooth function of a single variable.

Usage

```
CALL UVMGS (F, A, B, TOL, XMIN)
```

Arguments

F — User-supplied FUNCTION to compute the value of the function to be minimized. The form is F(X), where

X – The point at which the function is evaluated. (Input)
X should not be changed by F.
F – The computed function value at the point X. (Output)

F must be declared EXTERNAL in the calling program.

A — On input, A is the lower endpoint of the interval in which the minimum of F is to be located. On output, A is the lower endpoint of the interval in which the minimum of F is located. (Input/Output)

B — On input, B is the upper endpoint of the interval in which the minimum of F is to be located. On output, B is the upper endpoint of the interval in which the minimum of F is located. (Input/Output)

TOL — The allowable length of the final subinterval containing the minimum point. (Input)

XMIN — The approximate minimum point of the function `F` on the original interval (`A`, `B`). (Output)

Comments

1. Informational errors

Type	Code	
3	1	`TOL` is too small to be satisfied.
4	2	Due to rounding errors `F` does not appear to be unimodal.

2. On exit from `UVMGS` without any error messages, the following conditions hold: (`B`-`A`) ≤ `TOL`.
   `A` ≤ `XMIN` and `XMIN` ≤ `B`
   `F`(`XMIN`) ≤ `F`(`A`) and `F`(`XMIN`) ≤ `F`(`B`)

3. On exit from `UVMGS` with error code 2, the following conditions hold:
   `A` ≤ `XMIN` and `XMIN` ≤ `B`
   `F`(`XMIN`) ≥ `F`(`A`) and `F`(`XMIN`) ≥ `F`(`B`) (only one equality can hold).
   Further analysis of the function `F` is necessary in order to determine whether it is not unimodal in the mathematical sense or whether it appears to be not unimodal to the routine due to rounding errors in which case the `A`, `B`, and `XMIN` returned may be acceptable.

Algorithm

The routine `UVMGS` uses the *golden section search* technique to compute to the desired accuracy the independent variable value that minimizes a unimodal function of one independent variable, where a known finite interval contains the minimum.

Let τ = `TOL`. The number of iterations required to compute the minimizing value to accuracy τ is the greatest integer less than or equal to

$$\frac{\ln\left(\tau / (b-a)\right)}{\ln(1-c)}+1$$

where a and b define the interval and

$$c=\left(3-\sqrt{5}\right)/2$$

The first two test points are v_1 and v_2 that are defined as

$$v_1 = a + c(b - a), \text{ and } v_2 = b - c(b - a)$$

If $f(v_1) < f(v_2)$, then the minimizing value is in the interval (a, v_2). In this case, $b \leftarrow v_2$, $v_2 \leftarrow v_1$, and $v_1 \leftarrow a + c(b - a)$. If $f(v_1) \ge f(v_2)$, the minimizing value is in (v_1, b). In this case, $a \leftarrow v_1$, $v_1 \leftarrow v_2$, and $v_2 \leftarrow b - c(b - a)$.

The algorithm continues in an analogous manner where only one new test point is computed at each step. This process continues until the desired accuracy τ is achieved. XMIN is set to the point producing the minimum value for the current iteration.

Mathematically, the algorithm always produces the minimizing value to the desired accuracy; however, numerical problems may be encountered. If f is too flat in part of the region of interest, the function may appear to be constant to the computer in that region. Error code 2 indicates that this problem has occurred. The user may rectify the problem by relaxing the requirement on τ, modifying (scaling, etc.) the form of f or executing the program in a higher precision.

Example

A minimum point of $3x^2 - 2x + 4$ is found.

```
C                                 Specification of variables
      INTEGER    NOUT
      REAL       A, B, FCN, FMIN, TOL, XMIN
      EXTERNAL   FCN, UMACH, UVMGS
C                                 Initialize variables
      A   = 0.0E0
      B   = 5.0E0
      TOL = 1.0E-3
C                                 Minimize FCN
      CALL UVMGS (FCN, A, B, TOL, XMIN)
      FMIN = FCN(XMIN)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) XMIN, FMIN, A, B
99999 FORMAT ('   The minimum is at ', F5.3, //, '   The ',
     &       'function value is ', F5.3, //, '   The final ',
     &       'interval is (', F6.4, ',', F6.4, ')', /)
C
      END
C
C                                 REAL FUNCTION: F = 3*X**2 - 2*X + 4
      REAL FUNCTION FCN (X)
      REAL       X
C
      FCN = 3.0E0*X*X - 2.0E0*X + 4.0E0
C
      RETURN
      END
```

Output

```
The minimum is at 0.333

The function value is 3.667

The final interval is (0.3331,0.3340)
```

UMINF/DUMINF (Single/Double precision)

Minimize a function of N variables using a quasi-Newton method and a finite-difference gradient.

Usage

```
CALL UMINF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM,
            X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)
X – The point at which the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length N containing an initial guess of the computed solution. (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7.(Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

UMINF N(N + 8) units, or
DUMINF 2N(N + 8) units.

Workspace may be explicitly provided, if desired, by use of U2INF/DU2INF. The reference is

```
CALL U2INF (FCN, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X,FVALUE, WK)
```

The additional argument is

WK — Work vector of length N(N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain a BFGS approximation to the Hessian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for UMINF occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for UMINF occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UMINF, then set IPARAM(1) to zero and call the routine UMINF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UMINF:

   ```
   CALL U4INF (IPARAM, RPARAM)
   ```
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f(t)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in UMINF.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The i-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\max\left(\left|f(x)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The i-th component of the scaled step between two points x and y is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in UMINF.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}, \varepsilon_2 = \|s\|_2, s = \text{XSCALE}, \text{ and } t = \text{XGUESS}$$

RPARAM(7) = Size of initial trust region radius.
Default: Not used in UMINF.

If double precision is required, then DU4INF is called, and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine UMINF uses a quasi-Newton method to find the minimum of a function $f(x)$ of n variables. Only function values are required. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

Given a starting point x_c, the search direction is computed according to the formula

$$d = -B^{-1} g_c$$

where B is a positive definite approximation of the Hessian and g_c is the gradient evaluated at x_c. A line search is then used to find a new point

$$x_n = x_c + \lambda d, \qquad \lambda > 0$$

such that

$$f(x_n) \le f(x_c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality condition $\|g(x)\| = \varepsilon$ is checked where ε is a gradient tolerance.

When optimality is not achieved, B is updated according to the BFGS formula

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x_n - x_c$ and $y = g_n - g_c$. Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

Since a finite-difference method is used to estimate the gradient, for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, IMSL routine `UMING` (page 886) should be used instead.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N), XGUESS(N),
     &           XSCALE(N)
      EXTERNAL   ROSBRK, U4INF, UMACH, UMINF
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
C
C                                 Relax gradient tolerance stopping
C                                 criterion
      CALL U4INF (IPARAM, RPARAM)
      RPARAM(1) = 10.0E0*RPARAM(1)
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL UMINF (ROSBRK, N, XGUESS, XSCALE, FSCALE, IPARAM, RPARAM,
     &            X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
C
99999 FORMAT (’  The solution is ’, 6X, 2F8.3, //, ’  The function ’,
     &       ’value is ’, F8.3, //, ’  The number of iterations is ’,
     &       10X, I3, /, ’  The number of function evaluations is ’,
     &       I3, /, ’  The number of gradient evaluations is ’, I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
```

```
C
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is           15
The number of function evaluations is  40
The number of gradient evaluations is  19
```

UMING/DUMING (Single/Double precision)

Minimize a function of N variables using a quasi-Newton method and a user-supplied gradient.

Usage

```
CALL UMING (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where
N – Length of X. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where
N – Length of X and G. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length N containing the initial guess of the minimum. (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   UMING N * (N + 8) units, or
   DUMING 2 * N * (N + 8) units.

   Workspace may be explicitly provided, if desired, by use of U2ING/DU2ING. The reference is

```
CALL U2ING (FCN, GRAD, N, XGUESS, XSCALE, FSCALE,
            IPARAM, RPARAM, X, FVALUE, WK)
```

   The additional argument is

   WK — Work vector of length N * (N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain a BFGS approximation to the Hessian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for UMING occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for UMING occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UMING, then set IPARAM(1) to zero and call routine UMING (page 886). Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UMING:

```
CALL U4INF (IPARAM, RPARAM)
```

Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.

IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f(t)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in UMING.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The i-th component of the scaled gradient at
x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = `XSCALE`, and f_s = `FSCALE`.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM`(2) = Scaled step tolerance. (`STEPTL`)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM`(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.
Default: Not used in `UMING`.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: Not used in `UMING`.

If double precision is required, then `DU4INF` is called, and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine UMING uses a quasi-Newton method to find the minimum of a function $f(x)$ of n variables. Function values and first derivatives are required. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

Given a starting point x_c, the search direction is computed according to the formula

$$d = -B^{-1} g_c$$

where B is a positive definite approximation of the Hessian and g_c is the gradient evaluated at x_c. A line search is then used to find a new point

$$x_n = x_c + \lambda d, \qquad \lambda > 0$$

such that

$$f(x_n) \le f(x_c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality condition $\|g(x)\| = \varepsilon$ is checked where ε is a gradient tolerance.

When optimality is not achieved, B is updated according to the BFGS formula

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x_n - x_c$ and $y = g_n - g_c$. Another search direction is then computed to begin the next iteration. For more details, see Dennis and Schnabel (1983, Appendix A).

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XSCALE(N)
      EXTERNAL   ROSBRK, ROSGRD, UMACH, UMING
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
C
      IPARAM(1) = 0
C                                Minimize Rosenbrock function using
```

```
C                                  initial guesses of -1.2 and 1.0
      CALL UMING (ROSBRK, ROSGRD, N, XGUESS, XSCALE, FSCALE, IPARAM,
     &            RPARAM, X, F)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
C
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ',
     &       'value is ', F8.3, //, '  The number of iterations is ',
     &       10X, I3, /, '  The number of function evaluations is ',
     &       I3, /, '  The number of gradient evaluations is ', I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            18
The number of function evaluations is  31
The number of gradient evaluations is  22
```

UMIDH/DUMIDH (Single/Double precision)

Minimize a function of N variables using a modified Newton method and a finite-difference Hessian.

Usage

```
CALL UMIDH (FCN, GRAD, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X.
The usage is CALL GRAD (N, X, G), where
N – Length of X and G. (Input)
X – The point at which the gradient is evaluated. (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length N containing initial guess. (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   UMIDH N * (N + 9) units, or
   DUMIDH 2 * N * (N + 9) units.

   Workspace may be explicitly provided, if desired, by use of U2IDH/DU2IDH. The reference is

```
CALL U2IDH (FCN, GRAD, N, XGUESS, XSCALE, FSCALE,
            IPARAM, RPARAM, X, FVALUE, WK)
```

The additional argument is

WK — Work vector of length `N * (N + 9)`. `WK` contains the following information on output: The second `N` locations contain the last step taken. The third `N` locations contain the last Newton step. The fourth `N` locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current `X` value.

3. The first stopping criterion for `UMIDH` occurs when the norm of the gradient is less than the given gradient tolerance (`RPARAM`(1)). The second stopping criterion for `UMIDH` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UMIDH`, then set `IPARAM`(1) to zero and call routine `UMIDH`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UMIDH`:

```
CALL U4INF (IPARAM, RPARAM)
```
Set nondefault values for desired `IPARAM`, `RPARAM` elements.

Note that the call to `U4INF` will set `IPARAM` and `RPARAM` to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
`IPARAM`(1) = Initialization flag.

`IPARAM`(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter
Default: Not used in UMIDH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default:100

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at *x* is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1 / s_i\right)}{\max\left(\left|f(x)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)

The *i*-th component of the scaled step

between two points *x* and *y* is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1 / s_i\right)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in UMIDH.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i \right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: Based on initial scaled Cauchy step.

If double precision is required, then DU4INF is called, and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine UMIDH uses a modified Newton method to find the minimum of a function $f(x)$ of n variables. First derivatives must be provided by the user. The algorithm computes an optimal locally constrained step (Gay 1981) with a trust region restriction on the step. It handles the case that the Hessian is indefinite and provides a way to deal with negative curvature. For more details, see Dennis and Schnabel (1983, Appendix A) and Gay (1983).

Since a finite-difference method is used to estimate the Hessian for some single precision calculations, an inaccurate estimate of the Hessian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Hessian can be easily provided, IMSL routine UMIAH (page 896) should be used instead.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XSCALE(N)
      EXTERNAL   ROSBRK, ROSGRD, UMACH, UMIDH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
C
      IPARAM(1) = 0
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL UMIDH (ROSBRK, ROSGRD, N, XGUESS, XSCALE, FSCALE, IPARAM,
     &            RPARAM, X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
C
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ',
     &       'value is ', F8.3, //, '  The number of iterations is ',
     &       10X, I3, /, '  The number of function evaluations is ',
     &       I3, /, '  The number of gradient evaluations is ', I3, /,
     &       '  The number of Hessian evaluations is  ', I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            21
The number of function evaluations is  30
The number of gradient evaluations is  22
The number of Hessian evaluations is   21
```

UMIAH/DUMIAH (Single/Double precision)

Minimize a function of N variables using a modified Newton method and a user-supplied Hessian.

Usage

```
CALL UMIAH (FCN, GRAD, HESS, N, XGUESS, XSCALE, FSCALE,
            IPARAM, RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where
N – Length of X and G. (Input)
X – Vector of length N at which point the gradient is evaluated. (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

HESS — User-supplied SUBROUTINE to compute the Hessian at the point X. The usage is CALL HESS (N, X, H, LDH), where
N – Length of X. (Input)
X – Vector of length N at which point the Hessian is evaluated. (Input)
X should not be changed by HESS.
H – The Hessian evaluated at the point X. (Output)
LDH – Leading dimension of H exactly as specified in the dimension statement of the calling program. LDH must be equal to N in this routine. (Input)

HESS must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length N containing initial guess. (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

UMIAH N * (N + 9) units, or
DUMIAH 2 * N * (N + 9) units.

Workspace may be explicitly provided, if desired, by use of U2IAH/DU2IAH. The reference is

```
CALL U2IAH (FCN, GRAD, HESS, N, XGUESS, XSCALE,
            FSCALE, IPARAM, RPARAM, X, FVALUE, WK)
```

The additional argument is

WK — Work vector of length N * (N + 9). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for UMIAH occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for UMIAH occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UMIAH, then set IPARAM(1) to zero and call the routine UMIAH. Otherwise, if any nondefault

parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UMIAH:

CALL U4INF (IPARAM, RPARAM)
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter
Default: Not used in UMIAH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at *x* is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: max(10^{-10}, $\varepsilon^{2/3}$), max(10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in UMIAH.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called, and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine UMIAH uses a modified Newton method to find the minimum of a function $f(x)$ of n variables. First and second derivatives must be provided by the user. The algorithm computes an optimal locally constrained step (Gay 1981) with a trust region restriction on the step. This algorithm handles the case where the Hessian is indefinite and provides a way to deal with negative curvature. For more details, see Dennis and Schnabel (1983, Appendix A) and Gay (1983).

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized. Default values for parameters are used.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XSCALE(N)
```

```
      EXTERNAL   ROSBRK, ROSGRD, ROSHES, UMACH, UMIAH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
C
      IPARAM(1) = 0
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL UMIAH (ROSBRK, ROSGRD, ROSHES, N, XGUESS, XSCALE, FSCALE,
     &            IPARAM, RPARAM, X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
C
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ',
     &       'value is ', F8.3, //, '  The number of iterations is ',
     &       10X, I3, /, '  The number of function evaluations is ',
     &       I3, /, '  The number of gradient evaluations is ', I3, /,
     &       '  The number of Hessian evaluations is  ', I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
C
      SUBROUTINE ROSHES (N, X, H, LDH)
      INTEGER    N, LDH
      REAL       X(N), H(LDH,N)
C
      H(1,1) = -4.0E2*X(2) + 1.2E3*X(1)*X(1) + 2.0E0
      H(2,1) = -4.0E2*X(1)
      H(1,2) = H(2,1)
      H(2,2) = 2.0E2
C
      RETURN
      END
```

Output

```
The solution is          1.000   1.000

The function value is    0.000

The number of iterations is            21
The number of function evaluations is  31
The number of gradient evaluations is  22
The number of Hessian evaluations is   21
```

UMCGF/DUMCGF (Single/Double precision)

Minimize a function of `N` variables using a conjugate gradient algorithm and a finite-difference gradient.

Usage

```
CALL UMCGF (FCN, N, XGUESS, XSCALE, GRADTL, MAXFN, DFPRED,
            X, G, FVALUE)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length `N` containing the initial guess of the minimum. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)

GRADTL — Convergence criterion. (Input)
The calculation ends when the sum of squares of the components of `G` is less than `GRADTL`.

MAXFN — Maximum number of function evaluations. (Input)
If `MAXFN` is set to zero, then no restriction on the number of function evaluations is set.

DFPRED — A rough estimate of the expected reduction in the function. (Input)
`DFPRED` is used to determine the size of the initial change to `X`.

X — Vector of length `N` containing the computed solution. (Output)

G — Vector of length `N` containing the components of the gradient at the final parameter estimates. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   UMCGF 6 * N units, or
   DUMCGF 12 * N units.

   Workspace may be explicitly provided, if desired, by use of U2CGF/DU2CGF. The reference is

   ```
   CALL U2CGF (FCN, N, XGUESS, XSCALE, GRADTL, MAXFN,
               DFPRED, X, G, FVALUE, S, RSS, RSG,
               GINIT, XOPT, GOPT)
   ```

   The additional arguments are as follows:

   S — Vector of length N used for the search direction in each iteration.

   RSS — Vector of length N containing conjugacy information.

   RSG — Vector of length N containing conjugacy information.

   GINIT — Vector of length N containing the gradient values at the start of an iteration.

   XOPT — Vector of length N containing the parameter values that yield the least calculated value for FVALUE.

   GOPT — Vector of length N containing the gradient values that yield the least calculated value for FVALUE.

2. Informational errors

Type	Code	
4	1	The line search of an integration was abandoned. This error may be caused by an error in gradient.
4	2	The calculation cannot continue because the search is uphill.
4	3	The iteration was terminated because MAXFN was exceeded.
3	4	The calculation was terminated because two consecutive iterations failed to reduce the function.

3. Because of the close relation between the conjugate-gradient method and the method of steepest descent, it is very helpful to choose the scale of the variables in a way that balances the magnitudes of the components of a typical gradient vector. It can be particularly inefficient if a few components of the gradient are much larger than the rest.

4. If the value of the parameter GRADTL in the argument list of the routine is set to zero, then the subroutine will continue its calculation until it stops reducing the objective function. In this case, the usual behavior is that changes in the objective function become dominated by computer rounding errors before precision is lost in the gradient vector. Therefore, because the point of view has been taken that the user

requires the least possible value of the function, a value of the objective function that is small due to computer rounding errors can prevent further progress. Hence, the precision in the final values of the variables may be only about half the number of significant digits in the computer arithmetic, but the least value of FVALUE is usually found to be quite accurate.

Algorithm

The routine UMCGF uses a conjugate gradient method to find the minimum of a function $f(x)$ of n variables. Only function values are required.

The routine is based on the version of the conjugate gradient algorithm described in Powell (1977). The main advantage of the conjugate gradient technique is that it provides a fast rate of convergence without the storage of any matrices. Therefore, it is particularly suitable for unconstrained minimization calculations where the number of variables is so large that matrices of dimension n cannot be stored in the main memory of the computer. For smaller problems, however, a routine such as routine UMINF (page 881), is usually more efficient because each iteration makes use of additional information from previous iterations.

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine UMCGG (page 905) should be used instead.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
C                                  Declaration of variables
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    I, MAXFN, NOUT
      REAL       DFPRED, FVALUE, G(N), GRADTL, X(N), XGUESS(N),
     &           XS(N)
      EXTERNAL   ROSBRK, UMACH, UMCGF
C
      DATA XGUESS/-1.2E0, 1.0E0/, XS/2*1.0E0/
C
      DFPRED = 0.2
      GRADTL = 1.0E-6
      MAXFN  = 100
C                                  Minimize the Rosenbrock function
      CALL UMCGF (ROSBRK, N, XGUESS, XS, GRADTL, MAXFN, DFPRED, X, G,
     &            FVALUE)
```

```
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(I),I=1,N), FVALUE, (G(I),I=1,N)
99999 FORMAT ('  The solution is ', 2F8.3, //, '  The function ',
     &       'evaluated at the solution is ', F8.3, //, '  The ',
     &       'gradient is ', 2F8.3, /)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
      RETURN
      END
```

Output

```
The solution is    0.999   0.998

The function evaluated at the solution is    0.000

The gradient is   -0.001   0.000
```

UMCGG/DUMCGG (Single/Double precision)

Minimize a function of N variables using a conjugate gradient algorithm and a user-supplied gradient.

Usage

```
CALL UMCGG (FCN, GRAD, N, XGUESS, GRADTL, MAXFN, DFPRED, X,
            G, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X.   (Input)
X – The point at which the function is evaluated.   (Input)
X should not be changed by FCN.
F – The computed function value at the point X.   (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Length of X and G.   (Input)
X – The point at which the gradient is evaluated.   (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X.   (Output)

GRAD must be declared EXTERNAL in the calling program.

N — Dimension of the problem.   (Input)

XGUESS — Vector of length N containing the initial guess of the minimum.   (Input)

GRADTL — Convergence criterion.   (Input)
The calculation ends when the sum of squares of the components of G is less than GRADTL.

MAXFN — Maximum number of function evaluations.   (Input)

DFPRED — A rough estimate of the expected reduction in the function.   (Input)
DFPRED is used to determine the size of the initial change to X.

X — Vector of length N containing the computed solution.   (Output)

G — Vector of length N containing the components of the gradient at the final parameter estimates.   (Output)

FVALUE — Scalar containing the value of the function at the computed solution.   (Output)

Comments

1. Automatic workspace usage is

   UMCGG   6 * N units, or
   DUMCGG 12 * N units.

   Workspace may be explicitly provided, if desired, by use of U2CGG/DU2CGG. The reference is

   ```
   CALL U2CGG (FCN, GRAD, N, XGUESS, GRADTL, MAXFN,
               DFPRED, X, G, FVALUE, S, RSS, RSG,
               GINIT, XOPT, GOPT)
   ```

   The additional arguments are as follows:

   S — Vector of length N used for the search direction in each iteration.

   RSS — Vector of length N containing conjugacy information.

   RSG — Vector of length N containing conjugacy information.

   GINIT — Vector of length N containing the gradient values at the start on an iteration.

   XOPT — Vector of length N containing the parameter values which yield the least calculated value for FVALUE.

   GOPT — Vector of length N containing the gradient values which yield the least calculated value for FVALUE.

2. Informational errors

Type	Code	
4	1	The line search of an integration was abandoned. This error may be caused by an error in gradient.

4	2	The calculation cannot continue because the search is uphill.
4	3	The iteration was terminated because `MAXFN` was exceeded.
3	4	The calculation was terminated because two consecutive iterations failed to reduce the function.

3. The routine includes no thorough checks on the part of the user program that calculates the derivatives of the objective function. Therefore, because derivative calculation is a frequent source of error, the user should verify independently the correctness of the derivatives that are given to the routine.

4. Because of the close relation between the conjugate-gradient method and the method of steepest descent, it is very helpful to choose the scale of the variables in a way that balances the magnitudes of the components of a typical gradient vector. It can be particularly inefficient if a few components of the gradient are much larger than the rest.

5. If the value of the parameter `GRADTL` in the argument list of the routine is set to zero, then the subroutine will continue its calculation until it stops reducing the objective function. In this case, the usual behavior is that changes in the objective function become dominated by computer rounding errors before precision is lost in the gradient vector. Therefore, because the point of view has been taken that the user requires the least possible value of the function, a value of the objective function that is small due to computer rounding errors can prevent further progress. Hence, the precision in the final values of the variables may be only about half the number of significant digits in the computer arithmetic, but the least value of `FVALUE` is usually found to be quite accurate.

Algorithm

The routine `UMCGG` uses a conjugate gradient method to find the minimum of a function $f(x)$ of n variables. Function values and first derivatives are required.

The routine is based on the version of the conjugate gradient algorithm described in Powell (1977). The main advantage of the conjugate gradient technique is that it provides a fast rate of convergence without the storage of any matrices. Therefore, it is particularly suitable for unconstrained minimization calculations where the number of variables is so large that matrices of dimension n cannot be stored in the main memory of the computer. For smaller problems, however, a subroutine such as IMSL routine `UMING` (page 886), is usually more efficient because each iteration makes use of additional information from previous iterations.

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
C                                 Declaration of variables
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    I, MAXFN, NOUT
      REAL       DFPRED, FVALUE, G(N), GRADTL, X(N),
     &           XGUESS(N)
      EXTERNAL   ROSBRK, ROSGRD, UMACH, UMCGG
C
      DATA XGUESS/-1.2E0, 1.0E0/
C
      DFPRED = 0.2
      GRADTL = 1.0E-7
      MAXFN  = 100
C                                 Minimize the Rosenbrock function
      CALL UMCGG (ROSBRK, ROSGRD, N, XGUESS, GRADTL, MAXFN, DFPRED, X,
     &            G, FVALUE)
C                                 Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(I),I=1,N), FVALUE, (G(I),I=1,N)
99999 FORMAT ('  The solution is ', 2F8.3, //, '  The function ',
     &       'evaluated at the solution is ', F8.3, //, '  The ',
     &       'gradient is ', 2F8.3, /)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
```

Output

```
The solution is    1.000   1.000

The function evaluated at the solution is    0.000

The gradient is    0.000   0.000
```

UMPOL/DUMPOL (Single/Double precision)

Minimize a function of N variables using a direct search polytope algorithm.

Usage

```
CALL UMPOL (FCN, N, XGUESS, S, FTOL, MAXFCN, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Real vector of length N which contains an initial guess to the minimum. (Input)

S — On input, real scalar containing the length of each side of the initial simplex. (Input/Output)
If no reasonable information about S is known, S could be set to a number less than or equal to zero and UMPOL will generate the starting simplex from the initial guess with a random number generator. On output, the average distance from the vertices to the centroid that is taken to be the solution; see Comment 4.

FTOL — First convergence criterion. (Input)
The algorithm stops when a relative error in the function values is less than FTOL, i.e. when (F(worst) – F(best)) < FTOL * (1 + ABS(F(best))) where F(worst) and F(best) are the function values of the current worst and best points, respectively. Second convergence criterion. The algorithm stops when the standard deviation of the function values at the N + 1 current points is less than FTOL. If the subroutine terminates prematurely, try again with a smaller value for FTOL.

MAXFCN — On input, maximum allowed number of function evaluations. (Input/ Output)
On output, actual number of function evaluations needed.

X — Real vector of length N containing the best estimate of the minimum found. (Output)

FVALUE — Function value at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   UMPOL N**2 + 5 * N + 1 units, or
   DUMPOL 2 * N**2 + 10 * N + 2 units.

   Workspace may be explicitly provided, if desired, by use of U2POL/DU2POL. The reference is

   ```
   CALL U2POL (FCN, N, XGUESS, S, FTOL, MAXFCN, X,
               FVALUE, WK)
   ```

   The additional argument is

   WK — Real work vector of length N**2 + 5 * N + 1.

2. Informational error

Type	Code	
4	1	Maximum number of function evaluations exceeded.

3. Since UMPOL uses only function value information at each step to determine a new approximate minimum, it could be quite ineficient on smooth problems compared to other methods such as those implemented in routine UMINF that takes into account derivative information at each iteration. Hence, routine UMPOL should only be used as a last resort. Briefly, a set of N + 1 points in an N-dimensional space is called a simplex. The minimization process iterates by replacing the point with the largest function value by a new point with a smaller function value. The iteration continues until all the points cluster sufficiently close to a minimum.

4. The value returned in S is useful for assessing the flatness of the function near the computed minimum. The larger its value for a given value of FTOL, the flatter the function tends to be in the neighborhood of the returned point.

Algorithm

The routine UMPOL uses the polytope algorithm to find a minimum point of a function $f(x)$ of n variables. The polytope method is based on function comparison; no smoothness is assumed. It starts with $n + 1$ points $x_1, x_2, \ldots, x_{n+1}$. At each iteration, a new point is generated to replace the worst point x_j, which has the largest function value among these $n + 1$ points. The new point is constructed by the following formula:

$$x_k = c + \alpha(c - x_j)$$

where

$$c = \frac{1}{n} \sum_{i \neq j} x_i$$

and α (α > 0) is the *reflection coefficient*.

When x_k is a best point, that is $f(x_k) \leq f(x_i)$ for $i = 1, \ldots, n + 1$, an expansion point is computed $x_e = c + \beta(x_k - c)$ where $\beta(\beta > 1)$ is called the *expansion coefficient*. If the new point is a worst point, then the polytope would be contracted to get a better new point. If the contraction step is unsuccessful, the polytope is shrunk by moving the vertices halfway toward current best point. This procedure is repeated until one of the following stopping criteria is satisfied:

Criterion 1:

$$f_{best} - f_{worst} \leq \varepsilon_f (1. + |f_{best}|)$$

Criterion 2:

$$\sum_{i=1}^{n+1} (f_i - \frac{\sum_{j=1}^{n+1} f_j}{n+1})^2 \leq \varepsilon_f$$

where $f_i = f(x_i)$, $f_j = f(x_j)$, and ε_f is a given tolerance. For a complete description, see Nelder and Mead (1965) or Gill et al. (1981).

Example

The function

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

is minimized and the solution is printed.

```
C                                  Variable declarations
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    K, MAXFCN, NOUT
      REAL       FTOL, FVALUE, S, X(N), XGUESS(N)
      EXTERNAL   FCN, UMACH, UMPOL
C
C                                  Initializations
C                                  XGUESS = ( -1.2, 1.0)
C
      DATA XGUESS/-1.2, 1.0/
C
      FTOL   = 1.0E-10
      MAXFCN = 200
      S      = 1.0
C
      CALL UMPOL (FCN, N, XGUESS, S, FTOL, MAXFCN, X, FVALUE)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N), FVALUE
```

```
99999 FORMAT ('  The best estimate for the minimum value of the', /,
     &       '  function is X = (', 2(2X,F4.2), ')', /, '  with ',
     &       'function value FVALUE = ', E12.6)
C
      END
C                                  External function to be minimized
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 100.0*(X(1)*X(1)-X(2))**2 + (1.0-X(1))**2
      RETURN
      END
```

Output

```
The best estimate for the minimum value of the
function is X = (  1.00  1.00)
with function value FVALUE = 0.502496E-10
```

UNLSF/DUNLSF (Single/Double precision)

Solve a nonlinear least-squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

Usage

```
CALL UNLSF (FCN, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X, FVEC, FJAC, LDFJAC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function that defines the least-squares problem. The usage is `CALL FCN (M, N, X, F)`, where
- `M` – Length of `F`.   (Input)
- `N` – Length of `X`.   (Input)
- `X` – Vector of length `N` at which point the function is evaluated.   (Input) `X` should not be changed by `FCN`.
- `F` – Vector of length `M` containing the function values at `X`.   (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

M — Number of functions.   (Input)

N — Number of variables. `N` must be less than or equal to `M`.   (Input)

XGUESS — Vector of length `N` containing the initial guess.   (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables.   (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. By default, the values for `XSCALE` are set internally. See `IPARAM`(6) in Comment 4.

FSCALE — Vector of length M containing the diagonal scaling matrix for the functions. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution. (Output)

FVEC — Vector of length M containing the residuals at the approximate solution. (Output)

FJAC — M by N matrix containing a finite difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of FJAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   UNLSF 10 * N + 3 * M – 1 units, or
   DUNLSF 19 * N + 6 * M – 2 units.

   Workspace may be explicitly provided, if desired, by use of U2LSF/DU2LSF. The reference is

   ```
   CALL U2LSF (FCN, M, N, XGUESS, XSCALE, FSCALE,
               IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC,
               WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length 9 * N + 3 * M – 1. WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Gauss-Newton step. The fourth N locations contain an estimate of the gradient at the solution.

   IWK — Integer work vector of length N containing the permutations used in the QR factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.

4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3. The first stopping criterion for UNLSF occurs when the norm of the function is less than the absolute function tolerance (RPARAM(4)). The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance (RPARAM(1)). The third stopping criterion for UNLSF occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for UNLSF, then set IPARAM(1) to zero and call the routine UNLSF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling UNLSF:

   ```
   CALL U4LSF (IPARAM, RPARAM)
   ```
   Set nondefault values for desired IPARAM, RPARAM elements.

   Note that the call to U4LSF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

   IPARAM — Integer vector of length 6.

   IPARAM(1) = Initialization flag.

   IPARAM(2) = Number of good digits in the function.
   Default: Machine dependent.

   IPARAM(3) = Maximum number of iterations.
   Default: 100.

   IPARAM(4) = Maximum number of function evaluations.
   Default: 400.

   IPARAM(5) = Maximum number of Jacobian evaluations.
   Default: Not used in UNLSF.

   IPARAM(6) = Internal variable scaling flag.
   If IPARAM(6) = 1, then the values for XSCALE are set internally.
   Default: 1.

   RPARAM — Real vector of length 7.
   RPARAM(1) = Scaled gradient tolerance.
   The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\|F(x)\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = `XSCALE`, and f_s = `FSCALE`.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

`RPARAM(2)` = Scaled step tolerance. (`STEPTL`)
The i-th component of the scaled step between two points x and y is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = `XSCALE`.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

`RPARAM(3)` = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM(4)` = Absolute function tolerance.
Default: $\max(10^{-20}, \varepsilon^2)$, $\max(10^{-40}, \varepsilon^2)$ in double where ε is the machine precision.

`RPARAM(5)` = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM(6)` = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM(7)` = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DU4LSF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine `UNLSF` is based on the MINPACK routine `LMDIF` by Moré et al. (1980). It uses a modified Levenberg-Marquardt method to solve nonlinear least squares problems. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \geq n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in \mathbf{R}^n} \left\| F(x_c) + J(x_c)(x_n - x_c) \right\|_2$$

$$\text{subject to } \|x_n - x_c\|_2 \leq \delta_c$$

to get a new point x_n, which is computed as

$$x_n = x_c - \left(J(x_c)^T J(x_c) + \mu_c I \right)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \geq \|(J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c)\|_2$ and $\mu_c > 0$ otherwise. $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point x_c. This procedure is repeated until the stopping criteria are satisfied. For more details, see Levenberg (1944), Marquardt (1963), or Dennis and Schnabel (1983, Chapter 10).

Since a finite-difference method is used to estimate the Jacobian for some single precision calculations, an inaccurate estimate of the Jacobian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, routine `UNLSJ` (page 918) should be used instead.

Example

The nonlinear least squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved. `RPARAM`(4) is changed to a non-default value.

```
C                                  Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
C
      INTEGER    IPARAM(6), NOUT
      REAL       FJAC(LDFJAC,N), FSCALE(M), FVEC(M), RPARAM(7),
     &           X(N), XGUESS(N), XSCALE(N)
      EXTERNAL   ROSBCK, UMACH, UNLSF, U4LSF
C                                  Compute the least squares for the
C                                  Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/2*1.0E0/, FSCALE/2*1.0E0/
C
C                                  Relax the first stopping criterion by
C                                  calling U4LSF and scaling the
C                                  absolute function tolerance by 10.
      CALL U4LSF (IPARAM, RPARAM)
      RPARAM(4) = 10.0E0*RPARAM(4)
C
      CALL UNLSF (ROSBCK, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
     &            RPARAM, X, FVEC, FJAC, LDFJAC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
C
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ',
     &       'evaluated at the solution is ', /, 18X, 2F9.4, //,
     &       '  The number of iterations is ', 10X, I3, /, '  The ',
     &       'number of function evaluations is ', I3, /)
      END
C
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
C
      F(1) = 10.0E0*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
```

Output

```
The solution is    1.0000   1.0000

The function evaluated at the solution is
0.0000   0.0000

The number of iterations is           22
The number of function evaluations is  30
```

UNLSJ/DUNLSJ (Single/Double precision)

Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

Usage

```
CALL UNLSJ (FCN, JAC, M, N, XGUESS, XSCALE, FSCALE, IPARAM,
            RPARAM, X, FVEC, FJAC, LDFJAC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function which defines the least-squares problem. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – Vector of length `M` containing the function values at `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (M, N, X, FJAC, LDFJAC)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the Jacobian is evaluated. (Input)
`X` should not be changed by `JAC`.
`FJAC` – The computed `M` by `N` Jacobian at the point `X`. (Output)
`LDFJAC` – Leading dimension of `FJAC`. (Input)

`JAC` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

N — Number of variables. `N` must be less than or equal to `M`. (Input)

XGUESS — Vector of length `N` containing the initial guess. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. By default, the values for `XSCALE` are set internally. See `IPARAM`(6) in Comment 4.

FSCALE — Vector of length `M` containing the diagonal scaling matrix for the functions. (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution. (Output)

FVEC — Vector of length M containing the residuals at the approximate solution. (Output)

FJAC — M by N matrix containing a finite-difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of FJAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   UNLSJ 10 * N + 3 * M – 1 units, or
   DUNLSJ 19 * N + 6 * M – 2 units.

   Workspace may be explicitly provided, if desired, by use of U2LSJ/DU2LSJ. The reference is

   ```
   CALL U2LSJ (FCN, JAC, M, N, XGUESS, XSCALE, FSCALE,
               IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC,
               WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 9 * N + 3 * M – 1. WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Gauss-Newton step. The fourth N locations contain an estimate of the gradient at the solution.

   IWK — Work vector of length N containing the permutations used in the QR factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of Jacobian evaluations exceeded.

3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or `STEPTL` is too big.

3. The first stopping criterion for `UNLSJ` occurs when the norm of the function is less than the absolute function tolerance (`RPARAM`(4)). The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance (`RPARAM`(1)). The third stopping criterion for `UNLSJ` occurs when the scaled distance between the last two steps is less than the step tolerance (`RPARAM`(2)).

4. If the default parameters are desired for `UNLSJ`, then set `IPARAM`(1) to zero and call the routine `UNLSJ`. Otherwise, if any nondefault parameters are desired for `IPARAM` or `RPARAM`, then the following steps should be taken before calling `UNLSJ`:

   ```
   CALL U4LSF (IPARAM, RPARAM)
   ```
   Set nondefault values for desired `IPARAM`, `RPARAM` elements.

   Note that the call to `U4LSF` will set `IPARAM` and `RPARAM` to their default values, so only nondefault values need to be set above.

   The following is a list of the parameters and the default values:

   IPARAM — Integer vector of length 6.
   `IPARAM`(1) = Initialization flag.

   `IPARAM`(2) = Number of good digits in the function.
   Default: Machine dependent.

   `IPARAM`(3) = Maximum number of iterations.
   Default: 100.

   `IPARAM`(4) = Maximum number of function evaluations.
   Default: 400.

   `IPARAM`(5) = Maximum number of Jacobian evaluations.
   Default: 100.

   `IPARAM`(6) = Internal variable scaling flag.
   If `IPARAM`(6) = 1, then the values for `XSCALE` are set internally.
   Default: 1.

   RPARAM — Real vector of length 7.
   `RPARAM`(1) = Scaled gradient tolerance.
   The *i*-th component of the scaled gradient at `x` is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\left\|F(x)\right\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.

Default: max(10^{-10}, $\varepsilon^{2/3}$), max (10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.

Default: max (10^{-20}, ε^2), max(10^{-40}, ε^2) in double where ε is the machine precision.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then DU4LSF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine `UNLSJ` is based on the MINPACK routine `LMDER` by Moré et al. (1980). It uses a modified Levenberg-Marquardt method to solve nonlinear least squares problems. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

where $m \geq n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a current point, the algorithm uses the trust region approach:

$$\min_{x_n \in \mathbf{R}^n} \left\| F(x_c) + J(x_c)(x_n - x_c) \right\|_2$$

$$\text{subject to } \|x_n - x_c\|_2 \leq \delta_c$$

to get a new point x_n, which is computed as

$$x_n = x_c - \left(J(x_c)^T J(x_c) + \mu_c I \right)^{-1} J(x_c)^T F(x_c)$$

where $\mu_c = 0$ if $\delta_c \geq \|(J(x_c)^T J(x_c))^{-1} J(x_c)^T F(x_c)\|_2$ and $\mu_c > 0$ otherwise. $F(x_c)$ and $J(x_c)$ are the function values and the Jacobian evaluated at the current point x_c. This procedure is repeated until the stopping criteria are satisfied. For more details, see Levenberg (1944), Marquardt(1963), or Dennis and Schnabel (1983, Chapter 10).

Example

The nonlinear least-squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved; default values for parameters are used.

```
C                                 Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
C
      INTEGER    IPARAM(6), NOUT
      REAL       FJAC(LDFJAC,N), FSCALE(M), FVEC(M),
     &           RPARAM(7), X(N), XGUESS(N), XSCALE(N)
      EXTERNAL   ROSBCK, ROSJAC, UMACH, UNLSJ
C                                 Compute the least squares for the
C                                 Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/2*1.0E0/, FSCALE/2*1.0E0/
      IPARAM(1) = 0
C
      CALL UNLSJ (ROSBCK, ROSJAC, M, N, XGUESS, XSCALE, FSCALE,
     &            IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4), IPARAM(5)
C
99999 FORMAT (’  The solution is ’, 2F9.4, //, ’  The function ’,
     &       ’evaluated at the solution is ’, /, 18X, 2F9.4, //,
     &       ’  The number of iterations is ’, 10X, I3, /, ’  The ’,
     &       ’number of function evaluations is ’, I3, /, ’  The ’,
     &       ’number of Jacobian evaluations is ’, I3, /)
      END
C
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
C
      F(1) = 10.0E0*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
C
      SUBROUTINE ROSJAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
C
      FJAC(1,1) = -20.0E0*X(1)
      FJAC(2,1) = -1.0E0
      FJAC(1,2) = 10.0E0
      FJAC(2,2) = 0.0E0
      RETURN
      END
```

Output

```
The solution is    1.0000   1.0000

The function evaluated at the solution is
0.0000   0.0000

The number of iterations is            22
The number of function evaluations is  31
The number of Jacobian evaluations is  23
```

BCONF/DBCONF (Single/Double precision)

Minimize a function of N variables subject to bounds on the variables using a quasi-Newton method and a finite-difference gradient.

Usage

```
CALL BCONF (FCN, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
            FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`.   (Input)
`X` – Vector of length `N` at which point the function is evaluated.   (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`.   (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem.   (Input)

XGUESS — Vector of length `N` containing an initial guess of the computed solution.   (Input)
IBTYPE — Scalar indicating the types of bounds on variables.   (Input)

`IBTYPE`	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on variables.   (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XUB — Vector of length `N` containing the upper bounds on variables.   (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables.   (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling.   (Input)
`FSCALE` is used mainly in scaling the gradient. In the absence of other information, set `FSCALE` to 1.0.

IPARAM — Parameter vector of length 7.   (Input/Output)
Set `IPARAM`(1) to zero for default values of `IPARAM` and `RPARAM`. See Comment 4.

RPARAM — Parameter vector of length 7.   (Input/Output)
See Comment 4.

X — Vector of length `N` containing the computed solution.   (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   BCONF N * (2 * N + 8) + N units, or
   DBCONF 2 * N * (2 * N + 8) + N units.

   Workspace may be explicitly provided, if desired, by use of B2ONF/DB2ONF. The reference is

```
CALL B2ONF (FCN, N, XGUESS, IBTYPE, XLB, XUB,
            XSCALE, FSCALE, IPARAM, RPARAM, X,
            FVALUE, WK, IWK)
```

   The additional arguments are as follows:

   WK — Real work vector of length N * (2 * N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain a BFGS approximation to the Hessian at the solution.

   IWK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for BCONF occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for BCONF occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for BCONF, then set IPARAM(1) to zero and call the routine BCONF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCONF:

CALL U4INF (IPARAM, RPARAM)
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f(t)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in BCONF.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The i-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1/s_i\right)}{\max\left(\left|f(x)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1/s_i\right)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in BCONF.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: $1000 \max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine BCONF uses a quasi-Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -B^{-1} g^c$$

where B is a positive definite approximation of the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \qquad \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$||g(x_i)|| \le \varepsilon, l_i < x_i < u_i$$

$$g(x_i) < 0, \ x_i = u_i$$

$$g(x_i) > 0, x_i = l_i$$

are checked, where ε is a gradient tolerance. When optimality is not achieved, B is updated according to the BFGS formula:

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x^n - x^c$ and $y = g^n - g^c$. Another search direction is then computed to begin the next iteration.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the quasi-Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine `BCONG` (page 930) should be used instead.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0) and default values for parameters.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), ITP, L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N), XGUESS(N),
     &           XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   BCONF, ROSBRK, UMACH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/2*1.0E0/, FSCALE/1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C                                 All the bounds are provided
      ITP = 0
C                                 Default parameters are used
      IPARAM(1) = 0
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL BCONF (ROSBRK, N, XGUESS, ITP, XLB, XUB, XSCALE, FSCALE,
     &            IPARAM, RPARAM, X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
C
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ',
     &       'value is ', F8.3, //, '  The number of iterations is ',
     &       10X, I3, /, '  The number of function evaluations is ',
     &       I3, /, '  The number of gradient evaluations is ', I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            25
The number of function evaluations is  34
The number of gradient evaluations is  27
```

BCONG/DBCONG (Single/Double precision)

Minimize a function of N variables subject to bounds on the variables using a quasi-Newton method and a user-supplied gradient.

Usage

```
CALL BCONG (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
            FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Length of X and G. (Input)
X – Vector of length N at which point the gradient is evaluated. (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length N containing the initial guess of the minimum. (Input)
IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	Action
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XUB — Vector of length N containing the upper bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   BCONG N * (2 * N + 8) + N units, or
   DBCONG 2 * N * (2 * N + 8) + N units.

   Workspace may be explicitly provided, if desired, by use of B2ONG/DB2ONG. The reference is

   ```
   CALL B2ONG (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB,
               XSCALE, FSCALE, IPARAM, RPARAM, X,
               FVALUE, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length N * (2 * N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain a BFGS approximation to the Hessian at the solution.

   IWK — Integer work vector of length N.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.

4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for BCONG occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for BCONG occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for BCONG, then set IPARAM(1) to zero and call the routine BCONG. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCONG:

```
CALL U4INF (IPARAM, RPARAM)
```

Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
If IPARAM(6) = 0, the Hessian is initialized to the identity matrix; otherwise, it is initialized to a diagonal matrix containing

$$\max\left(\left|f(t)\right|, f_s\right) * s_i^2$$

on the diagonal where t = XGUESS, f_s = FSCALE, and s = XSCALE.
Default: 0.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: Not used in BCONG.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in BCONG.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine `BCONG` uses a quasi-Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } \; l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -B^{-1} g^c$$

where B is a positive definite approximation of the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \qquad \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$||g(x_i)|| \le \varepsilon, \; l_i < x_i < u_i$$

$$g(x_i) < 0, \; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

are checked, where ε is a gradient tolerance. When optimality is not achieved, B is updated according to the BFGS formula:

$$B \leftarrow B - \frac{Bss^T B}{s^T Bs} + \frac{yy^T}{y^T s}$$

where $s = x^n - x^c$ and $y = g^n - g^c$. Another search direction is then computed to begin the next iteration.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the quasi-Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and default values for parameters.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IPARAM(7), ITP, L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   BCONG, ROSBRK, ROSGRD, UMACH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/2*1.0E0/, FSCALE/1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C                                  All the bounds are provided
      ITP = 0
C                                  Default parameters are used
      IPARAM(1) = 0
C                                  Minimize Rosenbrock function using
C                                  initial guesses of -1.2 and 1.0
      CALL BCONG (ROSBRK, ROSGRD, N, XGUESS, ITP, XLB, XUB, XSCALE,
     &            FSCALE, IPARAM, RPARAM, X, F)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
C
99999 FORMAT ('  The solution is ', 6X, 2F8.3, //, '  The function ',
     &       'value is ', F8.3, //, '  The number of iterations is ',
     &       10X, I3, /, '  The number of function evaluations is ',
     &       I3, /, '  The number of gradient evaluations is ', I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is           23
The number of function evaluations is  32
The number of gradient evaluations is  24
```

BCODH/DBCODH (Single/Double precision)

Minimize a function of `N` variables subject to bounds on the variables using a modified Newton method and a finite-difference Hessian.

Usage

```
CALL BCODH (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
            FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)
`X` – Vector of length `N` at which point the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.
`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XGUESS — Vector of length `N` containing the initial guess of the minimum. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

`IBTYPE`	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on the variables. (Input)

XUB — Vector of length N containing the upper bounds on the variables. (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   BCODH N * (N + 8) + N units, or
   DBCODH 2 * N * (N + 8) + N units.

   Workspace may be explicitly provided, if desired, by use of B2ODH/DB2ODH. The reference is

   ```
   CALL B2ODH (FCN, GRAD, N, XGUESS, IBTYPE, XLB, XUB,
               XSCALE, FSCALE, IPARAM, RPARAM, X,
               FVALUE, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Real work vector of length N * (N + 8). WK contains the the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

   IWK — Integer work vector of length N.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
4	7	Maximum number of Hessian evaluations exceeded.

3. The first stopping criterion for BCODH occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for BCODH occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for BCODH, then set IPARAM(1) to zero and call the routine BCODH. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM; then the following steps should be taken before calling BCODH:

```
CALL U4INF (IPARAM, RPARAM)
```

Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
Default: Not used in BCODH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max(|x_i|, 1/s_i)}{\max(|f(x)|, f_s)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max(|x_i|, 1/s_i)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in BCODH.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine `BCODH` uses a modified Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -H^{-1} g^c$$

where H is the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \qquad \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \le \varepsilon, l_i < x_i < u_i$$

$$g(x_i) < 0, x_i = u_i$$

$$g(x_i) > 0, x_i = l_i$$

are checked where ε is a gradient tolerance. When optimality is not achieved, another search direction is computed to begin the next iteration. This process is repeated until the optimality criterion is met.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the modified Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the Hessian for some single precision calculations, an inaccurate estimate of the Hessian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Hessian can be easily provided, routine BCOAH (page 942) should be used instead.

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and default values for parameters.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IP, IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   BCODH, ROSBRK, ROSGRD, UMACH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C
      IPARAM(1) = 0
      IP        = 0
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL BCODH (ROSBRK, ROSGRD, N, XGUESS, IP, XLB, XUB, XSCALE,
     &            FSCALE, IPARAM, RPARAM, X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5)
C
99999 FORMAT (’  The solution is ’, 6X, 2F8.3, //, ’  The function ’,
     &       ’value is ’, F8.3, //, ’  The number of iterations is ’,
     &       10X, I3, /, ’  The number of function evaluations is ’,
     &       I3, /, ’  The number of gradient evaluations is ’, I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
C
      RETURN
      END
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
```

```
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            17
The number of function evaluations is  26
The number of gradient evaluations is  18
```

BCOAH/DBCOAH (Single/Double precision)

Minimize a function of N variables subject to bounds on the variables using a modified Newton method and a user-supplied Hessian.

Usage

```
CALL BCOAH (FCN, GRAD, HESS, N, XGUESS, IBTYPE, XLB, XUB,
            XSCALE, FSCALE, IPARAM, RPARAM, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Length of X and G. (Input)
X – Vector of length N at which point the gradient is evaluated. (Input)
X should not be changed by GRAD.
G – The gradient evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

HESS — User-supplied SUBROUTINE to compute the Hessian at the point X. The usage is CALL HESS (N, X, H, LDH), where

N – Length of X. (Input)
X – Vector of length N at which point the Hessian is evaluated. (Input)
X should not be changed by HESS.
H – The Hessian evaluated at the point X. (Output)
LDH – Leading dimension of H exactly as specified in the dimension statement of the calling program. (Input)

HESS must be declared EXTERNAL in the calling program.

N — Dimension of the problem.   (Input)

XGUESS — Vector of length N containing the initial guess.   (Input)

IBTYPE — Scalar indicating the types of bounds on variables.   (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on the variables.   (Input)

XUB — Vector of length N containing the upper bounds on the variables.
(Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables.   (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. In the absence of other information, set all entries to 1.0.

FSCALE — Scalar containing the function scaling.   (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set FSCALE to 1.0.

IPARAM — Parameter vector of length 7.   (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7.   (Input/Output)
See Comment 4.

X — Vector of length N containing the computed solution.   (Output)

FVALUE — Scalar containing the value of the function at the computed solution.
(Output)

Comments

1. Automatic workspace usage is

   BCOAH N * (N + 8) + N units, or
   DBCOAH 2 * N * (N + 8) + N units.

   Workspace may be explicitly provided, if desired, by use of B2OAH/DB2OAH. The reference is

```
CALL B2OAH (FCN, GRAD, HESS, N, XGUESS, IBTYPE, XLB,
            XUB, XSCALE, FSCALE, IPARAM, RPARAM, X,
            FVALUE, WK, IWK)
```

The additional arguments are as follows:

WK — Work vector of length N * (N + 8). WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Newton step. The fourth N locations contain an estimate of the gradient at the solution. The final N^2 locations contain the Hessian at the approximate solution.

IWK — Work vector of length N.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
4	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
4	5	Maximum number of gradient evaluations exceeded.
4	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.
4	7	Maximum number of Hessian evaluations exceeded.
3	8	The last global step failed to locate a lower point than the current X value.

3. The first stopping criterion for BCOAH occurs when the norm of the gradient is less than the given gradient tolerance (RPARAM(1)). The second stopping criterion for BCOAH occurs when the scaled distance between the last two steps is less than the step tolerance (RPARAM(2)).

4. If the default parameters are desired for BCOAH, then set IPARAM(1) to zero and call the routine BCOAH. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCOAH:

```
CALL U4INF (IPARAM, RPARAM)
```
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4INF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 7.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of gradient evaluations.
Default: 400.

IPARAM(6) = Hessian initialization parameter.
Default: Not used in BCOAH.

IPARAM(7) = Maximum number of Hessian evaluations.
Default: 100.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{\left|g_i\right| * \max\left(\left|x_i\right|, 1 / s_i\right)}{\max\left(\left|f(x)\right|, f_s\right)}$$

where $g = \nabla f(x)$, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step between two points *x* and *y* is computed as

$$\frac{\left|x_i - y_i\right|}{\max\left(\left|x_i\right|, 1 / s_i\right)}$$

where s = XSCALE.
Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: max(10^{-10}, $\varepsilon^{2/3}$), max (10^{-20}, $\varepsilon^{2/3}$) in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: Not used in BCOAH.

RPARAM(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 max(ε_1, ε_2) where

$$\varepsilon_1 \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \| s \|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is required, then DU4INF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine BCOAH uses a modified Newton method and an active set strategy to solve minimization problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to } l \le x \le u$$

From a given starting point x^c, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -H^{-1} g^c$$

where H is the Hessian and g^c is the gradient evaluated at x^c; both are computed with respect to the free variables. The search direction for the variables in IA is set to zero. A line search is used to find a new point x^n ,

$$x^n = x^c + \lambda d, \qquad \lambda \in (0, 1]$$

such that

$$f(x^n) \le f(x^c) + \alpha g^T d, \qquad \alpha \in (0, 0.5)$$

Finally, the optimality conditions

$$\|g(x_i)\| \le \varepsilon,\ l_i < x_i < u_i$$

$$g(x_i) < 0,\ x_i = u_i$$

$$g(x_i) > 0,\ x_i = l_i$$

are checked where ε is a gradient tolerance. When optimality is not achieved, another search direction is computed to begin the next iteration. This process is repeated until the optimality criterion is met.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more details on the modified Newton method and line search, see Dennis and Schnabel (1983). For more detailed information on active set strategy, see Gill and Murray (1976).

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and default values for parameters.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IP, IPARAM(7), L, NOUT
      REAL       F, FSCALE, RPARAM(7), X(N),
     &           XGUESS(N), XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   BCOAH, ROSBRK, ROSGRD, ROSHES, UMACH
C
      DATA XGUESS/-1.2E0, 1.0E0/, XSCALE/1.0E0, 1.0E0/, FSCALE/1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C
      IPARAM(1) = 0
      IP        = 0
C                                 Minimize Rosenbrock function using
C                                 initial guesses of -1.2 and 1.0
      CALL BCOAH (ROSBRK, ROSGRD, ROSHES, N, XGUESS, IP, XLB, XUB,
     &            XSCALE, FSCALE, IPARAM, RPARAM, X, F)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, F, (IPARAM(L),L=3,5), IPARAM(7)
C
99999 FORMAT (’  The solution is ’, 6X, 2F8.3, //, ’  The function ’,
     &       ’value is ’, F8.3, //, ’  The number of iterations is ’,
     &       10X, I3, /, ’  The number of function evaluations is ’,
     &       I3, /, ’  The number of gradient evaluations is ’, I3, /,
     &       ’  The number of Hessian evaluations is  ’, I3)
C
      END
C
      SUBROUTINE ROSBRK (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 1.0E2*(X(2)-X(1)*X(1))**2 + (1.0E0-X(1))**2
```

```
C
      RETURN
      END
C
      SUBROUTINE ROSGRD (N, X, G)
      INTEGER    N
      REAL       X(N), G(N)
C
      G(1) = -4.0E2*(X(2)-X(1)*X(1))*X(1) - 2.0E0*(1.0E0-X(1))
      G(2) = 2.0E2*(X(2)-X(1)*X(1))
C
      RETURN
      END
C
      SUBROUTINE ROSHES (N, X, H, LDH)
      INTEGER    N, LDH
      REAL       X(N), H(LDH,N)
C
      H(1,1) = -4.0E2*X(2) + 1.2E3*X(1)*X(1) + 2.0E0
      H(2,1) = -4.0E2*X(1)
      H(1,2) = H(2,1)
      H(2,2) = 2.0E2
C
      RETURN
      END
```

Output

```
The solution is          0.500   0.250

The function value is    0.250

The number of iterations is            18
The number of function evaluations is  29
The number of gradient evaluations is  19
The number of Hessian evaluations is   18
```

BCPOL/DBCPOL (Single/Double precision)

Minimize a function of N variables subject to bounds on the variables using a direct search complex algorithm.

Usage

```
CALL BCPOL (FCN, N, XGUESS, IBTYPE, XLB, XUB, FTOL, MAXFCN,
            X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Length of X.   (Input)
X – Vector of length N at which point the function is evaluated.   (Input)
X should not be changed by FCN.
F – The computed function value at the point X.   (Output)

FCN must be declared EXTERNAL in the calling program.

N — The number of variables. (Input)

XGUESS — Real vector of length N that contains an initial guess to the minimum. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on the first, variable. All other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on the variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XUB — Vector of length N containing the upper bounds on the variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

FTOL — First convergence criterion. (Input)
The algorithm stops when a relative error in the function values is less than FTOL, i.e. when (F(worst) – F(best)) < FTOL * (1 + ABS(F(best))) where F(worst) and F(best) are the function values of the current worst and best point, respectively. Second convergence criterion. The algorithm stops when the standard deviation of the function values at the 2 * N current points is less than FTOL. If the subroutine terminates prematurely, try again with a smaller value FTOL.

MAXFCN — On input, maximum allowed number of function evaluations. (Input/ Output)
On output, actual number of function evaluations needed.

X — Real vector of length N containing the best estimate of the minimum found. (Output)

FVALUE — Function value at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   BCPOL 2 * N**2 + 5 * N units, or
   DBCPOL 4 * N**2 + 10 * N units.

   Workspace may be explicitly provided, if desired, by use of B2POL/DB2POL. The reference is

   ```
   CALL B2POL (FCN, N, XGUESS, IBTYPE, XLB, XUB, FTOL,
               MAXFCN, X, FVALUE, WK)
   ```

   The additional argument is

   WK — Real work vector of length 2 * N**2 + 5 * N

2. Informational error

Type	Code	
3	1	The maximum number of function evaluations is exceeded.

3. Since BCPOL uses only function-value information at each step to determine a new approximate minimum, it could be quite inefficient on smooth problems compared to other methods such as those implemented in routine BCONF (page 923), which takes into account derivative information at each iteration. Hence, routine BCPOL should only be used as a last resort. Briefly, a set of 2 * N points in an N-dimensional space is called a complex. The minimization process iterates by replacing the point with the largest function value by a new point with a smaller function value. The iteration continues until all the points cluster sufficiently close to a minimum.

Algorithm

The routine BCPOL uses the complex method to find a minimum point of a function of n variables. The method is based on function comparison; no smoothness is assumed. It starts with $2n$ points $x_1, x_2, \ldots, x_{2n}$. At each iteration, a new point is generated to replace the worst point $x_{j,}$ which has the largest function value among these $2n$ points. The new point is constructed by the following formula:

$$x_k = c + \alpha(c - x_j)$$

where

$$c = \frac{1}{2n-1} \sum_{i \neq j} x_i$$

and α ($\alpha > 0$) is the *reflection coefficient*.

When x_k is a best point, that is, when $f(x_k) \le f(x_i)$ for $i = 1, \ldots, 2n$, an expansion point is computed $x_e = c + \beta(x_k - c)$, where $\beta(\beta > 1)$ is called the *expansion coefficient*. If the new point is a worst point, then the complex would be contracted to get a better new point. If the contraction step is unsuccessful, the complex is shrunk by moving the vertices halfway toward the current best point. Whenever the new point generated is beyond the bound, it will be set to the bound. This procedure is repeated until one of the following stopping criteria is satisfied:

Criterion 1:

$$f_{best} - f_{worst} \le \epsilon_f (1. + |f_{best}|)$$

Criterion 2:

$$\sum_{i=1}^{2n} (f_i - \frac{\sum_{j=1}^{2n} f_j}{2n})^2 \le \varepsilon_f$$

where $f_i = f(x_i)$, $f_j = f(x_j)$, and ε_f is a given tolerance. For a complete description, see Nelder and Mead (1965) or Gill et al. (1981).

Example

The problem

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$$
$$\text{subject to } -2 \le x_1 \le 0.5$$
$$-1 \le x_2 \le 2$$

is solved with an initial guess (–1.2, 1.0), and the solution is printed.

```
C                                 Variable declarations
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IBTYPE, K, MAXFCN, NOUT
      REAL       FTOL, FVALUE, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   BCPOL, FCN, UMACH
C
C                                 Initializations
C                                 XGUESS = (-1.2,  1.0)
C                                 XLB    = (-2.0, -1.0)
C                                 XUB    = ( 0.5,  2.0)
      DATA  XGUESS/-1.2, 1.0/, XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C
      FTOL   = 1.0E-5
      IBTYPE = 0
      MAXFCN = 300
C
      CALL BCPOL (FCN, N, XGUESS, IBTYPE, XLB, XUB, FTOL, MAXFCN, X,
     &            FVALUE)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (X(K),K=1,N), FVALUE
99999 FORMAT (’  The best estimate for the minimum value of the’, /,
     &       ’  function is X = (’, 2(2X,F4.2), ’)’, /, ’  with ’,
     &       ’function value FVALUE = ’, E12.6)
C
      END
C                                 External function to be minimized
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = 100.0*(X(2)-X(1)*X(1))**2 + (1.0-X(1))**2
      RETURN
      END
```

Output

```
The best estimate for the minimum value of the
function is X = (  0.50  0.25)
with function value FVALUE = 0.250002E+00
```

BCLSF/DBCLSF (Single/Double precision)

Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

Usage

```
CALL BCLSF (FCN, M, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
            FSCALE, IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

N — Number of variables. (Input)
`N` must be less than or equal to `M`.

XGUESS — Vector of length `N` containing the initial guess. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

`IBTYPE`	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length `N` containing the lower bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XUB — Vector of length `N` containing the upper bounds on variables. (Input, if `IBTYPE` = 0; output, if `IBTYPE` = 1 or 2; input/output, if `IBTYPE` = 3)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
`XSCALE` is used mainly in scaling the gradient and the distance between two

points. By default, the values for XSCALE are set internally. See IPARAM(6) in Comment 4.

FSCALE — Vector of length M containing the diagonal scaling matrix for the functions. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution. (Output)

FVEC — Vector of length M containing the residuals at the approximate solution. (Output)

FJAC — M by N matrix containing a finite difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of FJAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   BCLSF 13 * N + 3 * M – 1 units, or
   DBCLSF 24 * N + 6 * M – 2 units.

   Workspace may be explicitly provided, if desired, by use of B2LSF/DB2LSF. The reference is

   ```
   CALL B2LSF (FCN, M, N, XGUESS, IBTYPE, XLB, XUB,
               XSCALE, FSCALE, IPARAM, RPARAM, X, FVEC,
               FJAC, LDFJAC, WK, IWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length 11 * N + 3 * M – 1. WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Gauss-Newton step. The fourth N locations contain an estimate of the gradient at the solution.

   IWK — Work vector of length 2 * N containing the permutations used in the QR factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3. The first stopping criterion for BCLSF occurs when the norm of the function is less than the absolute function tolerance. The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance. The third stopping criterion for BCLSF occurs when the scaled distance between the last two steps is less than the step tolerance.

4. If the default parameters are desired for BCLSF, then set IPARAM(1) to zero and call the routine BCLSF. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCLSF:

```
CALL U4LSF (IPARAM, RPARAM)
```
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4LSF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of Jacobian evaluations.
Default: 100.

IPARAM(6) = Internal variable scaling flag.
If IPARAM(6) = 1, then the values for XSCALE are set internally.
Default: 1.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max\left(|x_i|, 1/s_i\right)}{\|F(x)\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The i-th component of the scaled step between two points x and y is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where s = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

RPARAM(4) = Absolute function tolerance.
Default: $\max(10^{-20}, \varepsilon^2)$, $\max(10^{-40}, \varepsilon^2)$ in double where ε is the machine precision.

RPARAM(5) = False convergence tolerance.
Default: 100 ε where ε is the machine precision.

RPARAM(6) = Maximum allowable step size.
Default: 1000 $\max(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum_{i=1}^{n} \left(s_i t_i\right)^2}$$

$\varepsilon_2 = \|s\|_2$, s = XSCALE, and t = XGUESS.

RPARAM(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then DU4LSF is called and RPARAM is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to “Error Handling” in the Introduction.

Algorithm

The routine BCLSF uses a modified Levenberg-Marquardt method and an active set strategy to solve nonlinear least squares problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

$$\text{subject to } l \le x \le u$$

where $m \ge n$, $F : \mathbf{R}^n \to \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a given starting point, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -\,(J^T J + \mu I)^{-1}\, J^T F$$

where μ is the Levenberg-Marquardt parameter, $F = F\,(x)$, and J is the Jacobian with respect to the free variables. The search direction for the variables in IA is set to zero. The trust region approach discussed by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are checked. The conditions are

$$||g(x_i)|| \le \varepsilon,\ l_i < x_i < u_i$$

$$g(x_i) < 0,\ \ x_i = u_i$$

$$g(x_i) > 0,\ x_i = l_i$$

where ε is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more detail on the Levenberg-Marquardt method, see Levenberg (1944), or Marquardt (1963). For more detailed information on active set strategy, see Gill and Murray (1976).

Since a finite-difference method is used to estimate the Jacobian for some single precision calculations, an inaccurate estimate of the Jacobian may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact Jacobian can be easily provided, routine BCLSJ (page 958) should be used instead.

Example

The nonlinear least squares problem

$$\min_{x \in \mathbf{R}^2} \frac{1}{2} \sum_{i=1}^{2} f_i(x)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved with an initial guess (–1.2, 1.0) and default values for parameters.

```
C                                  Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
C
      INTEGER    IPARAM(7), ITP, NOUT
      REAL       FJAC(LDFJAC,N), FSCALE(M), FVEC(M),
     &           RPARAM(7), X(N), XGUESS(N), XLB(N), XS(N), XUB(N)
      EXTERNAL   BCLSF, ROSBCK, UMACH
C                                  Compute the least squares for the
C                                  Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/, XS/2*1.0E0/, FSCALE/2*1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C                                  All the bounds are provided
      ITP = 0
C                                  Default parameters are used
      IPARAM(1) = 0
C
      CALL BCLSF (ROSBCK, M, N, XGUESS, ITP, XLB, XUB, XS, FSCALE,
     &            IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
C
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ',
     &       'evaluated at the solution is ', /, 18X, 2F9.4, //,
     &       '  The number of iterations is ', 10X, I3, /, '  The ',
     &       'number of function evaluations is ', I3, /)
      END
C
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
C
      F(1) = 1.0E1*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
```

Output

```
The solution is    0.5000   0.2500

The function evaluated at the solution is
0.0000   0.5000

The number of iterations is            15
The number of function evaluations is  20
```

BCLSJ/DBCLSJ (Single/Double precision)

Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

Usage

```
CALL BCLSJ (FCN, JAC, M, N, XGUESS, IBTYPE, XLB, XUB,
            XSCALE, FSCALE, IPARAM, RPARAM, X, FVEC, FJAC,
            LDFJAC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (M, N, X, FJAC, LDFJAC)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`FJAC` – The computed `M` by `N` Jacobian at the point `X`. (Output)
`LDFJAC` – Leading dimension of `FJAC`. (Input)

`JAC` must be declared `EXTERNAL` in the calling program.

M — Number of functions. (Input)

N — Number of variables. (Input)
`N` must be less than or equal to `M`.

XGUESS — Vector of length `N` containing the initial guess. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

`IBTYPE`	**Action**
0	User will supply all the bounds.

1 All variables are nonnegative.
2 All variables are nonpositive.
3 User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XUB — Vector of length N containing the upper bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
XSCALE is used mainly in scaling the gradient and the distance between two points. By default, the values for XSCALE are set internally. See IPARAM(6) in Comment 4.

FSCALE — Vector of length M containing the diagonal scaling matrix for the functions. (Input)
FSCALE is used mainly in scaling the gradient. In the absence of other information, set all entries to 1.0.

IPARAM — Parameter vector of length 6. (Input/Output)
Set IPARAM(1) to zero for default values of IPARAM and RPARAM. See Comment 4.

RPARAM — Parameter vector of length 7. (Input/Output)
See Comment 4.

X — Vector of length N containing the approximate solution. (Output)

FVEC — Vector of length M containing the residuals at the approximate solution. (Output)

FJAC — M by N matrix containing a finite difference approximate Jacobian at the approximate solution. (Output)

LDFJAC — Leading dimension of FJAC exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

BCLSJ 13 * N + 3 * M – 1 units, or
DBCLSJ 24 * N + 6 * M – 2 units.

Workspace may be explicitly provided, if desired, by use of B2LSJ/DB2LSJ. The reference is

```
CALL B2LSJ (FCN, JAC, M, N, XGUESS, IBTYPE, XLB,
            XUB, XSCALE, FSCALE, IPARAM, RPARAM, X,
            FVEC, FJAC, LDFJAC, WK, IWK)
```

The additional arguments are as follows:

WK — Work vector of length 11 * N + 3 * M – 1. WK contains the following information on output: The second N locations contain the last step taken. The third N locations contain the last Gauss-Newton step. The fourth N locations contain an estimate of the gradient at the solution.

IWK — Work vector of length 2 * N containing the permutations used in the QR factorization of the Jacobian at the solution.

2. Informational errors

Type	Code	
3	1	Both the actual and predicted relative reductions in the function are less than or equal to the relative function convergence tolerance.
3	2	The iterates appear to be converging to a noncritical point.
4	3	Maximum number of iterations exceeded.
4	4	Maximum number of function evaluations exceeded.
3	6	Five consecutive steps have been taken with the maximum step length.
4	5	Maximum number of Jacobian evaluations exceeded.
2	7	Scaled step tolerance satisfied; the current point may be an approximate local solution, or the algorithm is making very slow progress and is not near a solution, or STEPTL is too big.

3. The first stopping criterion for BCLSJ occurs when the norm of the function is less than the absolute function tolerance. The second stopping criterion occurs when the norm of the scaled gradient is less than the given gradient tolerance. The third stopping criterion for BCLSJ occurs when the scaled distance between the last two steps is less than the step tolerance.

4. If the default parameters are desired for BCLSJ, then set IPARAM(1) to zero and call the routine BCLSJ. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling BCLSJ:

```
CALL U4LSF (IPARAM, RPARAM)
```
Set nondefault values for desired IPARAM, RPARAM elements.

Note that the call to U4LSF will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

The following is a list of the parameters and the default values:

IPARAM — Integer vector of length 6.
IPARAM(1) = Initialization flag.

IPARAM(2) = Number of good digits in the function.
Default: Machine dependent.

IPARAM(3) = Maximum number of iterations.
Default: 100.

IPARAM(4) = Maximum number of function evaluations.
Default: 400.

IPARAM(5) = Maximum number of Jacobian evaluations.
Default: 100.

IPARAM(6) = Internal variable scaling flag.

If IPARAM(6) = 1, then the values for XSCALE are set internally.
Default: 1.

RPARAM — Real vector of length 7.
RPARAM(1) = Scaled gradient tolerance.
The *i*-th component of the scaled gradient at x is calculated as

$$\frac{|g_i| * \max\left(|x_i|, 1/s_i\right)}{\left\|F(x)\right\|_2^2}$$

where

$$g_i = \left(J(x)^T F(x)\right)_i * \left(f_s\right)_i^2$$

$J(x)$ is the Jacobian, s = XSCALE, and f_s = FSCALE.
Default:

$$\sqrt{\varepsilon}, \sqrt[3]{\varepsilon}$$

in double where ε is the machine precision.

RPARAM(2) = Scaled step tolerance. (STEPTL)
The *i*-th component of the scaled step
between two points *x* and *y* is computed as

$$\frac{|x_i - y_i|}{\max\left(|x_i|, 1/s_i\right)}$$

where *s* = XSCALE.

Default: $\varepsilon^{2/3}$ where ε is the machine precision.

RPARAM(3) = Relative function tolerance.
Default: $\max(10^{-10}, \varepsilon^{2/3})$, $\max(10^{-20}, \varepsilon^{2/3})$ in double where ε is the machine precision.

`RPARAM`(4) = Absolute function tolerance.

Default: max $(10^{-20}, \varepsilon^2)$, max$(10^{-40}, \varepsilon^2)$ in double where ε is the machine precision.

`RPARAM`(5) = False convergence tolerance.
Default: 100ε where ε is the machine precision.

`RPARAM`(6) = Maximum allowable step size.
Default: 1000 max$(\varepsilon_1, \varepsilon_2)$ where

$$\varepsilon_1 = \sqrt{\sum\nolimits_{i=1}^{n} (s_i t_i)^2}$$

$\varepsilon_2 = ||s||_2$, s = `XSCALE`, and t = `XGUESS`.

`RPARAM`(7) = Size of initial trust region radius.
Default: based on the initial scaled Cauchy step.

If double precision is desired, then `DU4LSF` is called and `RPARAM` is declared double precision.

5. Users wishing to override the default print/stop attributes associated with error messages issued by this routine are referred to `ERROR HANDLING` in the Introduction.

Algorithm

The routine `BCLSJ` uses a modified Levenberg-Marquardt method and an active set strategy to solve nonlinear least squares problems subject to simple bounds on the variables. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} \frac{1}{2} F(x)^T F(x) = \frac{1}{2} \sum_{i=1}^{m} f_i(x)^2$$

subject to $l \le x \le u$

where $m \ge n$, $F : \mathbf{R}^n \rightarrow \mathbf{R}^m$, and $f_i(x)$ is the i-th component function of $F(x)$. From a given starting point, an active set IA, which contains the indices of the variables at their bounds, is built. A variable is called a “free variable” if it is not in the active set. The routine then computes the search direction for the free variables according to the formula

$$d = -(J^T J + \mu I)^{-1} J^T F$$

where is the Levenberg-Marquardt parameter, $F = F(x)$, and J is the Jacobian with respect to the free variables. The search direction for the variables in IA is set to zero. The trust region approach discussed by Dennis and Schnabel (1983) is used to find the new point. Finally, the optimality conditions are checked. The conditions are

$$||g(x_i)|| \le \varepsilon,\ l_i < x_i < u_i$$

$$g(x_i) < 0, \;\; x_i = u_i$$

$$g(x_i) > 0, \; x_i = l_i$$

where ε is a gradient tolerance. This process is repeated until the optimality criterion is achieved.

The active set is changed only when a free variable hits its bounds during an iteration or the optimality condition is met for the free variables but not for all variables in IA, the active set. In the latter case, a variable that violates the optimality condition will be dropped out of IA. For more detail on the Levenberg-Marquardt method, see Levenberg (1944) or Marquardt (1963). For more detailed information on active set strategy, see Gill and Murray (1976).

Example

The nonlinear least squares problem

$$\min_{x\in\mathbf{R}^2} \frac{1}{2}\sum_{i=1}^{2} f_i(x)^2$$

$$\text{subject to } -2 \le x_1 \le 0.5$$

$$-1 \le x_2 \le 2$$

where

$$f_1(x) = 10\left(x_2 - x_1^2\right) \text{ and } f_2(x) = \left(1 - x_1\right)$$

is solved with an initial guess (–1.2, 1.0) and default values for parameters.

```
C                                  Declaration of variables
      INTEGER    LDFJAC, M, N
      PARAMETER  (LDFJAC=2, M=2, N=2)
C
      INTEGER    IPARAM(7), ITP, NOUT
      REAL       FJAC(LDFJAC,N), FSCALE(M), FVEC(M),
     &           RPARAM(7), X(N), XGUESS(N), XLB(N), XS(N), XUB(N)
      EXTERNAL   BCLSJ, ROSBCK, ROSJAC, UMACH
C                                  Compute the least squares for the
C                                  Rosenbrock function.
      DATA XGUESS/-1.2E0, 1.0E0/, XS/2*1.0E0/, FSCALE/2*1.0E0/
      DATA XLB/-2.0E0, -1.0E0/, XUB/0.5E0, 2.0E0/
C                                  All the bounds are provided
      ITP = 0
C                                  Default parameters are used
      IPARAM(1) = 0
C
      CALL BCLSJ (ROSBCK, ROSJAC, M, N, XGUESS, ITP, XLB, XUB, XS,
     &            FSCALE, IPARAM, RPARAM, X, FVEC, FJAC, LDFJAC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, FVEC, IPARAM(3), IPARAM(4)
C
99999 FORMAT ('  The solution is ', 2F9.4, //, '  The function ',
     &       'evaluated at the solution is ', /, 18X, 2F9.4, //,
     &       '  The number of iterations is ', 10X, I3, /, '  The ',
     &       'number of function evaluations is ', I3, /)
      END
C
      SUBROUTINE ROSBCK (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
C
      F(1) = 1.0E1*(X(2)-X(1)*X(1))
      F(2) = 1.0E0 - X(1)
      RETURN
      END
C
      SUBROUTINE ROSJAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
C
      FJAC(1,1) = -20.0E0*X(1)
      FJAC(2,1) = -1.0E0
      FJAC(1,2) = 10.0E0
      FJAC(2,2) = 0.0E0
      RETURN
      END
```

Output

```
The solution is    0.5000   0.2500

The function evaluated at the solution is
0.0000   0.5000

The number of iterations is            13
The number of function evaluations is  21
```

BCNLS/DBCNLS (Single/Double precision)

Solve a nonlinear least-squares problem subject to bounds on the variables and general linear constraints.

Usage

```
CALL BCNLS (FCN, M, N, MCON, C, LDC, BL, BU, IRTYPE,
            XLB, XUB, XGUESS, X, RNORM, ISTAT)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (M, N, X, F), where

M – Number of functions. (Input)

N – Number of variables. (Input)

X – Array of length N containing the point at which the function will be evaluated. (Input)

F – Array of length M containing the computed function at the point X. (Output)

The routine FCN must be declared EXTERNAL in the calling program.

M — Number of functions. (Input)

N — Number of variables. (Input)

MCON — The number of general linear constraints for the system, not including simple bounds. (Input)

C — `MCON` × `N` matrix containing the coefficients of the `MCON` general linear constraints. (Input)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)
`LDC` must be at least `MCON`.

BL — Vector of length `MCON` containing the lower limit of the general constraints. (Input).

BU — Vector of length `MCON` containing the upper limit of the general constraints. (Input).

IRTYPE — Vector of length `MCON` indicating the types of general constraints in the matrix `C`. (Input)
Let `R(I) = C(I, 1)*X(1) + … + C(I, N)*X(N)`. Then the value of `IRTYPE(I)` signifies the following:

IRTYPE(I)	**I-th CONSTRAINT**
0	`BL(I).EQ.R(I).EQ.BU(I)`
1	`R(I).LE.BU(I)`
2	`R(I).GE.BL(I)`
3	`BL(I).LE.R(I).LE.BU(I)`

XLB — Vector of length `N` containing the lower bounds on variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound. (Input)

XUB — Vector of length `N` containing the upper bounds on variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)

XGUESS — Vector of length `N` containing the initial guess. (Input)

X — Vector of length `N` containing the approximate solution. (Output)

RNORM — The Euclidean length of components of the function $f(x)$ after the approximate solution has been found. (Output).

ISTAT — Scalar indicating further information about the approximate solution `X`. (Output)
See the Comments section for a description of the tolerances and the vectors `IPARAM` and `RPARAM`.

ISTAT	**Meaning**
1	The function *f* (*x*) has a length less than TOLF = RPARAM(1). This is the expected value for ISTAT when an actual zero value of *f* (*x*) is anticipated.
2	The function *f* (*x*) has reached a local minimum. This is the expected value for ISTAT when a nonzero value of *f* (*x*) is anticipated.
3	A small change (absolute) was noted for the vector *x*. A full model problem step was taken. The condition for ISTAT = 2 may also be satisfied, so that a minimum has been found. However, this test is made before the test for ISTAT = 2.
4	A small change (relative) was noted for the vector *x*. A full model problem step was taken. The condition for ISTAT = 2 may also be satisfied, so that a minimum has been found. However, this test is made before the test for ISTAT = 2.
5	The number of terms in the quadratic model is being restricted by the amount of storage allowed for that purpose. It is suggested, but not required, that additional storage be given for the quadratic model parameters. This is accessed through the vector *IPARAM*, documented below.
6	Return for evaluation of function and Jacobian if reverse communication is desired. See the Comments below.

Comments

1. Automatic workspace is

BCNLS 51N + M + 15MCON + (M + MCON)(N + 1) + 4MX + NA(NA + 8) + 5(M + MX + 14) + 70 units, or

DBCNLS 92N + M +26MCON + 2(M + MCON)(N + 1) + 7MX + 2NA(NA + 8) + 10(M + MX + 14) + 99 units.

where MX = MAX(M, N), NA = MCON + 2N + 6

Workspace may be explicitly provided, if desired, by use of B2NLS/DB2NLS. The reference is

```
CALL B2NLS (FCN, M, N, MCON, C, LDC, BL, BU,
            IRTYPE, XLB, XUB, XGUESS, X, RNORM,
            ISTAT, IPARAM, RPARAM, JAC, F, FJ, LDFJ,
            IWORK, LIWORK, WORK, LWORK)
```

The additional arguments are as follows:

IPARAM — Integer vector of length six used to change certain default attributes of BCNLS. (Input).
If the default parameters are desired for BCNLS, set IPARAM(1) to zero. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, the following steps should be taken before calling B2NLS:

```
CALL B7NLS (IPARAM, RPARAM)
```

Set nondefault values for IPARAM and RPARAM.

If double precision is being used, DB7NLS should be called instead. Following is a list of parameters and the default values.

IPARAM(1) = Initialization flag.

IPARAM(2) = ITMAX, the maximum number of iterations allowed.
Default: 75

IPARAM(3) = a flag that suppresses the use of the quadratic model in the inner loop. If set to one, then the quadratic model is never used. Otherwise use the quadratic model where appropriate. This option decreases the amount of workspace as well as the computing overhead required. A user may wish to determine if the application really requires the use of the quadratic model.
Default: 0

IPARAM(4) = NTERMS, one more than the maximum number of terms used in the quadratic model.
Default: 5

IPARAM(5) = RCSTAT, a flag that determines whether forward or reverse communication is used. If set to zero, forward communication through functions FCN and JAC is used. If set to one, reverse communication is used, and the dummy routines B10LS/DB10LS and B11LS/DB11LS may be used in place of FCN and JAC, respectively. When BCNLS returns with ISTAT = 6, arrays F and FJ are filled with $f(x)$ and the Jacobian of $f(x)$, respectively. BCNLS is then called again.
Default: 0

IPARAM(6) = a flag that determines whether the analytic Jacobian, as supplied in JAC, is used, or if a finite difference approximation is computed. If set to zero, JAC is not accessed and finite differences are used. If set to one, JAC is used to compute the Jacobian.
Default: 0

RPARAM — Real vector of length 7 used to change certain default attributes of BCNLS. (Input)

For the description of RPARAM, we make the following definitions:

FC	current value of the length of $f(x)$
FB	best value of length of $f(x)$
FL	value of length of $f(x)$ at the previous step
PV	predicted value of length of $f(x)$, after the step is taken, using the approximating model
ε	machine epsilon = amach(4)

The conditions |FB − PV| ≤ TOLSNR*FB and |FC − PV| ≤ TOLP*FB and |FC − FL| ≤ TOLSNR*FB together with taking a full model step, must be satisfied before the condition ISTAT = 2 is returned. (Decreasing any of the values for TOLF, TOLD, TOLX, TOLSNR, or TOLP will likely increase the number of iterations required for convergence.)

RPARAM(1) = TOLF, tolerance used for stopping when FC ≤ TOLF.
$\text{Default: } \min(1.\text{E}-5, \sqrt{\varepsilon})$

RPARAM(2) = TOLX, tolerance for stopping when change to *x* values has length less than or equal to TOLX*length of *x* values.
$\text{Default: } \min(1.\text{E}-5, \sqrt{\varepsilon})$

RPARAM(3) = TOLD, tolerance for stopping when change to *x* values has length less than pr equal to TOLD.
$\text{Default: } \min(1.\text{E}-5, \sqrt{\varepsilon})$

RPARAM(4) = TOLSNR, tolerance used in stopping condition ISTAT = 2.
Default: 1.E–5

RPARAM(5) = TOLP, tolerance used in stopping condition ISTAT = 2.
Default: 1.E–5

RPARAM(6) = TOLUSE, tolerance used to avoid values of *x* in the quadratic model's interpolation of previous points. Decreasing this value may result in more terms being included in the quadratic model.
$\text{Default:} \sqrt{\varepsilon}$

RPARAM(7) = COND, largest condition number to allow when solving for the quadratic model coefficients. Increasing this value may result in more terms being included in the quadratic model.
Default: 30

JAC — User-supplied SUBROUTINE to evaluate the Jacobian. The usage is
CALL JAC(M, N, X, FJAC, LDFJAC), where
M – Number of functions. (Input)
N – Number of variables. (Input)
X – Array of length N containing the point at which the Jacobian will be evaluated. (Input)
FJAC – The computed M × N Jacobian at the point X. (Output)
LDFJAC – Leading dimension of the array FJAC. (Input)
The routine JAC must be declared EXTERNAL in the calling program.

F — Real vector of length N used to pass *f*(*x*) if reverse communication (IPARAM(4)) is enabled. (Input)

FJ — Real array of size M × N used to store the Jacobian matrix of *f*(*x*) if reverse communication (IPARAM(4)) is enabled. (Input)
Specifically,

$$FJ(i,j) = \frac{\partial f_i}{\partial x_j}$$

LDFJ — Leading dimension of FJ exactly as specified in the dimension statement of the calling program. (Input)

IWORK — Integer work vector of length LIWORK.

LIWORK — Length of work vector IWORK. LIWORK must be at least 5MCON + 12N + 47 + MAX(M, N)

WORK — Real work vector of length LWORK

LWORK — Length of work vector WORK. LWORK must be at least 41N + 6M + 11MCON + (M + MCON)(N + 1) + NA(NA + 7) + 8 MAX(M, N) + 99. Where NA = MCON + 2N + 6.

2. Informational errors

Type	Code	
3	1	The function $f(x)$ has reached a value that may be a local minimum. However, the bounds on the trust region defining the size of the step are being hit at each step. Thus, the situation is suspect. (Situations of this type can occur when the solution is at infinity at some of the components of the unknowns, x).
3	2	The model problem solver has noted a value for the linear or quadratic model problem residual vector length that is greater than or equal to the current value of the function, i.e. the Euclidean length of $f(x)$. This situation probably means that the evaluation of $f(x)$ has more uncertainty or noise than is possible to account for in the tolerances used to not a local minimum. The value of x is suspect, but a minimum has probably been found.
3	3	More than ITMAX iterations were taken to obtain the solution. The value obtained for x is suspect, although it is the best set of x values that occurred in the entire computation. The value of ITMAX can be increased though the IPARAM vector.

Algorithm

The routine BCNLS solves the nonlinear least squares problem

$$\min \sum_{i=1}^{m} f_i(x)^2$$

subject to

$$b_l \le Cx \le b_u$$
$$x_l \le x \le x_u$$

BCNLS is based on the routine DQED by R.J. Hanson and F.T. Krogh. The section of BCNLS that approximates, using finite differences, the Jacobian of $f(x)$ is a modification of JACBF by D.E. Salane.

Example 1

This example finds the four variables x_1, x_2, x_3, x_4 that are in the model function

$$h(t) = x_1 e^{x_2 t} + x_3 e^{x_4 t}$$

There are values of *h(t)* at five values of *t*.

$h(0.05) = 2.206$

$h(0.1) = 1.994$

$h(0.4) = 1.35$

$h(0.5) = 1.216$

$h(1.0) = 0.7358$

There are also the constraints that x_2, $x_4 \le 0$, x_1, $x_3 \ge 0$, and x_2 and x_4 must be separated by at least 0.05. Nothing more about the values of the parameters is know so the initial guess is 0.

```
      INTEGER    MCON, N
      PARAMETER  (MCON=1, N=4)
C                                  SPECIFICATIONS FOR PARAMETERS
      INTEGER    LDC, M
      PARAMETER  (M=5, LDC=MCON)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    IRTYPE(MCON), ISTAT, NOUT
      REAL       BL(MCON), C(MCON,N), RNORM, X(N), XGUESS(N), XLB(N),
     &           XUB(N)
C                                  SPECIFICATIONS FOR SUBROUTINES
      EXTERNAL   BCNLS, SSET, UMACH, WRRRN
C                                  SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL   FCN
C
      CALL UMACH (2, NOUT)
C                                  Define the separation between x(2)
C                                  and x(4)
      C(1,1) = 0.0
      C(1,2) = 1.0
      C(1,3) = 0.0
      C(1,4) = -1.0
      BL(1) = 0.05
      IRTYPE(1) = 2
C                                  Set lower bounds on variables
      XLB(1) = 0.0
      XLB(2) = 1.0E30
      XLB(3) = 0.0
      XLB(4) = 1.0E30
C                                  Set upper bounds on variables
      XUB(1) = -1.0E30
      XUB(2) = 0.0
      XUB(3) = -1.0E30
      XUB(4) = 0.0
C                                  Set initial guess to 0.0
      CALL SSET (N, 0.0, XGUESS, 1)
C
      CALL BCNLS (FCN, M, N, MCON, C, LDC, BL, BL, IRTYPE, XLB,
     &             XUB, XGUESS, X, RNORM, ISTAT)
C
      CALL WRRRN ('X', 1, N, X, 1, 0)
      WRITE (NOUT,99999) RNORM
99999 FORMAT (/, 'rnorm = ', E10.5)
      END
C
      SUBROUTINE FCN (M, N, X, F)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER    M, N
      REAL       X(*), F(*)
C                                  SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I
C                                  SPECIFICATIONS FOR SAVE VARIABLES
      REAL       H(5), T(5)
      SAVE       H, T
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  EXP
      REAL       EXP
C
      DATA T/0.05, 0.1, 0.4, 0.5, 1.0/
      DATA H/2.206, 1.994, 1.35, 1.216, 0.7358/
C
      DO 10  I=1, M
         F(I) = X(1)*EXP(X(2)*T(I)) + X(3)*EXP(X(4)*T(I)) - H(I)
   10 CONTINUE
      RETURN
      END
```

Output

```
                    X
       1        2        3        4
   1.999   -1.000    0.500   -9.954
rnorm = .42438E-03
```

Example 2

This example solves the same problem as the last example, but reverse communication is used to evaluate *f*(*x*) and the Jacobian of *f*(*x*). The use of the quadratic model is turned off.

```
      INTEGER    LDC, LDFJ, M, MCON, N
      PARAMETER  (M=5, MCON=1, N=4, LDC=MCON, LDFJ=M)
C                                  Specifications for local variables
      INTEGER    I, IPARAM(6), IRTYPE(MCON), ISTAT, IWORK(1000),
     &           LIWORK, LWORK, NOUT
      REAL       BL(MCON), C(MCON,N), F(M), FJ(M,N), RNORM, RPARAM(7),
     &           WORK(1000), X(N), XGUESS(N), XLB(N), XUB(N)
      REAL       H(5), T(5)
      SAVE       H, T
      INTRINSIC  EXP
      REAL       EXP
C                                  Specifications for subroutines
      EXTERNAL   B2NLS, B7NLS, SSET, UMACH, WRRRN
C                                  Specifications for functions
      EXTERNAL   B10LS, B11LS
C
      DATA T/0.05, 0.1, 0.4, 0.5, 1.0/
      DATA H/2.206, 1.994, 1.35, 1.216, 0.7358/
c
      CALL UMACH (2, NOUT)
c                                  Define the separation between x(2)
c                                  and x(4)
      C(1,1)    = 0.0
      C(1,2)    = 1.0
      C(1,3)    = 0.0
      C(1,4)    = -1.0
      BL(1)     = 0.05
      IRTYPE(1) = 2
c                                  Set lower bounds on variables
      XLB(1) = 0.0
      XLB(2) = 1.0E30
      XLB(3) = 0.0
      XLB(4) = 1.0E30
c                                  Set upper bounds on variables
      XUB(1) = -1.0E30
      XUB(2) = 0.0
      XUB(3) = -1.0E30
      XUB(4) = 0.0
c                                  Set initial guess to 0.0
      CALL SSET (N, 0.0, XGUESS, 1)
c                                  Call B7NLS to set default parameters
      CALL B7NLS (IPARAM, RPARAM)
c                                  Suppress the use of the quadratic
c                                  model, evaluate functions and
c                                  Jacobian by reverse communication
      IPARAM(3) = 1
      IPARAM(5) = 1
      IPARAM(6) = 1
      LWORK     = 1000
      LIWORK    = 1000
c                                  Specify dummy routines for FCN
c                                  and JAC since we are using reverse
c                                  communication
   10 CONTINUE
      CALL B2NLS (B10LS, M, N, MCON, C, LDC, BL, BL, IRTYPE, XLB,
     &            XUB, XGUESS, X, RNORM, ISTAT, IPARAM, RPARAM,
     &            B11LS, F, FJ, LDFJ, IWORK, LIWORK, WORK, LWORK)
c
c                                  Evaluate functions if the routine
c                                  returns with ISTAT = 6
      IF (ISTAT .EQ. 6) THEN
         DO 20  I=1, M
            FJ(I,1) = EXP(X(2)*T(I))
            FJ(I,2) = T(I)*X(1)*FJ(I,1)
            FJ(I,3) = EXP(X(4)*T(I))
            FJ(I,4) = T(I)*X(3)*FJ(I,3)
            F(I) = X(1)*FJ(I,1) + X(3)*FJ(I,3) - H(I)
   20    CONTINUE
         GO TO 10
      END IF
c
      CALL WRRRN ('X', 1, N, X, 1, 0)
      WRITE (NOUT,99999) RNORM
99999 FORMAT (/, 'rnorm = ', E10.5)
      END
```

Output

```
                     X
       1       2       3       4
   1.999  -1.000   0.500  -9.954
rnorm = .42413E-03
```

DLPRS/DDLPRS (Single/Double precision)

Solve a linear programming problem via the revised simplex algorithm.

Usage

```
CALL DLPRS (M, NVAR, A, LDA, BL, BU, C, IRTYPE, XLB, XUB,
            OBJ, XSOL, DSOL)
```

Arguments

M — Number of constraints.   (Input)

NVAR — Number of variables.   (Input)

A — `M` by `NVAR` matrix containing the coefficients of the `M` constraints.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)
`LDA` must be at least `M`.

BL — Vector of length `M` containing the lower limit of the general constraints; if there is no lower limit on the `I`-th constraint, then `BL(I)` is not referenced.   (Input)

BU — Vector of length `M` containing the upper limit of the general constraints; if there is no upper limit on the `I`-th constraint, then `BU(I)` is not referenced; if there are no range constraints, `BL` and `BU` can share the same storage locations.   (Input)

C — Vector of length `NVAR` containing the coefficients of the objective function.   (Input)

IRTYPE — Vector of length `M` indicating the types of general constraints in the matrix `A`.   (Input)
Let `R(I)` = `A(I`, 1) `*` `XSOL`(1) + … + `A(I`, `NVAR`) `*` `XSOL(NVAR)`. Then, the value of `IRTYPE(I)` signifies the following:

IRTYPE(I)	**I-th Constraint**
0	`BL(I).EQ.R(I).EQ.BU(I)`
1	`R(I).LE.BU(I)`
2	`R(I).GE.BL(I)`
3	`BL(I).LE.R(I).LE.BU(I)`

XLB — Vector of length `NVAR` containing the lower bound on the variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound.   (Input)

XUB — Vector of length NVAR containing the upper bound on the variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)

OBJ — Value of the objective function. (Output)

XSOL — Vector of length NVAR containing the primal solution. (Output)

DSOL — Vector of length M containing the dual solution. (Output)

Comments

1. Automatic workspace usage is

   DLPRS M * M + 57 * M + 3 * NVAR units, or
   DDLPRS 2 * M * M + 85 * M + 3 * NVAR units.

   Workspace may be explicitly provided, if desired, by use of D2PRS/DD2PRS. The reference is

   ```
   CALL D2PRS (M, NVAR, A, LDA, BL, BU, C, IRTYPE, XLB,
               XUB, OBJ, XSOL, DSOL, AWK, LDAWK, WK,
               IWK)
   ```

   The additional arguments are as follows:

   AWK — Real work array of dimension 1 by 1. (AWK is not used in the new implementation of the revised simplex algorithm. It is retained merely for calling sequence consistency.)

   LDAWK — Leading dimension of AWK exactly as specified in the dimension statement of the calling program. LDAWK should be 1. (LDAWK is not used in the new implementation of the revised simplex algorithm. It is retained merely for calling sequence consistency.)

   WK — Real work vector of length M * (M + 28).

   IWK — Integer work vector of length 29 * M + 3 * NVAR.

2. Informational errors

Type	Code	
3	1	The problem is unbounded.
4	2	Maximum number of iterations exceeded.
3	3	The problem is infeasible.
4	4	Numerical difficulty occurred; using double precision may help.
4	5	The bounds are inconsistent.

Algorithm

The routine DLPRS uses a revised simplex method to solve linear programming problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} c^T x$$

$$\text{subject to } b_l \le Ax \le b_u$$

$$x_l \le x \le x_u$$

where c is the objective coefficient vector, A is the coefficient matrix, and the vectors b_l, b_u, x_l and x_u are the lower and upper bounds on the constraints and the variables, respectively.

For a complete description of the revised simplex method, see Murtagh (1981) or Murty (1983).

Example

A linear programming problem is solved.

```
      INTEGER    LDA, M, NVAR
      PARAMETER  (M=2, NVAR=2, LDA=M)
C                                 M = number of constraints
C                                 NVAR = number of variables
C
      INTEGER    I, IRTYPE(M), NOUT
      REAL       A(LDA,NVAR), B(M), C(NVAR), DSOL(M), OBJ, XLB(NVAR),
     &           XSOL(NVAR), XUB(NVAR)
      EXTERNAL   DLPRS, SSCAL, UMACH
C
C                                 Set values for the following problem
C
C                                 Max 1.0*XSOL(1) + 3.0*XSOL(2)
C
C                                 XSOL(1) + XSOL(2) .LE. 1.5
C                                 XSOL(1) + XSOL(2) .GE. 0.5
C
C                                 0 .LE. XSOL(1) .LE. 1
C                                 0 .LE. XSOL(2) .LE. 1
C
      DATA XLB/2*0.0/, XUB/2*1.0/
      DATA A/4*1.0/, B/1.5, .5/, C/1.0, 3.0/
      DATA IRTYPE/1, 2/
C                                 To maximize, C must be multiplied by
C                                 -1.
      CALL SSCAL (NVAR, -1.0E0, C, 1)
C                                 Solve the LP problem.  Since there is
C                                 no range constraint, only B is
C                                 needed.
      CALL DLPRS (M, NVAR, A, LDA, B, B, C, IRTYPE, XLB, XUB, OBJ,
     &            XSOL, DSOL)
C                                 OBJ must be multiplied by -1 to get
C                                 the true maximum.
      OBJ = -OBJ
C                                 DSOL must be multiplied by -1 for
C                                 maximization.
      CALL SSCAL (M, -1.0E0, DSOL, 1)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) OBJ, (XSOL(I),I=1,NVAR), (DSOL(I),I=1,M)
C
99999 FORMAT (//, ’   Objective       = ’, F9.4, //, ’   Primal ’,
     &       ’Solution =’, 2F9.4, //, ’   Dual solution   =’, 2F9.4)
```

```
C
      END
```

Output

```
Objective       =    3.5000

Primal Solution =   0.5000   1.0000

Dual solution   =   1.0000   0.0000
```

SLPRS/DSLPRS (Single/Double precision)

Solve a sparse linear programming problem via the revised simplex algorithm.

Usage

```
CALL SLPRS (M, NVAR, NZ, A, IROW, JCOL, BL, BU, C, IRTYPE,
            XLB, XUB, OBJ, XSOL, DSOL)
```

Arguments

M — Number of constraints.   (Input)

NVAR — Number of variables.   (Input)

NZ — Number of nonzero coefficients in the matrix *A*.   (Input)

A — Vector of length `NZ` containing the coefficients of the `M` constraints.   (Input)

IROW — Vector of length `NZ` containing the row numbers of the corresponding element in `A`.   (Input)

JCOL — Vector of length `NZ` containing the column numbers of the corresponding elements in *A*. (Input)

BL — Vector of length `M` containing the lower limit of the general constraints; if there is no lower limit on the `I`-th constraint, then `BL(I)` is not referenced. (Input)

BU — Vector of length `M` containing the lower limit of the general constraints; if there is no lower limit on the `I`-th constraint, then `BU(I)` is not referenced. (Input)

C — Vector of length `NVAR` containing the coefficients of the objective function. (Input)

IRTYPE — Vector of length `M` indicating the types of general constraints in the matrix `A`.   (Input)
Let `R(I) = A(I, 1)*XSOL(1) + … + A(I, NVAR)*XSOL(NVAR)`

IRTYPE(I)	**I-th CONSTRAINT**
0	BL(I) = R(I) = BU(I)
1	R(I) ≤ BU(I)

2	R(I) ≥ BL(I)
3	BL(I) ≤ R(I) ≤ BU(I)

XLB — Vector of length NVAR containing the lower bound on the variables; if there is no lower bound on a variable, then 1.0E30 should be set as the lower bound. (Input)

XUB — Vector of length NVAR containing the upper bound on the variables; if there is no upper bound on a variable, then −1.0E30 should be set as the upper bound. (Input)

OBJ — Value of the objective function. (Output)

XSOL — Vector of length NVAR containing the primal solution. (Output)

DSOL — Vector of length M containing the dual solution. (Output)

Comments

1. Automatic workspace is

   S2PRS 5NVAR + 62M + 2MAX(NZ + NVAR + 8, 4NVAR + 7) units, or
   DS2PRS 9NVAR + 85M + 3MAX(NZ + NVAR + 8, 4NVAR + 7) units.

   Workspace may be explicitly provided, if desired, by use of S2PRS/DS2PRS. The reference is

   ```
   CALL S2PRS (M, NVAR, NZ, A, IROW, JCOL, BL, BU, C,
               IRTYPE, XLB, XUB, OBJ, XSOL, DSOL,
               IPARAM, RPARAM, COLSCL, ROWSCL, WORK,
               LW, IWORK, LIW)
   ```

   The additional arguments are as follows:

   IPARAM — Integer parameter vector of length 12. If the default parameters are desired for SLPRS, then set IPARAM(1) to zero and call the routine SLPRS. Otherwise, if any nondefault parameters are desired for IPARAM or RPARAM, then the following steps should be taken before calling SLPRS:

   CALL S5PRS (IPARAM, RPARAM)
   Set nondefault values for IPARAM and RPARAM.

   Note that the call to S5PRS will set IPARAM and RPARAM to their default values so only nondefault values need to be set above.

   IPARAM(1) = 0 indicates that a minimization problem is solved. If set to 1, a maximization problem is solved.
   Default: 0

   IPARAM(2) = switch indicating the maximum number of iterations to be taken before returning to the user. If set to zero, the maximum number of iterations taken is set to 3*(NVARS+M). If positive, that value is used as the iteration limit.
   Default: IPARAM(2) = 0

IPARAM(3) = indicator for choosing how columns are selected to enter the basis. If set to zero, the routine uses the steepest edge pricing strategy which is the best local move. If set to one, the minimum reduced cost pricing strategy is used. The steepest edge pricing strategy generally uses fewer iterations than the minimum reduced cost pricing, but each iteration costs more in terms of the amount of calculation performed. However, this is very problem-dependent.
Default: IPARAM(3) = 0

IPARAM(4) = MXITBR, the number of iterations between recalculating the error in the primal solution is used to monitor the error in solving the linear system. This is an expensive calculation and every tenth iteration is generally enough.
Default: IPARAM(4) = 10

IPARAM(5) = NPP, the number of negative reduced costs (at most) to be found at each iteration of choosing a variable to enter the basis. If set to zero, NPP = NVARS will be used, implying that all of the reduced costs are computed at each such step. This "Partial pricing" may increase the total number of iterations required. However, it decreases the number of calculation required at each iteration. The effect on overall efficiency is very problem-dependent. If set to some positive number, that value is used as NPP.
Default: IPARAM(5) = 0

IPARAM(6) = IREDFQ, the number of steps between basis matrix redecompositions. Redecompositions also occur whenever the linear systems for the primal and dual systems have lost half their working precision.
Default: IPARAM(6) = 50

IPARAM(7) = LAMAT, the length of the portion of WORK that is allocated to sparse matrix storage and decomposition. LAMAT must be greater than NZ + NVARS + 4.
Default: LAMAT = NZ + NVARS + 5

IPARAM(8) = LBM, then length of the portion of IWORK that is allocated to sparse matrix storage and decomposition. LBM must be positive.
Default: LBM = 8*M

IPARAM(9) = switch indicating that partial results should be saved after the maximum number of iterations, IPARAM(2), or at the optimum. If IPARAM(9) is not zero, data essential to continuing the calculation is saved to a file, attached to unit number IPARAM(9). The data saved includes all the information about the sparse matrix A and information about the current basis. If IPARAM(9) is set to zero, partial results are not saved. It is the responsibility of the calling program to open the output file.

IPARAM(10) = switch indicating that partial results have been computed and stored on unit number IPARAM(10), if greater than zero.

If IPARAM(10) is zero, a new problem is started.
Default: IPARAM(10) = 0

IPARAM(11) = switch indicating that the user supplies scale factors for the columns of the matrix *A*. If IPARAM(11) = 0, SLPRS computes the scale factors as the reciprocals of the max norm of each column. If IPARAM(11) is set to one, element I of the vector COLSCL is used as the scale factor for column I of the matrix *A*. The scaling is implicit, so no input data is actually changed.
Default: IPARAM(11) = 0

IPARAM(12) = switch indicating that the user supplied scale factors for the rows of the matrix *A*. If IPARAM(12) is set to zero, no row scaling is one. If IPARAM(12) is set to 1, element I of the vector ROWSCL is used as the scale factor for row I of the matrix *A*. The scaling is implicit, so no input data is actually changed.
Default: IPARAM(12) = 0

RPARAM — Real parameter vector of length 7.
RPARAM(1) = COSTSC, a scale factor for the vector of costs. Normally SLPRS computes this scale factor to be the reciprocal of the max norm if the vector costs after the column scaling has been applied. If RPARAM(1) is zero, SLPRS compute COSTSC.
Default: RPARAM(1) = 0.0

RPARAM(2) = ASMALL, the smallest magnitude of nonzero entries in the matrix *A*. If RPARAM(2) is nonzero, checking is done to ensure that all elements of *A* are at least as large as RPARAM(2). Otherwise, no checking is done.
Default: RPARAM(2) = 0.0

RPARAM(3) = ABIG, the largest magnitude of nonzero entries in the matrix *A*. If RPARAM(3) is nonzero, checking is done to ensure that all elements of *A* are no larger than RPARAM(3). Otherwise, no checking is done.
Default: RPARAM(3) = 0.0

RPARAM(4) = TOLLS, the relative tolerance used in checking if the residuals are feasible. RPARAM(4) is nonzero, that value is used as TOLLS, otherwise the default value is used.
Default: TOLLS = 1000.0*amach(4)

RPARAM(5) = PHI, the scaling factor used to scale the reduced cost error estimates. In some environments, it may be necessary to reset PHI to the range [0.01, 0.1], particularly on machines with short word length and working precision when solving a large problem. If RPARAM(5) is nonzero, that value is used as PHI, otherwise the default value is used.
Default: PHI = 1.0

RPARAM(6) = TOLABS, an absolute error test on feasibility. Normally a relative test is used with TOLLS (see RPARAM(4)). If this test fails, an absolute test will be applied using the value TOLABS.
Default: TOLABS = 0.0

RPARAM(7) = pivot tolerance of the underlying sparse factorization routine. If RPARAM(7) is set to zero, the default pivot tolerance is used, otherwise, the RPARAM(7) is used.
Default: RPARAM(7) = 0.1

COLSCL — Array of length NVARS containing column scale factors for the matrix *A*. (Input).
COLSCL is not used if IPARAM(11) is set to zero.

ROWSCL — Array of length M containing row scale factors for the matrix *A*. (Input)
ROWSCL is not used if IPARAM(12) is set to zero.

WORK — Work array of length LW.

LW — Length of real work array. LW must be at least 4NVAR + 23M + MAX(NZ + NVAR + 8, 4NVAR + 7).

IWORK — Integer work array of length LIW.

LIW — Length of integer work array. LIW must be at least NVAR + 39M + MAX(NZ + NVAR + 8, 4NVAR + 7).

Algorithm

This subroutine solves problems of the form

$$\min c^T x$$

subject to

$$b_l \le Ax \le b_u,$$
$$x_l \le x \le x_u$$

where c is the objective coefficient vector, A is the coefficient matrix, and the vectors b_l, b_u, x_l, and x_u are the lower and upper bounds on the constraints and the variables, respectively. SLPRS is designed to take advantage of sparsity in A. The routine is based on DPLO by Hanson and Hiebert.

Example

Solve a linear programming problem, with

$$A=\begin{bmatrix} 0 & 0.5 & & & \\ & 1 & 0.5 & & \\ & & 1 & \ddots & \\ & & & \ddots & 0.5 \\ & & & & 1 \end{bmatrix}$$

defined in sparse coordinate format.

```
      INTEGER    M, NVAR
      PARAMETER  (M=200, NVAR=200)
C                                  Specifications for local variables
      INTEGER    INDEX, IROW(3*M), J, JCOL(3*M), NOUT, NZ
      REAL       A(3*M), DSOL(M), OBJ, XSOL(NVAR)
      INTEGER    IRTYPE(M)
      REAL       B(M), C(NVAR), XL(NVAR), XU(NVAR)
C                                  Specifications for subroutines
      EXTERNAL   SLPRS, UMACH
C
      DATA B/199*1.7, 1.0/
      DATA C/-1.0, -2.0, -3.0, -4.0, -5.0, -6.0, -7.0, -8.0, -9.0,
     &     -10.0, 190*-1.0/
      DATA XL/200*0.1/
      DATA XU/200*2.0/
      DATA IRTYPE/200*1/
C
      CALL UMACH (2, NOUT)
C                                  Define A
      INDEX = 1
      DO 10  J=2, M
C                                  Superdiagonal element
         IROW(INDEX) = J - 1
         JCOL(INDEX) = J
         A(INDEX)    = 0.5
C                                  Diagonal element
         IROW(INDEX+1) = J
         JCOL(INDEX+1) = J
         A(INDEX+1) = 1.0
         INDEX      = INDEX + 2
   10 CONTINUE
      NZ = INDEX - 1
C
C
      XL(4) = 0.2
      CALL SLPRS (M, NVAR, NZ, A, IROW, JCOL, B, B, C, IRTYPE, XL, XU,
     &            OBJ, XSOL, DSOL)
C
      WRITE (NOUT,99999) OBJ
C
99999 FORMAT (/, 'The value of the objective function is ', E12.6)
C
      END
```

Output

```
The value of the objective function is -.280971E+03
```

QPROG/DQPROG (Single/Double precision)

Solve a quadratic programming problem subject to linear equality/inequality constraints.

Usage

```
CALL QPROG (NVAR, NCON, NEQ, A, LDA, B, G, H, LDH, DIAG,
            SOL, NACT, IACT, ALAMDA)
```

Arguments

NVAR — The number of variables. (Input)

NCON — The number of linear constraints. (Input)

NEQ — The number of linear equality constraints. (Input)

A — NCON by NVAR matrix. (Input)
The matrix contains the equality contraints in the first NEQ rows followed by the inequality constraints.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length NCON containing right-hand sides of the linear constraints. (Input)

G — Vector of length NVAR containing the coefficients of the linear term of the objective function. (Input)

H — NVAR by NVAR matrix containing the Hessian matrix of the objective function. (Input)
H should be symmetric positive definite; if H is not positive definite, the algorithm attempts to solve the QP problem with H replaced by a H + DIAG * I such that H + DIAG * I is positive definite. See Comment 3.

LDH — Leading dimension of H exactly as specified in the dimension statement of the calling program. (Input)

DIAG — Scalar equal to the multiple of the identity matrix added to H to give a positive definite matrix. (Output)

SOL — Vector of length NVAR containing solution. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector of length NVAR containing the indices of the final active constraints in the first NACT positions. (Output)

ALAMDA — Vector of length NVAR containing the Lagrange multiplier estimates of the final active constraints in the first NACT positions. (Output)

Comments

1. Automatic workspace usage is

   QPROG (3 * NVAR**2 + 11 * NVAR)/2 + NCON units, or
   DQPROG (3 * NVAR**2 + 11 * NVAR) + 2 * NCON units.

   Workspace may be explicitly provided, if desired, by use of Q2ROG/DQ2ROG. The reference is

   ```
   CALL Q2ROG (NVAR, NCON, NEQ, A, LDA, B, G, H, LDH,
               DIAG, SOL, NACT, IACT, ALAMDA, WK)
   ```

   The additional argument is

   WK — Work vector of length (3 * NVAR**2 + 11 * NVAR)/2 + NCON.

2. Informational errors

Type	Code	
3	1	Due to the effect of computer rounding error, a change in the variables fail to improve the objective function value; usually the solution is close to optimum.
4	2	The system of equations is inconsistent. There is no solution.

3. If a perturbation of H, H + DIAG * I, was used in the QP problem, then H + DIAG * I should also be used in the definition of the Lagrange multipliers.

Algorithm

The routine QPROG is based on M.J.D. Powell's implementation of the Goldfarb and Idnani (1983) dual quadratic programming (QP) algorithm for convex QP problems subject to general linear equality/inequality constraints, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} g^T x + \frac{1}{2} x^T H x$$

$$\text{subject to} \quad A_1 x = b_1$$

$$A_2 x \geq b_2$$

given the vectors b_1, b_2, and g and the matrices H, A_1, and A_2. H is required to be positive definite. In this case, a unique x solves the problem or the constraints are inconsistent. If H is not positive definite, a positive definite perturbation of H is used in place of H. For more details, see Powell (1983, 1985).

Example

A quadratic programming problem is solved.

```
C                                 Declare variables
      INTEGER    LDA, LDH, NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=2, NVAR=5, LDA=NCON, LDH=NVAR)
C
      INTEGER    IACT(NVAR), K, NACT, NOUT
      REAL       A(LDA,NVAR), ALAMDA(NVAR), B(NCON), DIAG, G(NVAR),
     &           H(LDH,LDH), SOL(NVAR)
      EXTERNAL   QPROG, UMACH
C
C                                 Set values of A, B, G and H.
C                                 A = ( 1.0  1.0  1.0  1.0  1.0)
C                                     ( 0.0  0.0  1.0 -2.0 -2.0)
C
C                                 B = ( 5.0 -3.0)
C
C                                 G = (-2.0  0.0  0.0  0.0  0.0)
C
C                                 H = ( 2.0  0.0  0.0  0.0  0.0)
C                                     ( 0.0  2.0 -2.0  0.0  0.0)
C                                     ( 0.0 -2.0  2.0  0.0  0.0)
C                                     ( 0.0  0.0  0.0  2.0 -2.0)
C                                     ( 0.0  0.0  0.0 -2.0  2.0)
C
      DATA A/1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, -2.0, 1.0, -2.0/
      DATA B/5.0, -3.0/
      DATA G/-2.0, 4*0.0/
      DATA H/2.0, 5*0.0, 2.0, -2.0, 3*0.0, -2.0, 2.0, 5*0.0, 2.0,
     &     -2.0, 3*0.0, -2.0, 2.0/
C
      CALL QPROG (NVAR, NCON, NEQ, A, LDA, B, G, H, LDH, DIAG, SOL,
     &            NACT, IACT, ALAMDA)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (SOL(K),K=1,NVAR)
99999 FORMAT (’  The solution vector is’, /, ’  SOL = (’, 5F6.1,
     &       ’  )’)
C
      END
```

Output

```
The solution vector is
SOL = (   1.0   1.0   1.0   1.0   1.0  )
```

LCONF/DLCONF (Single/Double precision)

Minimize a general objective function subject to linear equality/inequality constraints.

Usage

```
CALL LCONF (FCN, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB,
            XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT,
            ALAMDA)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (N, X, F), where

N – Value of NVAR. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
F – The computed function value at the point X. (Output)

FCN must be declared EXTERNAL in the calling program.

NVAR — The number of variables. (Input)

NCON — The number of linear constraints (excluding simple bounds). (Input)

NEQ — The number of linear equality constraints. (Input)

A — NCON by NVAR matrix. (Input)
The matrix contains the equality constraint gradients in the first NEQ rows, followed by the inequality constraint gradients.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length NCON containing right-hand sides of the linear constraints. (Input)
Specifically, the constraints on the variables X(I), I = 1, …, NVAR are A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).EQ.B(K), K = 1, …, NEQ.A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).LE.B(K), K = NEQ + 1, …, NCON. Note that the data that define the equality constraints come before the data of the inequalities.

XLB — Vector of length NVAR containing the lower bounds on the variables; choose a very large negative value if a component should be unbounded below or set XLB(I) = XUB(I) to freeze the I-th variable. (Input)
Specifically, these simple bounds are XLB(I).LE.X(I), I = 1, …, NVAR.

XUB — Vector of length NVAR containing the upper bounds on the variables; choose a very large positive value if a component should be unbounded above. (Input)
Specifically, these simple bounds are X(I).LE.XUB(I), I = 1, …, NVAR.

XGUESS — Vector of length NVAR containing the initial guess of the minimum. (Input)

ACC — The nonnegative tolerance on the first order conditions at the calculated solution. (Input)

MAXFCN — On input, maximum number of function evaluations allowed. (Input/ Output)
On output, actual number of function evaluations needed.

SOL — Vector of length NVAR containing solution. (Output)

OBJ — Value of the objective function. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector containing the indices of the final active constraints in the first NACT positions. (Output)
Its length must be at least NCON + 2 * NVAR.

ALAMDA — Vector of length NVAR containing the Lagrange multiplier estimates of the final active constraints in the first NACT positions. (Output)

Comments

1. Automatic workspace usage is

   LCONF NVAR**2 + 11 * NVAR + NCON units, or
   DLCONF 2 * (NVAR**2 + 11* NVAR + NCON) units.

   Workspace may be explicitly provided, if desired, by use of L2ONF/DL2ONF. The reference is

```
CALL L2ONF (FCN, NVAR, NCON, NEQ, A, LDA, B, XLB,
            XUB, XGUESS, ACC, MAXFCN, SOL, OBJ,
            NACT, IACT, ALAMDA, IPRINT, INFO, WK)
```

   The additional arguments are as follows:

   IPRINT — Print option (see Comment 3). (Input)

   INFO — Informational flag (see Comment 3). (Output)

   WK — Real work vector of length NVAR**2 + 11 * NVAR + NCON.

2. Informational errors

Type	Code	
4	4	The equality constraints are inconsistent.
4	5	The equality constraints and the bounds on the variables are found to be inconsistent.
4	6	No vector X satisfies all of the constraints. In particular, the current active constraints prevent any change in X that reduces the sum of constraint violations.
4	7	Maximum number of function evaluations exceeded.
4	9	The variables are determined by the equality constraints.

3. The following are descriptions of the arguments IPRINT and INFO:

   IPRINT — This argument must be set by the user to specify the frequency of printing during the execution of the routine LCONF. There is no printed output if IPRINT = 0. Otherwise, after ensuring feasibility, information is given every IABS(IPRINT) iterations and whenever a parameter called TOL is reduced. The printing provides the values of X(.), F(.) and G(.) = GRAD(F) if IPRINT is positive. If IPRINT is negative, this information is augmented by the current values of IACT(K) K = 1, …, NACT, PAR(K) K = 1, …, NACT and RESKT(I) I = 1,

…, N. The reason for returning to the calling program is also displayed when IPRINT is nonzero.

INFO — On exit from L2ONF, INFO will have one of the following integer values to indicate the reason for leaving the routine:

INFO = 1	SOL is feasible, and the condition that depends on ACC is satisfied.
INFO = 2	SOL is feasible, and rounding errors are preventing further progress.
INFO = 3	SOL is feasible, but the objective function fails to decrease although a decrease is predicted by the current gradient vector.
INFO = 4	In this case, the calculation cannot begin because LDA is less than NCON or because the lower bound on a variable is greater than the upper bound.
INFO = 5	This value indicates that the equality constraints are inconsistent. These constraints include any components of X(.) that are frozen by setting XL(I) = XU(I).
INFO = 6	In this case there is an error return because the equality constraints and the bounds on the variables are found to be inconsistent.
INFO = 7	This value indicates that there is no vector of variables that satisfies all of the constraints. Specifically, when this return or an INFO = 6 return occurs, the current active constraints (whose indices are IACT(K), K = 1, …, NACT) prevent any change in X(.) that reduces the sum of constraint violations. Bounds are only included in this sum if INFO = 6.
INFO = 8	Maximum number of function evaluations exceeded.
INFO = 9	The variables are determined by the equality constraints.

Algorithm

The routine LCONF is based on M.J.D. Powell’s TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \qquad A_1 x = b_1$$

$$A_2 x \le b_2$$

$$x_l \le x \le x_u$$

given the vectors b_1, b_2, x_l and x_u and the matrices A_1, and A_2.

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise x^0, the initial guess provided by the user, to satisfy

$$A_1 x = b_1$$

Next, x^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible x^k, let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem

$$\min f\left(x^k\right) + d^T \nabla f\left(x^k\right) + \frac{1}{2} d^T B^k d$$

$$\text{subject to} \qquad a_j d = 0 \quad j \in I_k$$

$$a_j d \le 0 \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on x. In the latter case, the $a_j = e_i$ for the bound constraint $x_i \le (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \le (-x_l)_i$. Here, e_i is a vector with a 1 as the i-th component, and zeroes elsewhere. λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions

$$f\left(x^k + \alpha^k d^k\right) \le f\left(x^k\right) + 0.1\alpha^k \left(d^k\right)^T \nabla f\left(x^k\right)$$

and

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) \ge 0.7\left(d^k\right)^T \nabla f\left(x^k\right)$$

The main idea in forming the set J_k is that, if any of the inequality constraints restricts the step-length α^k, then its index is not in J_k. Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation, B^k , is updated by the BFGS formula, if the condition

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) - \nabla f\left(x^k\right) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\left\| \nabla f\left(x^k\right) - A^k \lambda^k \right\|_2 \le \tau$$

is satisfied; here, τ is a user-supplied tolerance. For more details, see Powell (1988, 1989).

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine `LCONG` (page 990) should be used instead.

Example

The problem from Schittkowski (1987)

$$\min f(x) = -x_1 x_2 x_3$$

$$\begin{aligned} \text{subject to} \quad & -x_1 - 2x_2 - 2x_3 \le 0 \\ & x_1 + 2x_2 + 2x_3 \le 72 \\ & 0 \le x_1 \le 20 \\ & 0 \le x_2 \le 11 \\ & 0 \le x_3 \le 42 \end{aligned}$$

is solved with an initial guess $x_1 = 10$, $x_2 = 10$ and $x_3 = 10$.

```
C                                  Declaration of variables
      INTEGER    LDA, NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=0, NVAR=3, LDA=NCON)
C
      INTEGER    IACT(8), MAXFCN, NACT, NOUT
      REAL       A(NCON,NVAR), ACC, ALAMDA(NVAR), B(NCON), OBJ,
     &           SOL(NVAR), XGUESS(NVAR), XLB(NVAR), XUB(NVAR)
      EXTERNAL   FCN, LCONF, UMACH
C
C                                  Set values for the following problem.
C
C                                  Min  -X(1)*X(2)*X(3)
C
C                                  -X(1) - 2*X(2) - 2*X(3)  .LE.   0
C                                   X(1) + 2*X(2) + 2*X(3)  .LE.  72
C
C                                  0  .LE.  X(1)  .LE.  20
C                                  0  .LE.  X(2)  .LE.  11
C                                  0  .LE.  X(3)  .LE.  42
C
      DATA A/-1.0, 1.0, -2.0, 2.0, -2.0, 2.0/, B/0.0, 72.0/
      DATA XLB/3*0.0/, XUB/20.0, 11.0, 42.0/, XGUESS/3*10.0/
      DATA ACC/0.0/, MAXFCN/400/
```

```
C
      CALL UMACH (2, NOUT)
C
      CALL LCONF (FCN, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB, XGUESS,
     &            ACC, MAXFCN, SOL, OBJ, NACT, IACT, ALAMDA)
C
      WRITE (NOUT,99998) 'Solution:'
      WRITE (NOUT,99999) SOL
      WRITE (NOUT,99998) 'Function value at solution:'
      WRITE (NOUT,99999) OBJ
      WRITE (NOUT,99998) 'Number of function evaluations:', MAXFCN
      STOP
99998 FORMAT (//, ' ', A, I4)
99999 FORMAT (1X, 5F16.6)
      END
C
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(*), F
C
      F = -X(1)*X(2)*X(3)
      RETURN
      END
```

Output

```
Solution:
 20.000000       11.000000       15.000000

Function value at solution:
-3300.000000

Number of function evaluations:   5
```

LCONG/DLCONG (Single/Double precision)

Minimize a general objective function subject to linear equality/inequality constraints.

Usage

```
CALL LCONG (FCN, GRAD, NVAR, NCON, NEQ, A, LDA, B, XLB,
            XUB, XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT,
            ALAMDA)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Value of `NVAR`. (Input)
`X` – Vector of length `N` at which point the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied SUBROUTINE to compute the gradient at the point X. The usage is CALL GRAD (N, X, G), where

N – Value of NVAR. (Input)
X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by GRAD.
G – Vector of length N containing the values of the gradient of the objective function evaluated at the point X. (Output)

GRAD must be declared EXTERNAL in the calling program.

NVAR — The number of variables. (Input)

NCON — The number of linear constraints (excluding simple bounds). (Input)

NEQ — The number of linear equality constraints. (Input)

A — NCON by NVAR matrix. (Input)
The matrix contains the equality constraint gradients in the first NEQ rows, followed by the inequality constraint gradients.

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

B — Vector of length NCON containing right-hand sides of the linear constraints. (Input)
Specifically, the constraints on the variables X(I), I = 1, …, NVAR are A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).EQ.B(K), K = 1, …, NEQ.A(K, 1) * X(1) + … + A(K, NVAR) * X(NVAR).LE.B(K), K = NEQ + 1, …, NCON. Note that the data that define the equality constraints come before the data of the inequalities.

XLB — Vector of length NVAR containing the lower bounds on the variables; choose a very large negative value if a component should be unbounded below or set XLB(I) = XUB(I) to freeze the I-th variable. (Input)
Specifically, these simple bounds are XLB(I).LE.X(I), I = 1, …, NVAR.

XUB — Vector of length NVAR containing the upper bounds on the variables; choose a very large positive value if a component should be unbounded above. (Input)
Specifically, these simple bounds are X(I).LE. XUB(I), I = 1, …, NVAR.

XGUESS — Vector of length NVAR containing the initial guess of the minimum. (Input)

ACC — The nonnegative tolerance on the first order conditions at the calculated solution. (Input)

MAXFCN — On input, maximum number of function evaluations allowed.(Input/ Output)
On output, actual number of function evaluations needed.

SOL — Vector of length NVAR containing solution. (Output)

OBJ — Value of the objective function. (Output)

NACT — Final number of active constraints. (Output)

IACT — Vector containing the indices of the final active constraints in the first NACT positions. (Output)
Its length must be at least NCON + 2 * NVAR.

ALAMDA — Vector of length NVAR containing the Lagrange multiplier estimates of the final active constraints in the first NACT positions. (Output)

Comments

1. Automatic workspace usage is

   LCONG NVAR**2 + 11 * NVAR + NCON units, or
   DLCONG 2 * (NVAR**2 + 11 * NVAR + NCON) units.

   Workspace may be explicitly provided, if desired, by use of L2ONG/DL2ONG. The reference is

   ```
   CALL L2ONG (FCN, GRAD, NVAR, NCON, NEQ, A, LDA, B,
               XLB, XUB, XGUESS, ACC, MAXFCN, SOL, OBJ,
               NACT, IACT, ALAMDA, IPRINT, INFO, WK)
   ```

   The additional arguments are as follows:

   IPRINT — Print option (see Comment 3). (Input)

   INFO — Informational flag (see Comment 3). (Output)

   WK — Real work vector of length NVAR**2 + 11 * NVAR + NCON.

2. Informational errors

Type	Code	
4	4	The equality constraints are inconsistent.
4	5	The equality constraints and the bounds on the variables are found to be inconsistent.
4	6	No vector X satisfies all of the constraints. In particular, the current active constraints prevent any change in X that reduces the sum of constraint violations.
4	7	Maximum number of function evaluations exceeded.
4	9	The variables are determined by the equality constraints.

3. The following are descriptions of the arguments IPRINT and INFO:

   IPRINT — This argument must be set by the user to specify the frequency of printing during the execution of the routine LCONG. There is no printed output if IPRINT = 0. Otherwise, after ensuring feasibility, information is given every IABS(IPRINT) iterations and whenever a parameter called TOL is reduced. The printing provides the values of X(.), F(.) and G(.) = GRAD(F) if IPRINT is positive. If IPRINT is negative, this information is augmented by the current values of IACT(K) K = 1, …, NACT, PAR(K) K = 1, …, NACT and RESKT(I) I = 1, …, N. The reason for returning to the calling program is also displayed when IPRINT is nonzero.

INFO — On exit from L2ONG, INFO will have one of the following integer values to indicate the reason for leaving the routine:

INFO = 1 SOL is feasible and the condition that depends on ACC is satisfied.

INFO = 2 SOL is feasible and rounding errors are preventing further progress.

INFO = 3 SOL is feasible but the objective function fails to decrease although a decrease is predicted by the current gradient vector.

INFO = 4 In this case, the calculation cannot begin because LDA is less than NCON or because the lower bound on a variable is greater than the upper bound.

INFO = 5 This value indicates that the equality constraints are inconsistent. These constraints include any components of X(.) that are frozen by setting XL(I) = XU(I).

INFO = 6 In this case, there is an error return because the equality constraints and the bounds on the variables are found to be inconsistent.

INFO = 7 This value indicates that there is no vector of variables that satisfies all of the constraints. Specifically, when this return or an INFO = 6 return occurs, the current active constraints (whose indices are IACT(K), K = 1, …, NACT) prevent any change in X(.) that reduces the sum of constraint violations, where only bounds are included in this sum if INFO = 6.

INFO = 8 Maximum number of function evaluations exceeded.

INFO = 9 The variables are determined by the equality constraints.

Algorithm

The routine LCONG is based on M.J.D. Powell's TOLMIN, which solves linearly constrained optimization problems, i.e., problems of the form

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \quad A_1 x = b_1$$

$$A_2 x \le b_2$$

$$x_l \le x \le x_u$$

given the vectors b_1, b_2, x_l and x_u and the matrices A_1, and A_2.

The algorithm starts by checking the equality constraints for inconsistency and redundancy. If the equality constraints are consistent, the method will revise x^0, the initial guess provided by the user, to satisfy

$$A_1 x = b_1$$

Next, x^0 is adjusted to satisfy the simple bounds and inequality constraints. This is done by solving a sequence of quadratic programming subproblems to minimize the sum of the constraint or bound violations.

Now, for each iteration with a feasible x_k, let J_k be the set of indices of inequality constraints that have small residuals. Here, the simple bounds are treated as inequality constraints. Let I_k be the set of indices of active constraints. The following quadratic programming problem

$$\min f\left(x^k\right) + d^T \nabla f\left(x^k\right) + \frac{1}{2} d^T B^k d$$

$$\text{subject to} \qquad a_j d = 0 \quad j \in I_k$$

$$a_j d \le 0 \quad j \in J_k$$

is solved to get (d^k, λ^k) where a_j is a row vector representing either a constraint in A_1 or A_2 or a bound constraint on x. In the latter case, the $a_j = e_i$ for the bound constraint $x_i \le (x_u)_i$ and $a_j = -e_i$ for the constraint $-x_i \le (-x_l)_i$. Here, e_i is a vector with a 1 as the i-th component, and zeroes elsewhere. λ^k are the Lagrange multipliers, and B^k is a positive definite approximation to the second derivative $\nabla^2 f(x^k)$.

After the search direction d^k is obtained, a line search is performed to locate a better point. The new point $x^{k+1} = x^k + \alpha^k d^k$ has to satisfy the conditions

$$f\left(x^k + \alpha^k d^k\right) \le f\left(x^k\right) + 0.1\alpha^k \left(d^k\right)^T \nabla f\left(x^k\right)$$

and

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) \ge 0.7\left(d^k\right)^T \nabla f\left(x^k\right)$$

The main idea in forming the set J_k is that, if any of the inequality constraints restricts the step-length α^k, then its index is not in J_k. Therefore, small steps are likely to be avoided.

Finally, the second derivative approximation, B^k, is updated by the BFGS formula, if the condition

$$\left(d^k\right)^T \nabla f\left(x^k + \alpha^k d^k\right) - \nabla f\left(x^k\right) > 0$$

holds. Let $x^k \leftarrow x^{k+1}$, and start another iteration.

The iteration repeats until the stopping criterion

$$\left\| \nabla f\left(x^k\right) - A^k \lambda^k \right\|_2 \le \tau$$

is satisfied; here, τ is a user-supplied tolerance. For more details, see Powell (1988, 1989).

Example

The problem from Schittkowski (1987)

$$\min f(x) = -x_1 x_2 x_3$$

$$\text{subject to} \qquad -x_1 - 2x_2 - 2x_3 \le 0$$

$$x_1 + 2x_2 + 2x_3 \le 72$$

$$0 \le x_1 \le 20$$

$$0 \le x_2 \le 11$$

$$0 \le x_3 \le 42$$

is solved with an initial guess $x_1 = 10$, $x_2 = 10$ and $x_3 = 10$.

```
C                                 Declaration of variables
      INTEGER    LDA, NCON, NEQ, NVAR
      PARAMETER  (NCON=2, NEQ=0, NVAR=3, LDA=NCON)
C
      INTEGER    IACT(8), MAXFCN, NACT, NOUT
      REAL       A(NCON,NVAR), ACC, ALAMDA(NVAR), B(NCON), OBJ,
     &           SOL(NVAR), XGUESS(NVAR), XLB(NVAR), XUB(NVAR)
      EXTERNAL   FCN, GRAD, LCONG, UMACH
C
C                                 Set values for the following problem.
C
C                                 Min  -X(1)*X(2)*X(3)
C
C                                 -X(1) - 2*X(2) - 2*X(3)  .LE.   0
C                                  X(1) + 2*X(2) + 2*X(3)  .LE.  72
C
C                                 0  .LE.  X(1)  .LE.  20
C                                 0  .LE.  X(2)  .LE.  11
C                                 0  .LE.  X(3)  .LE.  42
C
      DATA A/-1.0, 1.0, -2.0, 2.0, -2.0, 2.0/, B/0.0, 72.0/
      DATA XLB/3*0.0/, XUB/20.0, 11.0, 42.0/, XGUESS/3*10.0/
      DATA ACC/0.0/, MAXFCN/400/
C
      CALL UMACH (2, NOUT)
C
      CALL LCONG (FCN, GRAD, NVAR, NCON, NEQ, A, LDA, B, XLB, XUB,
     &            XGUESS, ACC, MAXFCN, SOL, OBJ, NACT, IACT, ALAMDA)
C
      WRITE (NOUT,99998) ’Solution:’
      WRITE (NOUT,99999) SOL
      WRITE (NOUT,99998) ’Function value at solution:’
      WRITE (NOUT,99999) OBJ
      WRITE (NOUT,99998) ’Number of function evaluations:’, MAXFCN
      STOP
99998 FORMAT (//, ' ', A, I4)
99999 FORMAT (1X, 5F16.6)
      END
C
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(*), F
C
      F = -X(1)*X(2)*X(3)
      RETURN
      END
C
      SUBROUTINE GRAD (N, X, G)
      INTEGER    N
      REAL       X(*), G(*)
C
      G(1) = -X(2)*X(3)
      G(2) = -X(1)*X(3)
      G(3) = -X(1)*X(2)
      RETURN
      END
```

Output

```
Solution:
20.000000       11.000000       15.000000

Function value at solution:
-3300.000000

Number of function evaluations:   5
```

NCONF/DNCONF (Single/Double precision)

Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a finite difference gradient.

Usage

```
CALL NCONF (FCN, M, ME, N, XGUESS, IBTYPE, XLB, XUB,
            XSCALE, IPRINT, MAXITN, X, FVALUE)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the functions at a given point. The usage is CALL FCN (M, ME, N, X, ACTIVE, F, G), where

M – Total number of constraints. (Input)
ME – Number of equality constraints. (Input)
N – Number of variables. (Input)
X – The point at which the functions are evaluated. (Input)
X should not be changed by FCN.
ACTIVE – Logical vector of length MMAX indicating the active constraints. (Input)
MMAX = MAX(1, M)

F – The computed function value at the point X. (Output)
G – Vector of length MMAX containing the values of constraints at point X. (Output)

FCN must be declared EXTERNAL in the calling program.

M — Total number of constraints. (Input)

ME — Number of equality constraints. (Input)

N — Number of variables. (Input)

XGUESS — Vector of length N containing an initial guess of the computed solution. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable; all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)
If there is no lower bound for a variable, then the corresponding XLB value should be set to −1.0E6.

XUB — Vector of length N containing the upper bounds on variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3)
If there is no upper bound for a variable, then the corresponding XLB value should be set to 1.0E6.

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables. (Input)
All values of XSCALE must be greater than zero. In the absence of other information, set all entries to 1.0.

IPRINT — Parameter indicating the desired output level. (Input)

IPRINT	**Action**
0	No output printed.
1	Only a final convergence analysis is given.
2	One line of intermediate results are printed in each iteration.
3	Detailed information is printed in each iteration.

MAXITN — Maximum number of iterations allowed. (Input)

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the objective function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   NCONF N * (3 * N + 38 + MMAX) + 7 * MMAX + 6 * M + MAX(N, M) + 91 units, or

   DNCONF 2 * N * (3 * N + 38 + MMAX) + 14 * MMAX + 12 * M + MAX(N, M) + 163 units.
   MMAX = MAX(1, M)

   Workspace may be explicitly provided, if desired, by use of N2ONF/DN2ONF. The reference is

```
CALL N2ONF (FCN, M, ME, N, XGUESS, IBTYPE, XLB, XUB,
            XSCALE, IPRINT, MAXITN, X, FVALUE, WK,
            LWK, IWK, LIWK, CONWK)
```

   The additional arguments are as follows:

   WK — Work vector of length N * (3 * N + 38 + MMAX) + 6 * MMAX + 6 * M + 72

   LWK — Length of WK.

   IWK — Work vector of length 19 + MAX(M, N).

   LIWK — Length of LIWK.

   CONWK — Work vector of length MMAX.

2. Informational errors

Type	Code	
4	1	Search direction uphill.
4	2	Line search took more than 5 function calls.
4	3	Maximum number of iterations exceeded.
4	4	Search direction is close to zero.
4	5	The constraints for the QP subproblem are inconsistent.

3. If reverse communication is desired, then N0ONF/DN0ONF can be used rather than using an external subroutine to evaluate the function. The reference is

```
CALL N0ONF (IDO, M, ME, N, IBTYPE, XLB, XUB, IPRINT,
            MAXITN, X, FVALUE, G, DF, DG, LDDG, U,
            C, LDC, D, ACC, SCBOU, MAXFUN, ACTIVE,
            MODE, WK, IWK, CONWK)
```

   The additional arguments are as follows:

   IDO — Reverse communication parameter indicating task to be done. (Input/Output)

   On the initial call, IDO must be set to 0; and initial values must be passed into N0ONF/DN0ONF for X, FVALUE, G, DF, and DG. If the routine returns with IDO = 1, then the user has to compute FVALUE and G with respect to X. If the routine returns with IDO = 2, then the user has to compute DF and DG with respect to X. The user has to call the routine repeatedly until IDO does not equal to 1 or 2.

X — Vector of length N containing the initial guesses to the solution on input and the solution on output. (Input/Output)

FVALUE — Scalar containing the objective function value evaluated at the current X. (Input)

G — Vector of length MMAX containing constraint values at the current X. (Input)
MMAX is MAX(1, M).

DF — Vector of length N containing the gradient of the objective function evaluated at the current X. (Input)

DG — Array of dimension MMAX by N containing the gradient of the constraints evaluated at the current X. (Input)

LDDG — Leading dimension of DG exactly as specified in the dimension statement of the calling program. (Input)

U — Vector of length M + N + N + 2 containing the multipliers of the nonlinear constraints and of the bounds. (Output)
The first M locations contain the multipliers for the nonlinear constraints. The second N locations contain the multipliers for the lower bounds. The third N locations contain the multipliers for the upper bounds.

C — Array of dimension N + 1 by N + 1 containing the final approximation to the Hessian. (Output)

LDC — Leading dimension of C exactly as specified in the dimension statement of the calling program. (Input)

D — Vector of length N + 1 containing the diagonal elements of the Hessian. (Output)

ACC — Final accuracy. (Input)

SCBOU — Scalar containing the scaling variable for the problem function. (Input)
In the absence of further information, SCBOU may be set to 1.0E3.

MAXFUN — Scalar containing the maximum allowable function calls during the line search. (Input)

ACTIVE — Logical vector of length 2 * MMAX + 13. (Input/Output)
The first MMAX locations are used to determine which gradient constraints are active and must be set to .TRUE. on input. If ACTIVE(I) is .TRUE., then DG(I, K) is evaluated for K = 1, N. The last MMAX + 13 locations are used for workspace.

MODE — Desired solving version for the algorithm.   (Input)
If `MODE` = 2; then reverse communication is used. If `MODE` = 3; then reverse communication is used and initial guesses for the multipliers and Hessian matrix of the Lagrange function are provided on input.

WK — Work vector of length 2 `*` `N` `*` (`N` + 16) + 4 `*` `MMAX` + 5 `*` `M` + 68.

IWK — Work vector of length 19 + `MAX`(`M`, `N`).

CONWK — Work vector of length `M`.

Algorithm

The routine `NCONF` is based on subroutine `NLPQL`, a FORTRAN code developed by Schittkowski (1986). It uses a successive quadratic programming method to solve the general nonlinear programming problem. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\begin{aligned} \text{subject to} \qquad & g_j(x) = 0, \text{ for } \quad j = 1, \ldots, m_e \\ & g_j(x) \ge 0, \text{ for } \quad j = m_e + 1, \ldots, m \\ & x_l \le x \le x_u \end{aligned}$$

where all problem functions are assumed to be continuously differentiable. The method, based on the iterative formulation and solution of quadratic programming subproblems, obtains subproblems by using a quadratic approximation of the Lagrangian and by linearizing the constraints. That is,

$$\min_{d \in \mathbf{R}^n} \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d$$

$$\begin{aligned} \text{subject to} \qquad & \nabla g_j(x_k)^T d + g_j(x_k) = 0, \qquad && j = 1, \ldots, m_e \\ & \nabla g_j(x_k)^T d + g_j(x_k) \ge 0, \qquad && j = m_e + 1, \ldots, m \\ & x_l - x_k \le d \le x_u - x_k \end{aligned}$$

where B_k is a positive definite approximation of the Hessian and x_k is the current iterate. Let d_k be the solution of the subproblem. A line search is used to find a new point x_{k+1},

$$x_{k+1} = x_k + \lambda d_k, \quad \lambda \in (0, 1]$$

such that a “merit function” will have a lower function value at the new point. Here, the augmented Lagrange function (Schittkowski 1986) is used as the merit function.

When optimality is not achieved, B_k is updated according to the modified BFGS formula (Powell 1978). Note that this algorithm may generate infeasible points during the solution process. Therefore, if feasibility must be maintained for intermediate points, then this routine may not be suitable. For more theoretical and practical details, see Stoer (1985), Schittkowski (1983, 1986) and Gill et al. (1985).

Since a finite-difference method is used to estimate the gradient for some single precision calculations, an inaccurate estimate of the gradient may cause the algorithm to terminate at a noncritical point. In such cases, high precision arithmetic is recommended. Also, whenever the exact gradient can be easily provided, routine NCONG (page 1003) should be used instead.

Within NCONF, there is a user-callable subroutine N0ONF that gives the user the option to use “reverse communication.” This option allows the user to evaluate the functions and gradients in the main program. This option is useful when it is difficult to do the function evaluation in the fixed form required by NCONF.

Example 1

The problem

$$\min F(x) = \left(x_1 - 2\right)^2 + \left(x_2 - 1\right)^2$$

$$\text{subject to} \quad g_1(x) = x_1 - 2x_2 + 1 = 0$$

$$g_2(x) = -\left(x_1^2\right) / 4 - x_2^2 + 1 \geq 0$$

is solved with an initial guess (2.0, 2.0).

```
      INTEGER    IBTYPE, IPRINT, M, MAXITN, ME, N
      PARAMETER  (IBTYPE=0, IPRINT=0, M=2, MAXITN=100, ME=1, N=2)
C
      REAL       FVALUE, X(N), XGUESS(N), XLB(N), XSCALE(N), XUB(N)
      EXTERNAL   FCN, NCONF, WRRRN
C
      DATA XGUESS/2.0E0, 2.0E0/, XSCALE/2*1.0E0/
      DATA XLB/-1.0E6, -1.0E6/, XUB/1.0E6, 1.0E6/
C
      CALL NCONF (FCN, M, ME, N, XGUESS, IBTYPE, XLB, XUB, XSCALE,
     &            IPRINT, MAXITN, X, FVALUE)
C
      CALL WRRRN (’The solution is’, N, 1, X, N, 0)
      END
C
      SUBROUTINE FCN (M, ME, N, X, ACTIVE, F, G)
      INTEGER    M, ME, N
      REAL       X(*), F, G(*)
      LOGICAL    ACTIVE(*)
C                                  Himmelblau problem 1
      F = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
C
      IF (ACTIVE(1)) G(1) = X(1) - 2.0E0*X(2) + 1.0E0
      IF (ACTIVE(2)) G(2) = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      RETURN
      END
```

Output

```
The solution is
 1   0.8229
 2   0.9114
```

Example 2

This example uses the reverse communication option to solve the same problem as Example 1.

```
      INTEGER    LDC, LDDG, LWK, M, ME, N
      PARAMETER  (M=2, ME=1, N=2, LDC=N+1, LDDG=M,
     &           LWK=2*N*(N+16)+9*M+68)
C
      INTEGER    IBTYPE, IDO, IPRINT, IWK(19+M), MAXFUN, MAXITN,
     &           MODE
      REAL       ACC, AMACH, C(LDC,N+1), CONWK(M), D(N+1), DF(N),
     &           DG(LDDG,N), FVALUE, G(M), SCBOU, SQRT,
     &           U(M+N+N+2), WK(LWK), X(N), XLB(N), XUB(N)
      LOGICAL    ACTIVE(2*M+13)
      INTRINSIC  SQRT
      EXTERNAL   AMACH, N0ONF
C
      DATA IBTYPE/3/, MAXITN/100/, MODE/2/, MAXFUN/10/, IPRINT/0/
      DATA X/2.0E0, 2.0E0/, XLB(1)/-1.0E6/, XUB(1)/1.0E6/, SCBOU/1.0E3/
C                                 Set final accuracy (ACC)
      ACC = SQRT(AMACH(4))
C
      ACTIVE(1) = .TRUE.
      ACTIVE(2) = .TRUE.
      IDO       = 0
   10 IF (IDO.EQ.0 .OR. IDO.EQ.1) THEN
C                                 Evaluate the function at X.
         FVALUE = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
C                                 Evaluate the constraints at X.
         G(1) = X(1) - 2.0E0*X(2) + 1.0E0
         G(2) = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      END IF
C
      IF (IDO.EQ.0 .OR. IDO.EQ.2) THEN
C                                 Evaluate the function gradient at X.
         DF(1) = 2.0E0*(X(1)-2.0E0)
         DF(2) = 2.0E0*(X(2)-1.0E0)
C                                 If active evaluate the constraint
C                                 gradient at X.
         IF (ACTIVE(1)) THEN
            DG(1,1) = 1.0E0
            DG(1,2) = -2.0E0
         END IF
C
         IF (ACTIVE(2)) THEN
            DG(2,1) = -0.5E0*X(1)
            DG(2,2) = -2.0E0*X(2)
         END IF
      END IF
C                                Call N0ONF for the next update.
```

```
C
      CALL N0ONF (IDO, M, ME, N, IBTYPE, XLB, XUB, IPRINT, MAXITN, X,
     &            FVALUE, G, DF, DG, LDDG, U, C, LDC, D, ACC, SCBOU,
     &            MAXFUN, ACTIVE, MODE, WK, IWK, CONWK)
C                                If IDO does not equal 1 or 2, exit.
      IF (IDO.EQ.1 .OR. IDO.EQ.2) GO TO 10
C                                 Print the solution
      CALL WRRRN ('The solution is', N, 1, X, N, 0)
C
      END
```

Output

```
The solution is
 1   0.8229
 2   0.9114
```

NCONG/DNCONG (Single/Double precision)

Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a user-supplied gradient.

Usage

```
CALL NCONG (FCN, GRAD, M, ME, N, XGUESS, IBTYPE, XLB, XUB,
            IPRINT, MAXITN, X, FVALUE)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the functions at a given point. The usage is `CALL FCN (M, ME, N, X, ACTIVE, F, G)`, where

`M` – Total number of constraints. (Input)
`ME` – Number of equality constraints. (Input)
`N` – Number of variables. (Input)
`X` – The point at which the functions are evaluated. (Input)
`X` should not be changed by `FCN`.
`ACTIVE` – Logical vector of length `MMAX` indicating the active constraints. (Input)
`F` – The computed function value at the point `X`. (Output)
`G` – Vector of length `MMAX` containing the values of constraints at point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — User-supplied `SUBROUTINE` to evaluate the gradients at a given point. The usage is `CALL GRAD (M, ME, MMAX, N, X, ACTIVE, F, G, DF, DG)`, where

`M` – Total number of constraints. (Input)
`ME` – Number of equality constraints. (Input)
`MMAX` – Maximum of (1, `M`). (Input)
`N` – Number of variables. (Input)

X – Vector of length N at which point the function is evaluated. (Input)
X should not be changed by FCN.
ACTIVE – Logical vector of length MMAX indicating the active constraints. (Input)
F – The computed function value at the point X. (Input)
G – Vector of length MMAX containing the values of the constraints at point X. (Input)
DF – Vector of length N containing the values of the gradient of the objective function. (Output)
DG – MMAX by N array containing the values of the gradients for the active constraints. (Output)

GRAD must be declared EXTERNAL in the calling program.

M — Total number of constraints. (Input)

ME — Number of equality constraints. (Input)

N — Number of variables. (Input)

XGUESS — Vector of length N containing an initial guess of the computed solution. (Input)

IBTYPE — Scalar indicating the types of bounds on variables. (Input)

IBTYPE	**Action**
0	User will supply all the bounds.
1	All variables are nonnegative.
2	All variables are nonpositive.
3	User supplies only the bounds on 1st variable, all other variables will have the same bounds.

XLB — Vector of length N containing the lower bounds on the variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3) If there is no lower bound on a variable, then the corresponding XLB value should be set to −1.0E6.

XUB — Vector of length N containing the upper bounds on the variables. (Input, if IBTYPE = 0; output, if IBTYPE = 1 or 2; input/output, if IBTYPE = 3) If there is no upper bound on a variable, then the corresponding XUB value should be set to 1.0E6.

IPRINT — Parameter indicating the desired output level. (Input)

IPRINT	**Action**
0	No output printed.
1	Only a final convergence analysis is given.
2	One line of intermediate results is printed for each iteration.
3	Detailed information is printed for each iteration.

MAXITN — Maximum number of iterations allowed. (Input)

X — Vector of length N containing the computed solution. (Output)

FVALUE — Scalar containing the value of the objective function at the computed solution. (Output)

Comments

1. Automatic workspace usage is

   NCONG N * (3 * N + 38 + MMAX) + 6 * (M + MMAX) + MAX(N, M) + 91 units, or

   DNCONG 2 * N * (3 * N + 38 + MMAX) + 12 * (M + MMAX) + MAX(N, M) + 163 units.
   MMAX = MAX(1, M)

   Workspace may be explicitly provided, if desired, by use of N2ONG/DN2ONG. The reference is

   ```
   CALL N2ONG (FCN, GRAD, M, ME, N, XGUESS, IBTYPE,
               XLB, XUB, IPRINT, MAXITN, X, FVALUE, WK,
               LWK, IWK, LIWK)
   ```

   The additional arguments are as follows:

   WK — Work vector of length N * (3 * N + 38 + MMAX) + 6 * (M + MMAX) + 72.

   LWK — Scalar containing the value for the length of WK.

   IWK — Work vector of length 19 + MAX(M, N).

   LIWK — Scalar containing the value for the length of IWK.

2. Informational errors

Type	Code	
4	1	Search direction uphill.
4	2	Line search took more than 5 function calls.
4	3	Maximum number of iterations exceeded.
4	4	Search direction is close to zero.
4	5	The constraints for the QP subproblem are inconsistent.

Algorithm

The routine NCONG is based on subroutine NLPQL, a FORTRAN code developed by Schittkowski (1986). It uses a successive quadratic programming method to solve the general nonlinear programming problem. The problem is stated as follows:

$$\min_{x \in \mathbf{R}^n} f(x)$$

$$\text{subject to} \quad g_j(x) = 0, \text{ for } \quad j = 1, \dots, m_e$$
$$g_j(x) \geq 0, \text{ for } \quad j = m_e + 1, \dots, m$$
$$x_l \leq x \leq x_u$$

where all problem functions are assumed to be continuously differentiable. The method, based on the iterative formulation and solution of quadratic programming subproblems, obtains subproblems by using a quadratic approximation of the Lagrangian and by linearizing the constraints. That is,

$$\min_{d \in \mathbf{R}^n} \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d$$
$$\text{subject to} \quad \nabla g_j(x_k)^T d + g_j(x_k) = 0, \qquad j = 1, \dots, m_e$$
$$\nabla g_j(x_k)^T d + g_j(x_k) \geq 0, \qquad j = m_e + 1, \dots, m$$
$$x_l - x_k \leq d \leq x_u - x_k$$

where B_k is a positive definite approximation of the Hessian and x_k is the current iterate. Let d_k be the solution of the subproblem. A line search is used to find a new point x_{k+1},

$$x_{k+1} = x_k + \lambda d_k, \quad \lambda \in (0, 1]$$

such that a “merit function” will have a lower function value at the new point. Here, the augmented Lagrange function (Schittkowski 1986) is used as the merit function.

When optimality is not achieved, B_k is updated according to the modified BFGS formula (Powell 1978). Note that this algorithm may generate infeasible points during the solution process. Therefore, if feasibility must be maintained for intermediate points, then this routine may not be suitable. For more theoretical and practical details, see Stoer (1985), Schittkowski (1983, 1986) and Gill et al. (1985).

Example

The problem

$$\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2$$
$$\text{subject to} \quad g_1(x) = x_1 - 2x_2 + 1 = 0$$
$$g_2(x) = -x_1^2 / 4 - x_2^2 + 1 \geq 0$$

is solved with an initial guess (2.0, 2.0).

```
      INTEGER    IBTYPE, IPRINT, M, MAXITN, ME, N
      PARAMETER  (IBTYPE=0, IPRINT=0, M=2, MAXITN=100, ME=1, N=2)
C
      REAL       FVALUE, X(N), XGUESS(N), XLB(N), XUB(N)
      EXTERNAL   FCN, GRAD, NCONG, WRRRN
C
      DATA XGUESS/2.0E0, 2.0E0/
      DATA XLB/-1.0E6, -1.0E6/, XUB/1.0E6, 1.0E6/
C
      CALL NCONG (FCN, GRAD, M, ME, N, XGUESS, IBTYPE, XLB, XUB,
     &            IPRINT, MAXITN, X, FVALUE)
C
      CALL WRRRN ('The solution is', N, 1, X, N, 0)
      END
C
      SUBROUTINE FCN (M, ME, N, X, ACTIVE, F, G)
      INTEGER    M, ME, N
      REAL       X(*), F, G(*)
      LOGICAL    ACTIVE(*)
C                                    Himmelblau problem 1
      F = (X(1)-2.0E0)**2 + (X(2)-1.0E0)**2
C
      IF (ACTIVE(1)) G(1) = X(1) - 2.0E0*X(2) + 1.0E0
      IF (ACTIVE(2)) G(2) = -(X(1)**2)/4.0E0 - X(2)**2 + 1.0E0
      RETURN
      END
C
      SUBROUTINE GRAD (M, ME, MMAX, N, X, ACTIVE, F, G, DF, DG)
      INTEGER    M, ME, MMAX, N
      REAL       X(*), F, G(*), DF(*), DG(MMAX,*)
      LOGICAL    ACTIVE(*)
C
      DF(1) = 2.0E0*(X(1)-2.0E0)
      DF(2) = 2.0E0*(X(2)-1.0E0)
C
      IF (ACTIVE(1)) THEN
         DG(1,1) = 1.0E0
         DG(1,2) = -2.0E0
      END IF
C
      IF (ACTIVE(2)) THEN
         DG(2,1) = -0.5E0*X(1)
         DG(2,2) = -2.0E0*X(2)
      END IF
      RETURN
      END
```

Output

```
 The solution is
 1   0.8229
 2   0.9114
```

CDGRD/DCDGRD (Single/Double precision)

Approximate the gradient using central differences.

Usage

```
CALL CDGRD (FCN, N, XC, XSCALE, EPSFCN, GC)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XC — Vector of length `N` containing the point at which the gradient is to be estimated. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.

EPSFCN — Estimate for the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.

GC — Vector of length `N` containing the estimated gradient at `XC`. (Output)

Comments

This is Algorithm A5.6.4, Dennis and Schnabel, 1983, page 323.

Algorithm

The routine `CDGRD` uses the following finite-difference formula to estimate the gradient of a function of *n* variables at *x*:

$$\frac{f\left(x+h_i e_i\right)-f\left(x-h_i e_i\right)}{2h_i} \quad \text{for } i=1,\ldots,n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, s_i is the scaling factor of the *i*-th variable, and e_i is the *i*-th unit vector. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

Example

In this example, the gradient of $f(x) = x_1 - x_1 x_2 - 2$ is estimated by the finite-difference method at the point (1.0, 1.0).

```
INTEGER    I, N, NOUT
PARAMETER  (N=2)
REAL       EPSFCN, GC(N), XC(N), XSCALE(N)
```

```
      EXTERNAL   CDGRD, FCN, UMACH
C                                  Initialization.
      DATA XSCALE/2*1.0E0/, XC/2*1.0E0/
C                                  Set function noise.
      EPSFCN = 0.01
C
      CALL CDGRD (FCN, N, XC, XSCALE, EPSFCN, GC)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (GC(I),I=1,N)
99999 FORMAT (’  The gradient is’, 2F8.2, /)
C
      END
C
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = X(1) - X(1)*X(2) - 2.0E0
C
      RETURN
      END
```

Output

```
The gradient is    0.00   -1.00
```

FDGRD/DFDGRD (Single/Double precision)

Approximate the gradient using forward differences.

Usage

`CALL FDGRD (FCN, N, XC, XSCALE, FC, EPSFCN, GC)`

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XC — Vector of length `N` containing the point at which the gradient is to be estimated. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.

FC — Scalar containing the value of the function at `XC`. (Input)

EPSFCN — Estimate of the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.

GC — Vector of length `N` containing the estimated gradient at `XC`. (Output)

Comments

This is Algorithm A5.6.3, Dennis and Schnabel, 1983, page 322.

Algorithm

The routine `FDGRD` uses the following finite-difference formula to estimate the gradient of a function of *n* variables at *x*:

$$\frac{f(x+h_ie_i)-f(x)}{h_i} \qquad \text{for } i=1,\ldots,n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, e_i is the *i*-th unit vector, and s_i is the scaling factor of the *i*-th variable. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended. When accuracy of the gradient is important, IMSL routine `CDGRD` (page 1007) should be used.

Example

In this example, the gradient of $f(x) = x_1 - x_1x_2 - 2$ is estimated by the finite-difference method at the point (1.0, 1.0).

```
      INTEGER    I, N, NOUT
      PARAMETER  (N=2)
      REAL       EPSFCN, FC, GC(N), XC(N), XSCALE(N)
      EXTERNAL   FCN, FDGRD, UMACH
C                                  Initialization.
      DATA XSCALE/2*1.0E0/, XC/2*1.0E0/
C                                  Set function noise.
      EPSFCN = 0.01
C                                  Get function value at current
C                                  point.
      CALL FCN (N, XC, FC)
C
      CALL FDGRD (FCN, N, XC, XSCALE, FC, EPSFCN, GC)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) (GC(I),I=1,N)
99999 FORMAT (’  The gradient is’, 2F8.2, /)
C
      END
C
      SUBROUTINE FCN (N, X, F)
      INTEGER    N
      REAL       X(N), F
C
      F = X(1) - X(1)*X(2) - 2.0E0
C
      RETURN
      END
```

Output

```
The gradient is    0.00   -1.00
```

FDHES/DFDHES (Single/Double precision)

Approximate the Hessian using forward differences and function values.

Usage

`CALL FDHES (FCN, N, XC, XSCALE, FC, EPSFCN, H, LDH)`

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XC — Vector of length `N` containing the point at which the Hessian is to be approximated. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.

FC — Function value at `XC`. (Input)

EPSFCN — Estimate of the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.

H — `N` by `N` matrix containing the finite difference approximation to the Hessian in the lower triangle. (Output)

LDH — Row dimension of `H` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

`FDHES` 2 * `N` units, or
`DFDHES` 4 * `N` units.

Workspace may be explicitly provided, if desired, by use of `F2HES`/`DF2HES`. The reference is

```
CALL F2HES (FCN, N, XC, XSCALE, FC, EPSFCN, H, LDH,
            WK1, WK2)
```

The additional arguments are as follows:

WK1 — Real work vector of length `N`.

WK2 — Real work vector of length `N`.

2. This is Algorithm A5.6.2 from Dennis and Schnabel, 1983; page 321.

Algorithm

The routine `FDHES` uses the following finite-difference formula to estimate the Hessian matrix of function *f* at *x*:

$$\frac{f\left(x+h_ie_i+h_je_j\right)-f\left(x+h_ie_i\right)-f\left(x+h_je_j\right)+f(x)}{h_ih_j}$$

where $h_i = \varepsilon^{1/3} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, $h_j = \varepsilon^{1/3} \max\{|x_j|, 1/s_i\} \operatorname{sign}(x_j)$, ε is the machine epsilon or user-supplied estimate of the relative noise, s_i and s_j are the scaling factors of the *i*-th and *j*-th variables, and e_i and e_j are the *i*-th and *j*-th unit vectors, respectively. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

Example

The Hessian is estimated for the following function at (1, –1)

$$f(x) = x_1^2 - x_1x_2 - 2$$

```
C                                 Declaration of variables
      INTEGER    N, LDHES, NOUT
      PARAMETER  (N=2, LDHES=2)
      REAL       XC(N), XSCALE(N), FVALUE, HES(LDHES,N), EPSFCN
      EXTERNAL   FCN
C                                   Initialization
      DATA XSCALE/2*1.0E0/, XC/1.0E0,-1.0E0/
C                                   Set function noise
      EPSFCN = 0.001
C                                   Evaluate the function at
C                                   current point
      CALL FCN (N, XC, FVALUE)
C                                 Get Hessian forward difference
C                                 approximation
      CALL FDHES (FCN, N, XC, XSCALE, FVALUE, EPSFCN, HES, LDHES)
```

```
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((HES(I,J),J=1,I),I=1,N)
99999 FORMAT ('  The lower triangle of the Hessian is', /,
     &         5X,F10.2,/,5X,2F10.2,/)
C
      END
C
      SUBROUTINE FCN (N, X, F)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER N
      REAL    X(N), F
C
      F = X(1)*(X(1) - X(2)) - 2.0E0
C
      RETURN
      END
```

Output

```
 The lower triangle of the Hessian is
  2.00
 -1.00      0.00
```

GDHES/DGDHES (Single/Double precision)

Approximate the Hessian using forward differences and a user-supplied gradient.

Usage

`CALL GDHES (GRAD, N, XC, XSCALE, GC, EPSFCN, H, LDH)`

Arguments

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

`N` – Length of `X` and `G`. (Input)
`X` – The point at which the gradient is evaluated. (Input)
`X` should not be changed by `GRAD`.
`G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

XC — Vector of length `N` containing the point at which the Hessian is to be estimated. (Input)

XSCALE — Vector of length `N` containing the diagonal scaling matrix for the variables. (Input)
In the absence of other information, set all entries to 1.0.

GC — Vector of length `N` containing the gradient of the function at `XC`. (Input)

EPSFCN — Estimate of the relative noise in the function. (Input)
`EPSFCN` must be less than or equal to 0.1. In the absence of other information, set `EPSFCN` to 0.0.

H — `N` by `N` matrix containing the finite-difference approximation to the Hessian in the lower triangular part and diagonal. (Output)

LDH — Leading dimension of `H` exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   `GDHES  N` units, or
   `DGDHES` 2 `* N` units.

   Workspace may be explicitly provided, if desired, by use of `G2HES`/`DG2HES`. The reference is

   ```
   CALL G2HES (GRAD, N, XC, XSCALE, GC, EPSFCN, H, LDH,
               WK)
   ```

   The additional argument is

   WK — Work vector of length `N`.

2. This is Algorithm A5.6.1, Dennis and Schnabel, 1983; page 320.

Algorithm

The routine `GDHES` uses the following finite-difference formula to estimate the Hessian matrix of function *F* at *x*:

$$\frac{g\left(x+h_j e_j\right)-g(x)}{h_j}$$

where $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, s_j is the scaling factor of the *j*-th variable, *g* is the analytic gradient of *F* at *x*, and e_j is the *j*-th unit vector. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

Example

The Hessian is estimated by the finite-difference method at point (1.0, 1.0) from the following gradient functions:

$$g_1 = 2x_1x_2 - 2$$

$$g_2 = x_1x_1 + 1$$

```
C                                 Declaration of variables
```

```
      INTEGER    N, LDHES, NOUT
      PARAMETER  (N=2, LDHES=2)
      REAL       XC(N), XSCALE(N), GC(N), HES(LDHES,N), EPSFCN
      EXTERNAL   GRAD
C
      DATA XSCALE/2*1.0E0/, XC/2*1.0E0/
C                                 Set function noise
      EPSFCN = 0.0
C                                 Evaluate the gradient at the
C                                 current point
      CALL GRAD (N, XC, GC)
C                                 Get Hessian forward-difference
C                                 approximation
      CALL GDHES (GRAD, N, XC, XSCALE, GC, EPSFCN, HES, LDHES)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((HES(I,J),J=1,N),I=1,N)
99999 FORMAT ('  THE HESSIAN IS', /, 2(5X,2F10.2,/),/)
C
      END
C
      SUBROUTINE GRAD (N, X, G)
C                                  SPECIFICATIONS FOR ARGUMENTS
      INTEGER N
      REAL    X(N), G(N)
C
      G(1) = 2.0E0*X(1)*X(2) - 2.0E0
      G(2) = X(1)*X(1) + 1.0E0
C
      RETURN
      END
```

Output

```
 THE HESSIAN IS
 2.00      2.00
 2.00      0.00
```

FDJAC/DFDJAC (Single/Double precision)

Approximate the Jacobian of M functions in N unknowns using forward differences.

Usage

```
CALL FDJAC (FCN, M, N, XC, XSCALE, FC, EPSFCN, FJAC,
            LDFJAC)
```

Arguments

FCN — User-supplied SUBROUTINE to evaluate the function to be minimized. The usage is CALL FCN (M, N, X, F), where

M – Length of F. (Input)
N – Length of X. (Input)
X – The point at which the function is evaluated. (Input)
X should not be changed by FCN.

F – The computed function at the point X.   (Output)

FCN must be declared EXTERNAL in the calling program.

M — The number of functions.   (Input)

N — The number of variables.   (Input)

XC — Vector of length N containing the point at which the gradient is to be estimated.   (Input)

XSCALE — Vector of length N containing the diagonal scaling matrix for the variables.   (Input)
In the absence of other information, set all entries to 1.0.

FC — Vector of length M containing the function values at XC.   (Input)

EPSFCN — Estimate for the relative noise in the function.   (Input)
EPSFCN must be less than or equal to 0.1. In the absence of other information, set EPSFCN to 0.0.

FJAC — M by N matrix containing the estimated Jacobian at XC.   (Output)

LDFJAC — Leading dimension of FJAC exactly as specified in the dimension statement of the calling program.   (Input)

Comments

1. Automatic workspace usage is

   FDJAC M units, or
   DFDJAC 2 * M units.

   Workspace may be explicitly provided, if desired, by use of F2JAC/DF2JAC. The reference is

   ```
   CALL F2JAC (FCN, M, N, XC, XSCALE, FC, EPSFCN, FJAC,
               LDFJAC, WK)
   ```

   The additional argument is

   WK — Work vector of length M.

2. This is Algorithm A5.4.1, Dennis and Schnabel, 1983, page 314.

Algorithm

The routine FDJAC uses the following finite-difference formula to estimate the Jacobian matrix of function *f* at *x*:

$$\frac{f\left(x+h_j e_j\right)-f(x)}{h_j}$$

where e_j is the *j*-th unit vector, $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, and s_j is the scaling factor of the *j*-th variable. For more details, see Dennis and Schnabel (1983).

Since the finite-difference method has truncation error, cancellation error, and rounding error, users should be aware of possible poor performance. When possible, high precision arithmetic is recommended.

Example

In this example, the Jacobian matrix of

$$f_1(x) = x_1 x_2 - 2$$
$$f_2(x) = x_1 - x_1 x_2 + 1$$

is estimated by the finite-difference method at the point (1.0, 1.0).

```
C                                  Declaration of variables
      INTEGER   N, M, LDFJAC, NOUT
      PARAMETER (N=2, M=2, LDFJAC=2)
      REAL      FJAC(LDFJAC,N), XSCALE(N), XC(N), FC(M), EPSFCN
      EXTERNAL  FCN, FDJAC, UMACH
C
      DATA XSCALE/2*1.0E0/, XC/2*1.0E0/
C                                  Set function noise
      EPSFCN = 0.01
C                                  Evaluate the function at the
C                                  current point
      CALL FCN (M, N, XC, FC)
C                                  Get Jacobian forward-difference
C                                  approximation
      CALL FDJAC (FCN, M, N, XC, XSCALE, FC, EPSFCN, FJAC, LDFJAC)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) ((FJAC(I,J),J=1,N),I=1,M)
99999 FORMAT ('  The Jacobian is', /, 2(5X,2F10.2,/),/)
C
      END
C
      SUBROUTINE FCN (M, N, X, F)
C                                   SPECIFICATIONS FOR ARGUMENTS
      INTEGER M, N
      REAL    X(N), F(M)
C
      F(1) = X(1)*X(2) - 2.0E0
      F(2) = X(1) - X(1)*X(2) + 1.0E0
C
      RETURN
      END
```

Output

```
 The Jacobian is
 1.00      1.00
 0.00     -1.00
```

CHGRD/DCHGRD (Single/Double precision)

Check a user-supplied gradient of a function.

Usage

```
CALL CHGRD (FCN, GRAD, N, X, INFO)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function of which the gradient will be checked. The usage is `CALL FCN (N, X, F)`, where

`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

GRAD — Vector of length `N` containing the estimated gradient at `X`. (Input)

N — Dimension of the problem. (Input)

X — Vector of length `N` containing the point at which the gradient is to be checked. (Input)

INFO — Integer vector of length `N`. (Output)

`INFO(I)` = 0 means the user-supplied gradient is a poor estimate of the numerical gradient at the point `X(I)`.

`INFO(I)` = 1 means the user-supplied gradient is a good estimate of the numerical gradient at the point `X(I)`.

`INFO(I)` = 2 means the user-supplied gradient disagrees with the numerical gradient at the point `X(I)`, but it might be impossible to calculate the numerical gradient.

`INFO(I)` = 3 means the user-supplied gradient and the numerical gradient are both zero at `X(I)`, and, therefore, the gradient should be rechecked at a different point.

Comments

1. Automatic workspace usage is

   `CHGRD` 2 * `N` + 2 units, or
   `DCHGRD` 4 * `N` + 4 units.

   Workspace may be explicitly provided, if desired, by use of `C2GRD`/`DC2GRD`. The reference is

   ```
   CALL C2GRD (FCN, GRAD, N, X, INFO, FX, XSCALE,
               EPSFCN, XNEW)
   ```

The additional arguments are as follows:

FX — The functional value at `X`.

XSCALE — Real vector of length `N` containing the diagonal scaling matrix.

EPSFCN — The relative “noise” of the function `FCN`.

XNEW — Real work vector of length `N`.

2. Informational errors

Type	Code	
4	1	The user-supplied gradient is a poor estimate of the numerical gradient.

Algorithm

The routine `CHGRD` uses the following finite-difference formula to estimate the gradient of a function of *n* variables at *x*:

$$g_i(x) = \frac{f(x + h_i e_i) - f(x)}{h_i} \qquad \text{for } i = 1, \ldots, n$$

where $h_i = \varepsilon^{1/2} \max\{|x_i|, 1/s_i\} \operatorname{sign}(x_i)$, ε is the machine epsilon, e_i is the *i*-th unit vector, and s_i is the scaling factor of the *i*-th variable.

The routine `CHGRD` checks the user-supplied gradient $\nabla f(x)$ by comparing it with the finite-difference gradient $g(x)$. If

$$\left| g_i(x) - (\nabla f(x))_i \right| < \tau \left| (\nabla f(x))_i \right|$$

where $\tau = \varepsilon^{1/4}$, then $(\nabla f(x))_i$, which is the *i*-th element of $\nabla f(x)$, is declared correct; otherwise, `CHGRD` computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $(\nabla f(x))_i$ is reported as incorrect. In the case of a large error bound, `CHGRD` uses a nearly optimal stepsize to recompute $g_i(x)$ and reports that $(\nabla f(x))_i$ is correct if

$$\left| g_i(x) - (\nabla f(x))_i \right| < 2\tau \left| (\nabla f(x))_i \right|$$

Otherwise, $(\nabla f(x))_i$ is considered incorrect unless the error bound for the optimal step is greater than $\tau \, |(\nabla f(x))_i|$. In this case, the numeric gradient may be impossible to compute correctly. For more details, see Schnabel (1985).

Example

The user-supplied gradient of

$$f(x) = x_i + x_2 e^{-(t - x_3)^{2/x_4}}$$

at (625, 1, 3.125, 0.25) is checked where t = 2.125.

```
C                                Declare variables
      INTEGER    N
      PARAMETER  (N=4)
C
      INTEGER    INFO(N)
      REAL       GRAD(N), X(N)
      EXTERNAL   CHGRD, DRIV, FCN, WRIRN
C
C                                Input values for point X
C                                X = (625.0, 1.0, 3.125, .25)
C
      DATA X/625.0E0, 1.0E0, 3.125E0, 0.25E0/
C
      CALL DRIV (N, X, GRAD)
C
      CALL CHGRD (FCN, GRAD, N, X, INFO)
      CALL WRIRN ('The information vector', 1, N, INFO, 1, 0)
C
      END
C
      SUBROUTINE FCN (N, X, FX)
      INTEGER    N
      REAL       X(N), FX
C
      REAL       EXP
      INTRINSIC  EXP
C
      FX = X(1) + X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))
      RETURN
      END
C
      SUBROUTINE DRIV (N, X, GRAD)
      INTEGER    N
      REAL       X(N), GRAD(N)
C
      REAL       EXP
      INTRINSIC  EXP
C
      GRAD(1) = 1.0E0
      GRAD(2) = EXP(-1.0E0*(2.125E0-X(3))**2/X(4))
      GRAD(3) = X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))*2.0E0/X(4)*
     &          (2.125-X(3))
      GRAD(4) = X(2)*EXP(-1.0E0*(2.125E0-X(3))**2/X(4))*
     &          (2.125E0-X(3))**2/(X(4)*X(4))
      RETURN
      END
```

Output

```
 The information vector
 1   2   3   4
 1   1   1   1
```

CHHES/DCHHES (Single/Double precision)

Check a user-supplied Hessian of an analytic function.

Usage

```
CALL CHHES (GRAD, HESS, N, X, INFO, LDINFO)
```

Arguments

GRAD — User-supplied `SUBROUTINE` to compute the gradient at the point `X`. The usage is `CALL GRAD (N, X, G)`, where

> `N` – Length of `X` and `G`. (Input)
> `X` – The point at which the gradient is evaluated. `X` should not be changed by `GRAD`. (Input)
> `G` – The gradient evaluated at the point `X`. (Output)

`GRAD` must be declared `EXTERNAL` in the calling program.

HESS — User-supplied `SUBROUTINE` to compute the Hessian at the point `X`. The usage is `CALL HESS (N, X, H, LDH)`, where

> `N` – Length of `X`. (Input)
> `X` – The point at which the Hessian is evaluated. (Input)
> `X` should not be changed by `HESS`.
> `H` – The Hessian evaluated at the point `X`. (Output)
> `LDH` – Leading dimension of `H` exactly as specified in in the dimension statement of the calling program. (Input)

`HESS` must be declared `EXTERNAL` in the calling program.

N — Dimension of the problem. (Input)

X — Vector of length `N` containing the point at which the Hessian is to be checked. (Input)

INFO — Integer matrix of dimension `N` by `N`. (Output)

`INFO(I, J)` = 0 means the Hessian is a poor estimate for function `I` at the point `X(J)`.

`INFO(I, J)` = 1 means the Hessian is a good estimate for function `I` at the point `X(J)`.

`INFO(I, J)` = 2 means the Hessian disagrees with the numerical Hessian for function `I` at the point `X(J)`, but it might be impossible to calculate the numerical Hessian.

`INFO(I, J)` = 3 means the Hessian for function `I` at the point `X(J)` and the numerical Hessian are both zero, and, therefore, the gradient should be rechecked at a different point.

LDINFO — Leading dimension of `INFO` exactly as specified in the dimension statement of the calling program. (Input)

Comments

Automatic workspace usage is

CHHES N * (N + 5) + 1 units, or
DCHHES 2 * N * (N + 5) + 2 units.

Workspace may be explicitly provided, if desired, by use of C2HES/DC2HES. The reference is

```
CALL C2HES (GRAD, HESS, N, X, INFO, LDINFO, G, HX, HS,
            XSCALE, EPSFCN, INFT, NEWX)
```

The additional arguments are as follows:

G — Vector of length N containing the value of the gradient GRD at X.

HX — Real matrix of dimension N by N containing the Hessian evaluated at X.

HS — Real work vector of length N.

XSCALE — Vector of length N used to store the diagonal scaling matrix for the variables.

EPSFCN — Estimate of the relative noise in the function.

INFT — Vector of length N. For I = 1 through N, INFT contains information about the Jacobian.

NEWX — Real work array of length N.

Algorithm

The routine CHHES uses the following finite-difference formula to estimate the Hessian of a function of *n* variables at *x*:

$$B_{ij}(x) = \left(g_i\left(x + h_j e_j\right) - g_i(x)\right) / h_j \qquad \text{for } j = 1, \ldots, n$$

where $h_j = \varepsilon^{1/2}\max\{|x_j|, 1/s_j\}\,\mathrm{sign}(x_j)$, ε is the machine epsilon, e_j is the *j*-th unit vector, s_j is the scaling factor of the *j*-th variable, and $g_i(x)$ is the gradient of the function with respect to the *i*-th variable.

Next, CHHES checks the user-supplied Hessian $H(x)$ by comparing it with the finite difference approximation $B(x)$. If

$$|B_{ij}(x) - H_{ij}(x)| < \tau\, |H_{ij}(x)|$$

where $\tau = \varepsilon^{1/4}$, then $H_{ij}(x)$ is declared correct; otherwise, CHHES computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $H_{ij}(x)$ is reported as incorrect. In the case of a large error bound, CHHES uses a nearly optimal stepsize to recompute $B_{ij}(x)$ and reports that $B_{ij}(x)$ is correct if

$$|B_{ij}(x) - H_{ij}(x)| < 2\tau\, |H_{ij}(x)|$$

Otherwise, $H_{ij}(x)$ is considered incorrect unless the error bound for the optimal step is greater than $\tau\,|H_{ij}(x)|$. In this case, the numeric approximation may be impossible to compute correctly. For more details, see Schnabel (1985).

Example

The user-supplied Hessian of

$$f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

at (−1.2, 1.0) is checked, and the error is found.

```
      INTEGER    LDINFO, N
      PARAMETER  (N=2, LDINFO=N)
C
      INTEGER    INFO(LDINFO,N)
      REAL       X(N)
      EXTERNAL   CHHES, GRD, HES
C
C                                Input values for X
C                                  X = (-1.2, 1.0)
C
      DATA X/-1.2, 1.0/
C
      CALL CHHES (GRD, HES, N, X, INFO, LDINFO)
C
      END
C
      SUBROUTINE GRD (N, X, UG)
      INTEGER    N
      REAL       X(N), UG(N)
C
      UG(1) = -400.0*X(1)*(X(2)-X(1)*X(1)) + 2.0*X(1) - 2.0
      UG(2) = 200.0*X(2) - 200.0*X(1)*X(1)
      RETURN
      END
C
      SUBROUTINE HES (N, X, HX, LDHS)
      INTEGER    N, LDHS
      REAL       X(N), HX(LDHS,N)
C
      HX(1,1) = -400.0*X(2) + 1200.0*X(1)*X(1) + 2.0
      HX(1,2) = -400.0*X(1)
      HX(2,1) = -400.0*X(1)
C                                 A sign change is made to HX(2,2)
C
      HX(2,2) = -200.0
      RETURN
      END
```

Output

```
*** FATAL    ERROR 1 from CHHES.  The Hessian evaluation with respect to
***          X(2) and X(2) is a poor estimate.
```

CHJAC/DCHJAC (Single/Double precision)

Check a user-supplied Jacobian of a system of equations with `M` functions in `N` unknowns.

Usage

```
CALL CHJAC (FCN, JAC, M, N, X, INFO, LDINFO)
```

Arguments

FCN — User-supplied `SUBROUTINE` to evaluate the function to be minimized. The usage is `CALL FCN (M, N, X, F)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`F` – The computed function value at the point `X`. (Output)

`FCN` must be declared `EXTERNAL` in the calling program.

JAC — User-supplied `SUBROUTINE` to evaluate the Jacobian at a point `X`. The usage is `CALL JAC (M, N, X, FJAC, LDFJAC)`, where

`M` – Length of `F`. (Input)
`N` – Length of `X`. (Input)
`X` – The point at which the function is evaluated. (Input)
`X` should not be changed by `FCN`.
`FJAC` – The computed `M` by `N` Jacobian at the point `X`. (Output)
`LDFJAC` – Leading dimension of `FJAC`. (Input)

`JAC` must be declared `EXTERNAL` in the calling program.

M — The number of functions in the system of equations. (Input)

N — The number of unknowns in the system of equations. (Input)

X — Vector of length `N` containing the point at which the Jacobian is to be checked. (Input)

INFO — Integer matrix of dimension `M` by `N`. (Output)

`INFO(I, J)` = 0 means the user-supplied Jacobian is a poor estimate for function `I` at the point `X(J)`.

`INFO(I, J)` = 1 means the user-supplied Jacobian is a good estimate for function `I` at the point `X(J)`.

`INFO(I, J)` = 2 means the user-supplied Jacobian disagrees with the numerical Jacobian for function `I` at the point `X(J)`, but it might be impossible to calculate the numerical Jacobian.

INFO(I, J) = 3 means the user-supplied Jacobian for function I at the point X(J) and the numerical Jacobian are both zero. Therefore, the gradient should be rechecked at a different point.

LDINFO — Leading dimension of INFO exactly as specified in the dimension statement of the calling program. (Input)

Comments

1. Automatic workspace usage is

   CHJAC M * (N + 1) + 4 * N + 1 units, or
   DCHJAC 2 * M * (N + 1) + 7 * N + 2 units.

   Workspace may be explicitly provided, if desired, by use of C2JAC/DC2JAC. The reference is

   ```
   CALL C2JAC (FCN, JAC, N, X, INFO, LDINFO, FX, FJAC,
               GRAD, XSCALE, EPSFCN, INFT, NEWX)
   ```

   The additional arguments are as follows:

   FX — Vector of length M containing the value of each function in FCN at X.

   FJAC — Real matrix of dimension M by N containing the Jacobian of FCN evaluated at X.

   GRAD — Real work vector of length N used to store the gradient of each function in FCN.

   XSCALE — Vector of length N used to store the diagonal scaling matrix for the variables.

   EPSFCN — Estimate of the relative noise in the function.

   INFT — Vector of length N. For I = 1 through N, INFT contains information about the Jacobian.

   NEWX — Real work array of length N.

2. Informational errors

Type	Code	
4	1	The user-supplied Jacobian is a poor estimate of the numerical Jacobian.

Algorithm

The routine CHJAC uses the following finite-difference formula to estimate the gradient of the *i*-th function of *n* variables at *x*:

$$g_{ij}(x) = (f_i(x + h_j e_j) - f_i(x))/h_j \qquad \text{for } j = 1, \ldots, n$$

where $h_j = \varepsilon^{1/2} \max\{|x_j|, 1/s_j\} \operatorname{sign}(x_j)$, ε is the machine epsilon, e_j is the *j*-th unit vector, and s_j is the scaling factor of the *j*-th variable.

Next, CHJAC checks the user-supplied Jacobian $J(x)$ by comparing it with the finite difference gradient $g_i(x)$. If

$$|g_{ij}(x) - J_{ij}(x)| < \tau \, |J_{ij}(x)|$$

where $\tau = \varepsilon^{1/4}$, then $J_{ij}(x)$ is declared correct; otherwise, CHJAC computes the bounds of calculation error and approximation error. When both bounds are too small to account for the difference, $J_{ij}(x)$ is reported as incorrect. In the case of a large error bound, CHJAC uses a nearly optimal stepsize to recompute $g_{ij}(x)$ and reports that $J_{ij}(x)$ is correct if

$$|g_{ij}(x) - J_{ij}(x)| < 2\tau \, |J_{ij}(x)|$$

Otherwise, $J_{ij}(x)$ is considered incorrect unless the error bound for the optimal step is greater than $\tau \, |J_{ij}(x)|$. In this case, the numeric gradient may be impossible to compute correctly. For more details, see Schnabel (1985).

Example

The user-supplied Jacobian of

$$f_1 = 1 - x_1$$

$$f_2 = 10\left(x_2 - x_1^2\right)$$

at (–1.2, 1.0) is checked.

```
      INTEGER    LDINFO, N
      PARAMETER  (M=2,N=2,LDINFO=M)
C
      INTEGER    INFO(LDINFO,N)
      REAL       X(N)
      EXTERNAL   CHJAC, FCN, JAC, WRIRN
C
C                                 Input value for X
C                                    X = (-1.2, 1.0)
C
      DATA X/-1.2, 1.0/
C
      CALL CHJAC (FCN, JAC, M, N, X, INFO, LDINFO)
      CALL WRIRN ('The information matrix', M, N, INFO, LDINFO, 0)
C
      END
C
      SUBROUTINE FCN (M, N, X, F)
      INTEGER    M, N
      REAL       X(N), F(M)
C
      F(1) = 1.0 - X(1)
      F(2) = 10.0*(X(2)-X(1)*X(1))
      RETURN
      END
C
      SUBROUTINE JAC (M, N, X, FJAC, LDFJAC)
      INTEGER    M, N, LDFJAC
      REAL       X(N), FJAC(LDFJAC,N)
C
      FJAC(1,1) = -1.0
      FJAC(1,2) = 0.0
      FJAC(2,1) = -20.0*X(1)
      FJAC(2,2) = 10.0
      RETURN
      END
```

Output

```
*** WARNING  ERROR 2 from C2JAC.  The numerical value of the Jacobian
***          evaluation for function 1 at the point X(2) = 1.000000E+00 and
***          the user-supplied value are both zero.  The Jacobian for this
***          function should probably be re-checked at another value for
***          this point.

The information matrix
    1   2
1   1   3
2   1   1
```

GGUES/DGGUES (Single/Double precision)

Generate points in an `N`-dimensional space.

Usage

`CALL GGUES (N, A, B, K, IDO, S)`

Arguments

N — Dimension of the space. (Input)

A — Vector of length `N`. (Input)
See `B`.

B — Real vector of length `N`. (Input)
`A` and `B` define the rectangular region in which the points will be generated, i.e., `A(I)` < `S(I)` < `B(I)` for `I` = 1, 2, …, `N`. Note that if `B(I)` < `A(I)`, then `B(I)` < `S(I)` < `A(I)`.

K — The number of points to be generated. (Input)

IDO — Initialization parameter. (Input/Output)
`IDO` must be set to zero for the first call. `GGUES` resets `IDO` to 1 and returns the first generated point in `S`. Subsequent calls should be made with `IDO` = 1.

S — Vector of length `N` containing the generated point. (Output)
Each call results in the next generated point being stored in `S`.

Comments

1. Automatic workspace usage is

GGUES   N units, or
DGGUES 2 * N units.

Workspace may be explicitly provided, if desired, by use of G2UES/DG2UES. The reference is

```
CALL G2UES (N, A, B, K, IDO, S, WK, IWK)
```

The additional argument is

WK — Work vector of length N. WK must be preserved between calls to G2UES.

IWK — Work vector of length 10. IWK must be preserved between calls to G2UES.

2. Informational error

Type	Code	
4	1	Attempt to generate more than K points.

3. The routine GGUES may be used with any nonlinear optimization routine that requires starting points. The rectangle to be searched (defined by A, B, and N) must be determined; and the number of starting points, K, must be chosen. One possible use for GGUES would be to call GGUES to generate a point in the chosen rectangle. Then, call the nonlinear optimization routine using this point as an initial guess for the solution. Repeat this process K times. The number of iterations that the optimization routine is allowed to perform should be quite small (5 to 10) during this search process. The best (or best several) point(s) found during the search may be used as an initial guess to allow the optimization routine to determine the optimum more accurately. In this manner, an N dimensional rectangle may be effectively searched for a global optimum of a nonlinear function. The choice of K depends upon the nonlinearity of the function being optimized. A function with many local optima requires a larger value than a function with only a few local optima.

Algorithm

The routine GGUES generates starting points for algorithms that optimize functions of several variables–or, almost equivalently–algorithms that solve simultaneous nonlinear equations.

The routine GGUES is based on systematic placement of points to optimize the dispersion of the set. For more details, see Aird and Rice (1977).

Example

We want to search the rectangle with vertices at coordinates (1, 1), (3, 1), (3, 2), and (1, 2) ten times for a global optimum of a nonlinear function. To do this, we need to generate starting points. The following example illustrates the use of GGUES in this process:

```
C                                 Variable Declarations
      INTEGER    N
      PARAMETER  (N=2)
C
      INTEGER    IDO, J, K, NOUT
      REAL       A(N), B(N), S(N)
      EXTERNAL   GGUES, UMACH
C                                 Initializations
C
C                                 A   = ( 1.0, 1.0)
C                                 B   = ( 3.0, 2.0)
C
      DATA A/1.0, 1.0/
      DATA B/3.0, 2.0/
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998)
99998 FORMAT ('  Point Number', 7X, 'Generated Point')
C
      K = 10
      IDO = 0
      DO 10  J=1, K
         CALL GGUES (N, A, B, K, IDO, S)
C
         WRITE (NOUT,99999) J, S(1), S(2)
99999    FORMAT (1X, I7, 14X, '(', F4.1, ',', F6.3, ')')
C
   10 CONTINUE
C
      END
```

Output

```
Point Number       Generated Point
    1              ( 1.5, 1.125)
    2              ( 2.0, 1.500)
    3              ( 2.5, 1.750)
    4              ( 1.5, 1.375)
    5              ( 2.0, 1.750)
    6              ( 1.5, 1.625)
    7              ( 2.5, 1.250)
    8              ( 1.5, 1.875)
    9              ( 2.0, 1.250)
   10              ( 2.5, 1.500)
```

Chapter 9: Basic Matrix/Vector Operations

Routines

9.2. Other Matrix/Vector Operations

9.2.1 Matrix Copy

9.2.2 Matrix Conversion

Basic Linear Algebra Subprograms

The basic linear algebra subprograms, normally referred to as the BLAS, are routines for low-level operations such as dot products, matrix times vector, and matrix times matrix. Lawson et al. (1979) published the original set of 38 BLAS. The IMSL BLAS collection includes these 38 subprograms plus additional ones that extend their functionality. Since Dongarra et al. (1988 and 1990) published extensions to this set, it is customary to refer to the original 38 as Level 1 BLAS. The Level 1 operations are performed on one or two vectors of data. An extended set of subprograms perform operations involving a matrix and one or two vectors. These are called the Level 2 BLAS (page 1046). An additional extended set of operations on matrices is called the Level 3 BLAS (page 1046).

Users of the BLAS will often benefit from using versions of the BLAS supplied by hardware vendors, if available. This can provide for more efficient execution of many application programs. The BLAS provided by IMSL are written in FORTRAN. Those supplied by vendors may be written in other languages, such as assembler. The documentation given below for the BLAS is compatible with a vendor's version of the BLAS that conforms to the published specifications.

Programming Notes for Level 1 BLAS

The Level 1 BLAS do not follow the usual IMSL naming conventions. Instead, the names consist of a prefix of one or more of the letters "I," "S," "D," "C" and "Z;" a root name; and sometimes a suffix. For subprograms involving a mixture of data types, the output type is indicated by the first prefix letter. The suffix denotes a variant algorithm. The prefix denotes the type of the operation according to the following table:

I	Integer		
S	Real	C	Complex
D	Double	Z	Double complex
SD	Single and Double	CZ	Single and double complex
DQ	Double and Quadruple	ZQ	Double and quadruple complex

Vector arguments have an increment parameter that specifies the storage space or stride between elements. The correspondence between the vectors *x* and *y* and the arguments SX and SY, and INCX and INCY is

$$x_i = \begin{cases} \mathrm{SX}\left((\mathrm{I}-1)*\mathrm{INCX}+1\right) & \text{if INCX} \geq 0 \\ \mathrm{SX}\left((\mathrm{I}-\mathrm{N})*\mathrm{INCX}+1\right) & \text{if INCX} < 0 \end{cases}$$

$$y_i = \begin{cases} \mathrm{SY}\left((\mathrm{I}-1)*\mathrm{INCY}+1\right) & \text{if INCY} \geq 0 \\ \mathrm{SY}\left((\mathrm{I}-\mathrm{N})*\mathrm{INCY}+1\right) & \text{if INCY} < 0 \end{cases}$$

Function subprograms SXYZ, DXYZ, page 1040, refer to a third vector argument *z*. The storage increment INCZ for *z* is defined like INCX, INCY. In the Level 1 BLAS, only positive values of INCX are allowed for operations that have a single vector argument. The loops in all of the Level 1 BLAS process the vector arguments in order of increasing *i*. For INCX, INCY, INCZ < 0, this implies processing in reverse storage order.

The function subprograms in the Level 1 BLAS are all illustrated by means of an assignment statement. For example, see SDOT (page 1039). Any value of a function subprogram can be used in an expression or as a parameter passed to a subprogram as long as the data types agree.

Descriptions of the Level 1 BLAS Subprograms

The set of Level 1 BLAS are summarized in Table 9.1. This table also lists the page numbers where the subprograms are described in more detail.

Specification of the Level 1 BLAS

With the definitions,

$$\mathrm{MX} = \max\{1, 1 + (N-1)|\mathrm{INCX}|\}$$

$$\mathrm{MY} = \max\{1, 1 + (N-1)|\mathrm{INCY}|\}$$

$$\mathrm{MZ} = \max\{1, 1 + (N-1)|\mathrm{INCZ}|\}$$

the subprogram descriptions assume the following FORTRAN declarations:

```
IMPLICIT INTEGER          (I-N)
IMPLICIT REAL             S
IMPLICIT DOUBLE PRECISION D
IMPLICIT COMPLEX          C
IMPLICIT DOUBLE COMPLEX   Z

INTEGER                   IX(MX)
REAL                      SX(MX), SY(MY), SZ(MZ),
                          SPARAM(5)
DOUBLE PRECISION          DX(MX), DY(MY), DZ(MZ),
                          DPARAM(5)
DOUBLE PRECISION          DACC(2), DZACC(4)
COMPLEX                   CX(MX), CY(MY)
DOUBLE COMPLEX            ZX(MX), ZY(MY)
```

Since FORTRAN 77 does not include the type DOUBLE COMPLEX, subprograms with DOUBLE COMPLEX arguments are not available for all systems. Some systems use the declaration COMPLEX * 16 instead of DOUBLE COMPLEX.

In the following descriptions, the original BLAS are marked with an * in the left column.

Table 9.1: Level 1 Basic Linear Algebra Subprograms

Operation	Integer	Real	Double	Complex	Double Complex	Pg.
$x_i \leftarrow a$	ISET	SSET	DSET	CSET	ZSET	1037
$y_i \leftarrow x_i$	ICOPY	SCOPY	DCOPY	CCOPY	ZCOPY	1037
$x_i \leftarrow ax_i$ $a \in \mathbf{R}$		SSCAL	DSCAL	CSCAL CSSCAL	ZSCAL ZDSCAL	1037
$y_i \leftarrow ax_i$ $a \in \mathbf{R}$		SVCAL	DVCAL	CVCAL CSVCAL	ZVCAL ZDVCAL	1038
$x_i \leftarrow x_i + a$	IADD	SADD	DADD	CADD	ZADD	1038
$x_i \leftarrow a - x_i$	ISUB	SSUB	DSUB	CSUB	ZSUB	1038
$y_i \leftarrow ax_i + y_i$		SAXPY	DAXPY	CAXPY	ZAXPY	1038
$y_i \leftrightarrow x_i$	ISWAP	SSWAP	DSWAP	CSWAP	ZSWAP	1038
$x \cdot y$ $\bar{x} \cdot \mathrm{y}$		SDOT	DDOT	CDOTU CDOTC	ZDOTU ZDOTC	1039
$x \cdot y\,\dagger$ $\bar{x} \cdot y\,\dagger$		DSDOT		CZDOTU CZDOTC	ZQDOTU ZQDOTC	1039
$a + x \cdot y\,\dagger$ $a + \bar{x} \cdot y\,\dagger$		SDSDOT	DQDDOT	CZUDOT CZCDOT	ZQUDOT ZQCDOT	1039
$b + x \cdot y\,\dagger$ ACC $+ b + x \cdot y\,\dagger$		SDDOTI SDDOTA	DQDOTI DQDOTA	CZDOTI CZDOTA	ZQDOTI ZQDOTA	1040
$z_i \leftarrow x_i y_i$		SHPROD	DHPROD			1040
$\sum x_i y_i z_i$		SXYZ	DXYZ			1040
$\sum x_i$	ISUM	SSUM	DSUM			1041
$\sum \lvert x_i \rvert$		SASUM	DASUM	SCASUM	DZASUM	1041
$\lVert x \rVert_2$		SNRM2	DNRM2	SCNRM2	DZNRM2	1041
$\Pi\, x_i$		SPRDCT	DPRDCT			1041
$i:\; x_i = \min_j x_j$	IIMIN	ISMIN	IDMIN			1042
$i:\; x_i = \max_j x_j$	IIMAX	ISMAX	IDMAX			1042
$i:\; \lvert x_i \rvert = \min_j \lvert x_j \rvert$		ISAMIN	IDAMIN	ICAMIN	IZAMIN	1042

Operation	Integer	Real	Double	Complex	Double Complex	Pg.
i : $\|x_i\| = \max_j \|x_j\|$		`ISAMAX`	`IDAMAX`	`ICAMAX`	`IZAMAX`	1042
Construct Givens rotation		`SROTG`	`DROTG`			1043
Apply Givens rotation		`SROT`	`DROT`	`CSROT`	`ZDROT`	1043
Construct modified Givens transform		`SROTMG`	`DROTMG`			1044
Apply modified Givens transform		`SROTM`	`DROTM`	`CSROTM`	`ZDROTM`	1045

†Higher precision accumulation used

Set a Vector to a Constant Value

```
CALL ISET (N, IA, IX, INCX)
CALL SSET (N, SA, SX, INCX)
CALL DSET (N, DA, DX, INCX)
CALL CSET (N, CA, CX, INCX)
CALL ZSET (N, ZA, ZX, INCX)
```

These subprograms set $x_i \leftarrow a$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Copy a Vector

```
 CALL ICOPY (N, IX, INCX, IY, INCY)
*CALL SCOPY (N, SX, INCX, SY, INCY)
*CALL DCOPY (N, DX, INCX, DY, INCY)
*CALL CCOPY (N, CX, INCX, CY, INCY)
 CALL ZCOPY (N, ZX, INCX, ZY, INCY)
```

These subprograms set $y_i \leftarrow x_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Scale a Vector

```
*CALL SSCAL (N, SA, SX, INCX)
*CALL DSCAL (N, DA, DX, INCX)
*CALL CSCAL (N, CA, CX, INCX)
 CALL ZSCAL (N, ZA, ZX, INCX)
*CALL CSSCAL (N, SA, CX, INCX)
 CALL ZDSCAL (N, DA, ZX, INCX)
```

These subprograms set $x_i \leftarrow ax_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately. CAUTION: For `CSSCAL` and `ZDSCAL`, the scalar quantity a is real and the vector x is complex.

Multiply a Vector by a Constant

```
CALL SVCAL (N, SA, SX, INCX, SY, INCY)
CALL DVCAL (N, DA, DX, INCX, DY, INCY)
CALL CVCAL (N, CA, CX, INCX, CY, INCY)
CALL ZVCAL (N, ZA, ZX, INCX, ZY, INCY)
CALL CSVCAL (N, SA, CX, INCX, CY, INCY)
CALL ZDVCAL (N, DA, ZX, INCX, ZY, INCY)
```

These subprograms set $y_i \leftarrow ax_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately. CAUTION: For `CSVCAL` and `ZDVCAL`, the scalar quantity a is real and the vector x is complex.

Add a Constant to a Vector

```
CALL IADD (N, IA, IX, INCX)
CALL SADD (N, SA, SX, INCX)
CALL DADD (N, DA, DX, INCX)
CALL CADD (N, CA, CX, INCX)
CALL ZADD (N, ZA, ZX, INCX)
```

These subprograms set $x_i \leftarrow x_i + a$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Subtract a Vector from a Constant

```
CALL ISUB (N, IA, IX, INCX)
CALL SSUB (N, SA, SX, INCX)
CALL DSUB (N, DA, DX, INCX)
CALL CSUB (N, CA, CX, INCX)
CALL ZSUB (N, ZA, ZX, INCX)
```

These subprograms set $x_i \leftarrow a - x_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Constant Times a Vector Plus a Vector

```
*CALL SAXPY (N, SA, SX, INCX, SY, INCY)
*CALL DAXPY (N, DA, DX, INCX, DY, INCY)
*CALL CAXPY (N, CA, CX, INCX, CY, INCY)
 CALL ZAXPY (N, ZA, ZX, INCX, ZY, INCY)
```

These subprograms set $y_i \leftarrow ax_i + y_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Swap Two Vectors

```
 CALL ISWAP (N, IX, INCX, IY, INCY)
*CALL SSWAP (N, SX, INCX, SY, INCY)
*CALL DSWAP (N, DX, INCX, DY, INCY)
*CALL CSWAP (N, CX, INCX, CY, INCY)
 CALL ZSWAP (N, ZX, INCX, ZY, INCY)
```

These subprograms perform the exchange $y_i \leftrightarrow x_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Dot Product

```
*SW =  SDOT  (N, SX, INCX, SY, INCY)
*DW =  DDOT  (N, DX, INCX, DY, INCY)
*CW =  CDOTU (N, CX, INCX, CY, INCY)
*CW =  CDOTC (N, CX, INCX, CY, INCY)
 ZW =  ZDOTU (N, ZX, INCX, ZY, INCY)
 ZW =  ZDOTC (N, ZX, INCX, ZY, INCY)
```

The function subprograms SDOT, DDOT, CDOTU, and ZDOTU compute

$$\sum_{i=1}^{N} x_i y_i$$

The function subprograms CDOTC and ZDOTC compute

$$\sum_{i=1}^{N} \bar{x}_i y_i$$

The suffix C indicates that the complex conjugates of x_i are used. The suffix U indicates that the unconjugated values of x_i are used. If $N \le 0$, then the subprograms return zero.

Dot Product with Higher Precision Accumulation

```
*DW =  DSDOT  (N, SX, INCX, SY, INCY)
 CW =  CZDOTC (N, CX, INCX, CY, INCY)
 CW =  CZDOTU (N, CX, INCX, CY, INCY)
 ZW =  ZQDOTC (N, ZX, INCX, ZY, INCY)
 ZW =  ZQDOTU (N, ZX, INCX, ZY, INCY)
```

The function subprogram DSDOT computes

$$\sum_{i=1}^{N} x_i y_i$$

using double precision accumulation. The function subprograms CZDOTU and ZQDOTU compute

$$\sum_{i=1}^{N} x_i y_i$$

using double and quadruple complex accumulation, respectively. The function subprograms CZDOTC and ZQDOTC compute

$$\sum_{i=1}^{N} \bar{x}_i y_i$$

using double and quadruple complex accumulation, respectively. If $N \le 0$, then the subprograms return zero.

Constant Plus Dot Product with Higher Precision Accumulation

```
*SW = SDSDOT (N, SA, SX, INCX, SY, INCY)
 DW = DQDDOT (N, DA, DX, INCX, DY, INCY)
 CW = CZCDOT (N, CA, CX, INCX, CY, INCY)
 CW = CZUDOT (N, CA, CX, INCX, CY, INCY)
 ZW = ZQCDOT (N, ZA, ZX, INCX, ZY, INCY)
 ZW = ZQUDOT (N, ZA, ZX, INCX, ZY, INCY)
```

The function subprograms SDSDOT, DQDDOT, CZUDOT, and ZQUDOT compute

$$a + \sum_{i=1}^{N} x_i y_i$$

using higher precision accumulation where SDSDOT uses double precision accumulation, DQDDOT uses quadruple precision accumulation, CZUDOT uses double complex accumulation, and ZQUDOT uses quadruple complex accumulation. The function subprograms CZCDOT and ZQCDOT compute

$$a + \sum_{i=1}^{N} \bar{x}_i y_i$$

using double complex and quadruple complex accumulation, respectively. If $N \le 0$, then the subprograms return zero.

Dot Product Using the Accumulator

```
 SW =  SDDOTI (N, SB,  DACC, SX, INCX, SY, INCY)
 SW =  SDDOTA (N, SB,  DACC, SX, INCX, SY, INCY)
 CW =  CZDOTI (N, CB,  DACC, CX, INCX, CY, INCY)
 CW =  CZDOTA (N, CB,  DACC, CX, INCX, CY, INCY)
*DW =  DQDOTI (N, DB,  DACC, DX, INCX, DY, INCY)
*DW =  DQDOTA (N, DB,  DACC, DX, INCX, DY, INCY)
 ZW =  ZQDOTI (N, ZB, DZACC, ZX, INCX, ZY, INCY)
 ZW =  ZQDOTA (N, ZB, DZACC, ZX, INCX, ZY, INCY)
```

The variable DACC, a double precision array of length two, is used as a quadruple precision accumulator. DZACC, a double precision array of length four, is its complex analog. The function subprograms, with a name ending in I, initialize DACC to zero. All of the function subprograms then compute

$$\mathrm{DACC} + b + \sum_{i=1}^{N} x_i y_i$$

and store the result in DACC. The result, converted to the precision of the function, is also returned as the function value. If $N \le 0$, then the function subprograms return zero.

Hadamard Product

```
CALL SHPROD (N, SX, INCX, SY, INCY, SZ, INCZ)
CALL DHPROD (N, DX, INCX, DY, INCY, DZ, INCZ)
```

These subprograms set $z_i \leftarrow x_i y_i$ for $i = 1, 2, \ldots, N$. If $N \le 0$, then the subprograms return immediately.

Triple Inner Product

```
SW = SXYZ (N, SX, INCX, SY, INCY, SZ, INCZ)
DW = DXYZ (N, DX, INCX, DY, INCY, DZ, INCZ)
```

These function subprograms compute

$$\sum_{i=1}^{N} x_i y_i z_i$$

If $N \le 0$ then the subprograms return zero.

Sum of the Elements of a Vector

```
IW = ISUM (N, IX, INCX)
SW = SSUM (N, SX, INCX)
DW = DSUM (N, DX, INCX)
```

These function subprograms compute

$$\sum_{i=1}^{N} x_i$$

If $N \le 0$, then the subprograms return zero.

Sum of the Absolute Values of the Elements of a Vector

```
*SW = SASUM (N, SX, INCX)
*DW = DASUM (N, DX, INCX)
*SW = SCASUM (N, CX, INCX)
 DW = DZASUM (N, ZX, INCX)
```

The function subprograms `SASUM` and `DASUM` compute

$$\sum_{i=1}^{N} \left| x_i \right|$$

The function subprograms `SCASUM` and `DZASUM` compute

$$\sum_{i=1}^{N} \left[\left| \Re x_i \right| + \left| \Im x_i \right| \right]$$

If $N \le 0$, then the subprograms return zero. CAUTION: For `SCASUM` and `DZASUM`, the function subprogram returns a real value.

Euclidean or ℓ_2 Norm of a Vector

```
*SW = SNRM2  (N, SX, INCX)
*DW = DNRM2  (N, DX, INCX)
*SW = SCNRM2 (N, CX, INCX)
 DW = DZNRM2 (N, ZX, INCX)
```

These function subprograms compute

$$\left[\sum_{i=1}^{N} \left| x_i \right|^2 \right]^{1/2}$$

If $N \le 0$, then the subprograms return zero. CAUTION: For `SCNRM2` and `DZNRM2`, the function subprogram returns a real value.

Product of the Elements of a Vector

```
SW = SPRDCT (N, SX, INCX)
DW = DPRDCT (N, DX, INCX)
```

These function subprograms compute

$$\prod_{i=1}^{N} x_i$$

If $N \le 0$, then the subprograms return zero.

Index of Element Having Minimum Value

```
IW = IIMIN (N, IX, INCX)
IW = ISMIN (N, SX, INCX)
IW = IDMIN (N, DX, INCX)
```

These function subprograms compute the smallest index i such that $x_i = \min_{1 \le j \le N} x_j$. If $N \le 0$, then the subprograms return zero.

Index of Element Having Maximum Value

```
IW = IIMAX (N, IX, INCX)
IW = ISMAX (N, SX, INCX)
IW = IDMAX (N, DX, INCX)
```

These function subprograms compute the smallest index i such that $x_i = \max_{1 \le j \le N} x_j$. If $N \le 0$, then the subprograms return zero.

Index of Element Having Minimum Absolute Value

```
IW = ISAMIN (N, SX, INCX)
IW = IDAMIN (N, DX, INCX)
IW = ICAMIN (N, CX, INCX)
IW = IZAMIN (N, ZX, INCX)
```

The function subprograms `ISAMIN` and `IDAMIN` compute the smallest index i such that $|x_i| = \min_{1 \le j \le N} |x_j|$. The function subprograms `ICAMIN` and `IZAMIN` compute the smallest index i such that

$$\left|\Re x_i\right| + \left|\Im x_i\right| = \min_{1 \le j \le N}\left[\left|\Re x_j\right| + \left|\Im x_j\right|\right]$$

If $N \le 0$, then the subprograms return zero.

Index of Element Having Maximum Absolute Value

```
*IW = ISAMAX (N, SX, INCX)
*IW = IDAMAX (N, DX, INCX)
*IW = ICAMAX (N, CX, INCX)
 IW = IZAMAX (N, ZX, INCX)
```

The function subprograms `ISAMAX` and `IDAMAX` compute the smallest index i such that $|x_i| = \max_{1 \le j \le N} |x_j|$. The function subprograms `ICAMAX` and `IZAMAX` compute the smallest index i such that

$$\left|\Re x_i\right| + \left|\Im x_i\right| = \max_{1 \le j \le N}\left[\left|\Re x_j\right| + \left|\Im x_j\right|\right]$$

If $N \le 0$, then the subprograms return zero.

Construct a Givens Plane Rotation

```
*CALL SROTG (SA, SB, SC, SS)
*CALL DROTG (SA, SB, SC, SS)
```

Given the values *a* and *b*, these subprograms compute

$$c = \begin{cases} a/r & \text{if } r \neq 0 \\ 1 & \text{if } r = 0 \end{cases}$$

and

$$s = \begin{cases} b/r & \text{if } r \neq 0 \\ 1 & \text{if } r = 0 \end{cases}$$

where $r = \sigma(a^2 + b^2)^{1/2}$ and

$$\sigma = \begin{cases} \text{sign}(a) & \text{if } |a| > |b| \\ \text{sign}(b) & \text{otherwise} \end{cases}$$

Then, the values *c*, *s* and *r* satisfy the matrix equation

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

The introduction of σ is not essential to the computation of the Givens rotation matrix; but its use permits later stable reconstruction of *c* and *s* from just one stored number, an idea due to Stewart (1976). For this purpose, the subprogram also computes

$$z = \begin{cases} s & \text{if } |s| < c \text{ or } c = 0 \\ 1/c & \text{if } 0 < |c| \ \leq \ s \end{cases}$$

In addition to returning *c* and *s*, the subprograms return *r* overwriting *a*, and *z* overwriting *b*.

Reconstruction of *c* and *s* from *z* can be done as follows:

If $z = 1$, then set $c = 0$ and $s = 1$

If $|z| < 1$, then set

$$c = \sqrt{1 - z^2} \ \text{ and } \ s = z$$

If $|z| > 1$, then set

$$c = 1/z \ \text{ and } \ s = \sqrt{1 \text{-} c^2}$$

Apply a Plane Rotation

```
*CALL SROT (N, SX, INCX, SY, INCY, SC, SS)
*CALL DROT (N, DX, INCX, DY, INCY, DC, DS)
```

```
 CALL CSROT (N, CX, INCX, CY, INCY, SC, SS)
 CALL ZDROT (N, ZX, INCX, ZY, INCY, DC, DS)
```

These subprograms compute

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \leftarrow \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{for } i = 1, \ldots, N$$

If $N \le 0$, then the subprograms return immediately. CAUTION: For `CSROT` and `ZDROT`, the scalar quantities c and s are real, and x and y are complex.

Construct a Modified Givens Transformation

```
*CALL SROTMG (SD1, SD2, SX1, SY1, SPARAM)
*CALL DROTMG (DD1, DD2, DX1, DY1, DPARAM)
```

The input quantities d_1, d_2, x_1 and y_1 define a 2-vector $[w_1, z_1]^T$ by the following:

$$\begin{bmatrix} w_i \\ z_i \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ 0 & \sqrt{d_2} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

The subprograms determine the modified Givens rotation matrix H that transforms y_1, and thus, z_1 to zero. They also replace d_1, d_2 and x_1 with

$$\tilde{d}_1, \ \tilde{d}_2 \text{ and } \tilde{x}_1$$

respectively. That is,

$$\begin{bmatrix} \tilde{w}_1 \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{\tilde{d}_1} & 0 \\ 0 & \sqrt{\tilde{d}_2} \end{bmatrix} H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \sqrt{\tilde{d}_1} & 0 \\ 0 & \sqrt{\tilde{d}_2} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \tilde{x}_1 \\ 0 \end{bmatrix}$$

A representation of this matrix is stored in the array `SPARAM` or `DPARAM`. The form of the matrix H is flagged by `PARAM`(1).

`PARAM`(1) = 1. In this case,

$$\left|d_1 x_1^2\right| \le \left|d_2 y_1^2\right|$$

and

$$H = \begin{bmatrix} \text{PARAM(2)} & 1 \\ -1 & \text{PARAM(5)} \end{bmatrix}$$

The elements `PARAM`(3) and `PARAM`(4) are not changed.

`PARAM`(1) = 0. In this case,

$$\left|d_1 x_1^2\right| > \left|d_2 y_1^2\right|$$

and

$$H = \begin{bmatrix} 1 & \text{PARAM}(4) \\ \text{PARAM}(3) & 1 \end{bmatrix}$$

The elements PARAM(2) and PARAM(5) are not changed.

PARAM(1) = –1. In this case, rescaling was done and

$$H = \begin{bmatrix} \text{PARAM}(2) & \text{PARAM}(4) \\ \text{PARAM}(3) & \text{PARAM}(5) \end{bmatrix}$$

PARAM(1) = –2. In this case, $H = I$ where I is the identity matrix. The elements PARAM(2), PARAM(3), PARAM(4) and PARAM(5) are not changed.

The values of d_1, d_2 and x_1 are changed to represent the effect of the transformation. The quantity y_1, which would be zeroed by the transformation, is left unchanged.

The input value of d_1 should be nonnegative, but d_2 can be negative for the purpose of removing data from a least-squares problem.

See Lawson et al. (1979) for further details.

Apply a Modified Givens Transformation

```
*CALL SROTM (N, SX, INCX, SY, INCY, SPARAM)
*CALL DROTM (N, DX, INCX, DY, INCY, DPARAM)
 CALL CSROTM (N, CX, INCX, CY, INCY, SPARAM)
 CALL ZDROTM (N, ZX, INCX, ZY, INCY, DPARAM)
```

If PARAM(1) = 1.0, then these subprograms compute

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \leftarrow \begin{bmatrix} \text{PARAM}(2) & 1 \\ -1 & \text{PARAM}(5) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{ for } i = 1, \ldots, N$$

If PARAM(1) = 0.0, then the subprograms compute

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \leftarrow \begin{bmatrix} 1 & \text{PARAM}(4) \\ \text{PARAM}(3) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{ for } i = 1, \ldots, N$$

If PARAM(1) = –1.0, then the subprograms compute

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \leftarrow \begin{bmatrix} \text{PARAM}(2) & \text{PARAM}(4) \\ \text{PARAM}(3) & \text{PARAM}(5) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \text{ for } i = 1, \ldots, N$$

If $N \le 0$ or if PARAM(1) = –2.0, then the subprograms return immediately. CAUTION: For CSROTM and ZDROTM, the scalar quantities PARAM(*) are real and x and y are complex.

Programming Notes for Level 2 and Level 3 BLAS

For definitions of the matrix data structures used in the discussion below, see page 1206. The Level 2 and Level 3 BLAS, like the Level 1 BLAS, do not follow the IMSL naming conventions. Instead, the names consist of a prefix of one of the letters "S," "D," "C" or "Z." Next is a root name denoting the kind of matrix. This is followed by a suffix indicating the type of the operation.[1] The prefix denotes the type of operation according to the following table:

S	Real	C	Complex
D	Double	Z	Double Complex

The root names for the kind of matrix:

GE	GEneral	GB	General Band
SY	SYmmetric	SB	Symmetric Band
HE	HErmitian	HB	Hermitian Band
TR	TRiangular	TB	Triangular Band

The suffixes for the type of operation:

MV	Matrix-Vector Product	SV	Solve for Vector
R	Rank-One Update		
RU	Rank-One Update, Unconjugated	RC	Rank-One Update, Conjugated
R2	Rank-Two Update		
MM	Matrix-Multiply	SM	Symmetric Matrix Multiply
RK	Rank-K Update	R2K	Rank 2K Update

[1] IMSL does not support the *Packed Symmetric, Packed-Hermitian,* or *Packed-Triangular* data structures, with respective root names SP, HP or TP, nor any extended precision versions of the Level 2 BLAS.

The specifications of the operations are provided by subprogram arguments of CHARACTER*1 data type. Both lower and upper case of the letter have the same meaning:

TRANS, TRANSA, TRANSB	'N'	No Transpose
	'T'	Transpose
	'C'	Conjugage and Transpose
UPLO	'L'	Lower Triangular
	'U'	Upper Triangular
DIAG	'N'	Non-unit Triangular
	'U'	Unit Triangular
SIDE	'L'	Multiply "*A*" Matrix on Left side or

'R' Right side of the "*B*" matrix

Note: See the "Triangular Mode" section in the Reference Material (page 1208) for definitions of these terms.

Descriptions of the Level 2 and Level 3 BLAS

The subprograms for Level 2 and Level 3 BLAS that perform operations involving the expression βy or βC do not require that the contents of y or C be defined when $\beta = 0$. In that case, the expression βy or βC is defined to be zero. Note that for the `_GEMV` and `_GBMV` subprograms, the dimensions of the vectors x and y are implied by the specification of the operation. If `TRANS` = `'N'` , the dimension of y is m; if `TRANS` = `'T'` or = `'C'`, the dimension of y is n. The Level 2 and Level 3 BLAS are summarized in Table 9.2. This table also lists the page numbers where the subprograms are described in more detail.

Specification of the Level 2 BLAS

Type and dimension for variables occurring in the subprogram specifications are

```
INTEGER           INCX, INCY, NCODA, NLCA, NUCA, LDA, M, N
CHARACTER*1       DIAG, TRANS, UPLO

REAL              SALPHA, SBETA, SX(*), SY(*), SA(LDA,*)
DOUBLE PRECISION  DALPHA, DBETA, DX(*), DY(*), DA(LDA,*)
COMPLEX           CALPHA, CBETA, CX(*), CY(*), CA(LDA,*)
DOUBLE COMPLEX    ZALPHA, ZBETA, ZX(*), ZY(*), ZA(LDA,*)
```

There is a lower bound on the leading dimension `LDA`. It must be ≥ the number of rows in the matrix that is contained in this array. Vector arguments have an increment parameter that specifies the storage space or stride between elements. The correspondence between the vector x, y and the arguments `SX`, `SY` and `INCX`, `INCY` is

$$x_i = \begin{cases} \mathrm{SX}((\mathrm{I}-1)*\mathrm{INCX}+1) & \text{if INCX} > 0 \\ \mathrm{SX}((\mathrm{I}-\mathrm{N})*\mathrm{INCX}+1) & \text{if INCX} < 0 \end{cases}$$

$$y_i = \begin{cases} \mathrm{SY}((\mathrm{I}-1)*\mathrm{INCY}+1) & \text{if INCY} > 0 \\ \mathrm{SY}((\mathrm{I}-\mathrm{N})*\mathrm{INCY}+1) & \text{if INCY} < 0 \end{cases}$$

In the Level 2 BLAS, only nonzero values of `INCX`, `INCY` are allowed for operations that have vector arguments. The Level 3 BLAS do not refer to `INCX`, `INCY`.

Specification of the Level 3 BLAS

Type and dimension for variables occurring in the subprogram specifications are

```
INTEGER           K, LDA, LDB, LDC, M, N
CHARACTER*1       DIAG, TRANS, TRANSA, TRANSB, SIDE, UPLO
REAL              SALPHA, SBETA, SA(LDA,*), SB(LDB,*),
                  SC(LDC,*)
DOUBLE PRECISION  DALPHA, DBETA, DA(LDA,*), DB(LDB,*),
                  DC(LDC,*)
COMPLEX           CALPHA, CBETA, CA(LDA,*), CB(LDB,*),
                  CC(LDC,*)
DOUBLE COMPLEX    ZALPHA, ZBETA, ZA(LDA,*), ZB(LDB,*),
                  ZC(LDC,*)
```

Each of the integers K, M, N must be ≥ 0. It is an error if any of them are < 0. If any of them are = 0, the subprograms return immediately. There are lower bounds on the leading dimensions LDA, LDB, LDC. Each must be ≥ the number of rows in the matrix that is contained in this array.

Table 9.2: Level 2 and 3 Basic Linear Algebra Subprograms

Operation	Real	Double	Complex	Double Complex	Pg.
Matrix-Vector Multiply, General	SGEMV	DGEMV	CGEMV	ZGEMV	1049
Matrix-Vector Multiply, Banded	SGBMV	DGBMV	CGBMV	ZGBMV	1049
Matrix-Vector Multiply, Hermitian			CHEMV	ZHEMV	1050
Matrix-Vector Multiply, Hermitian and Banded			CHBMV	ZHBMV	1050
Matrix-Vector Multiply Symmetric and Real	SSYMV	DSYMV			1050
Matrix-Vector Multiply, Symmetric and Banded	SSBMV	DSBMV			1050
Matrix-Vector Multiply, Triangular	STRMV	DTRMV	CTRMV	ZTRMV	1050
Matrix-Vector Multiply, Triangular and Banded	STBMV	DTBMV	CTBMV	ZTBMV	1051
Matrix-Vector Solve, Triangular	STRSV	DTRSV	CTRSV	ZTRSV	1051
Matrix-Vector Solve, Triangular and Banded	STBSV	DTBSV	CTBSV	ZTBSV	1051
Rank-One Matrix Update, General and Real	SGER	DGER			1052
Rank-One Matrix Update, General, Complex and Transpose			CGERU	ZGERU	1052
Rank-One Matrix Update, General, Complex, and Conjugate Transpose			CGERC	ZGERC	1052
Rank-One Matrix Update, Hermitian and Conjugate Transpose			CHER	ZHER	1052
Rank-Two Matrix Update, Hermitian and Conjugate Transpose			CHER2	ZHER2	1052
Rank-One Matrix Update, Symmetric and Real	SSYR	DSYR			1053

Operation	Real	Double	Complex	Double Complex	Pg.
Rank-Two Matrix Update, Symmetric and Real	SSYR2	DSYR2			1053
Matrix--Matrix Multiply, General	SGEMM	DGEMM	CGEMM	ZGEMM	1053
Matrix-Matrix Multiply, Symmetric	SSYMM	DSYMM	CSYMM	ZSYMM	1053
Matrix-Matrix Multiply, Hermitian			CHEMM	ZHEMM	1054
Rank - *k* Update, Symmetric	SSYRK	DSYRK	CSYRK	ZSYRK	1054
Rank - *k* Update, Hermitian			CHERK	ZHERK	1054
Rank - 2*k* Update, Symmetric	SSYR2K	DSYR2K	CSYR2K	ZSYR2K	1055
Rank - 2*k* Update, Hermitian			CHER2K	ZHER2K	1055
Matrix-Matrix Multiply, Triangular	STRMM	DTRMM	CTRMM	ZTRMM	1055
Matrix-Matrix solve, Triangular	STRSM	DTRSM	CTRSM	ZTRSM	1056

Matrix–Vector Multiply, General

```
CALL SGEMV (TRANS, M, N, SALPHA, SA, LDA, SX, INCX,
            SBETA,SY, INCY)
CALL DGEMV (TRANS, M, N, DALPHA, DA, LDA, DX, INCX, DBETA,
            DY, INCY)
CALL CGEMV (TRANS, M, N, CALPHA, CA, LDA, CX, INCX, CBETA,
            CY, INCY)
CALL ZGEMV (TRANS, M, N, ZALPHA, ZA, LDA, ZX, INCX, ZBETA,
            ZY, INCY)
```

For all data types, A is an $M \times N$ matrix. These subprograms set y to one of the expressions: $y \leftarrow \alpha Ax + \beta y$, $y \leftarrow \alpha A^T x + \beta y$, or for complex data,

$$y \leftarrow \alpha \overline{A}^T + \beta y$$

The character flag TRANS determines the operation.

Matrix–Vector Multiply, Banded

```
CALL SGBMV (TRANS, M, N, NLCA, NUCA SALPHA, SA, LDA, SX,
            INCX, SBETA,SY, INCY)
CALL DGBMV (TRANS, M, N, NLCA, NUCA DALPHA, DA, LDA, DX,
            INCX, DBETA,DY, INCY)
CALL CGBMV (TRANS, M, N, NLCA, NUCA CALPHA, CA, LDA, CX,
            INCX, CBETA,CY, INCY)
CALL ZGBMV (TRANS, M, N, NLCA, NUCA ZALPHA, ZA, LDA, ZX,
            INCX, ZBETA,ZY, INCY)
```

For all data types, A is an $M \times N$ matrix with NLCA lower codiagonals and NUCA upper codiagonals. The matrix is stored in band storage mode. These subprograms set y to one of the expressions: $y \leftarrow \alpha Ax + \beta y$, $y \leftarrow \alpha A^T x + \beta y$, or for complex data,

$$y \leftarrow \alpha \bar{A}^T x + \beta y$$

The character flag `TRANS` determines the operation.

Matrix-Vector Multiply, Hermitian

```
CALL CHEMV (UPLO, N, CALPHA, CA, LDA, CX, INCX, CBETA,
            CY,INCY)
CALL ZHEMV (UPLO, N, ZALPHA, ZA, LDA, ZX, INCX, ZBETA, ZY,
            INCY)
```

For all data types, A is an $N \times N$ matrix. These subprograms set $y \leftarrow \alpha Ax + \beta y$ where A is an Hermitian matrix. The matrix A is either referenced using its upper or lower triangular part. The character flag `UPLO` determines the part used.

Matrix-Vector Multiply, Hermitian and Banded

```
CALL CHBMV (UPLO, N, NCODA, CALPHA, CA, LDA, CX, INCX,
            CBETA, CY,INCY)
CALL ZHBMV (UPLO, N, NCODA, ZALPHA, ZA, LDA, ZX, INCX,
            ZBETA, ZY,INCY)
```

For all data types, A is an $N \times N$ matrix with `NCODA` codiagonals. The matrix is stored in band Hermitian storage mode. These subprograms set $y \leftarrow \alpha Ax + \beta y$. The matrix A is either referenced using its upper or lower triangular part. The character flag `UPLO` determines the part used.

Matrix-Vector Multiply, Symmetric and Real

```
CALL SSYMV (UPLO, N, SALPHA, SA, LDA, SX, INCX, SBETA, SY,
            INCY)
CALL DSYMV (UPLO, N, DALPHA, DA, LDA, DX, INCX, DBETA, DY,
            INCY)
```

For all data types, A is an $N \times N$ matrix. These subprograms set $y \leftarrow \alpha Ax + \beta y$ where A is a symmetric matrix. The matrix A is either referenced using its upper or lower triangular part. The character flag `UPLO` determines the part used.

Matrix-Vector Multiply, Symmetric and Banded

```
CALL SSBMV (UPLO, N, NCODA, SALPHA, SA, LDA, SX, INCX,
            SBETA, SY,INCY)
CALL DSBMV (UPLO, N, NCODA, DALPHA, DA, LDA, DX, INCX,
            DBETA, DY,INCY)
```

For all data types, A is an $N \times N$ matrix with `NCODA` codiagonals. The matrix is stored in band symmetric storage mode. These subprograms set $y \leftarrow \alpha Ax + \beta y$. The matrix A is either referenced using its upper or lower triangular part. The character flag `UPLO` determines the part used.

Matrix-Vector Multiply, Triangular

```
CALL STRMV (UPLO, TRANS, DIAG, N, SA, LDA, SX, INCX)
CALL DTRMV (UPLO, TRANS, DIAG, N, DA, LDA, DX, INCX)
CALL CTRMV (UPLO, TRANS, DIAG, N, CA, LDA, CX, INCX)
CALL ZTRMV (UPLO, TRANS, DIAG, N, ZA, LDA, ZX, INCX)
```

For all data types, A is an $N \times N$ triangular matrix. These subprograms set x to one of the expressions: $x \leftarrow Ax$, $x \leftarrow A^T x$, or for complex data,

$$x \leftarrow \overline{A}^T x$$

The matrix A is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `UPLO`, `TRANS`, and `DIAG` determine the part of the matrix used and the operation performed.

Matrix-Vector Multiply, Triangular and Banded

```
CALL STBMV (UPLO, TRANS, DIAG, N, NCODA, SA, LDA, SX, INCX)
CALL DTBMV (UPLO, TRANS, DIAG, N, NCODA, DA, LDA, DX, INCX)
CALL CTBMV (UPLO, TRANS, DIAG, N, NCODA, CA, LDA, CX, INCX)
CALL ZTBMV (UPLO, TRANS, DIAG, N, NCODA, ZA, LDA, ZX, INCX)
```

For all data types, A is an $N \times N$ matrix with `NCODA` codiagonals. The matrix is stored in band triangular storage mode. These subprograms set x to one of the expressions: $x \leftarrow Ax$, $x \leftarrow A^T x$, or for complex data,

$$x \leftarrow \overline{A}^T x$$

The matrix A is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `UPLO`, `TRANS`, and `DIAG` determine the part of the matrix used and the operation performed.

Matrix-Vector Solve, Triangular

```
CALL STRSV (UPLO, TRANS, DIAG, N, SA, LDA, SX, INCX)
CALL DTRSV (UPLO, TRANS, DIAG, N, DA, LDA, DX, INCX)
CALL CTRSV (UPLO, TRANS, DIAG, N, CA, LDA, CX, INCX)
CALL ZTRSV (UPLO, TRANS, DIAG, N, ZA, LDA, ZX, INCX)
```

For all data types, A is an $N \times N$ triangular matrix. These subprograms solve x for one of the expressions: $x \leftarrow A^{-1} x$, $x \leftarrow (A^{-1})^T x$, or for complex data,

$$x \leftarrow \left(\overline{A}^T\right)^{-1} x$$

The matrix A is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `UPLO`, `TRANS`, and `DIAG` determine the part of the matrix used and the operation performed.

Matrix-Vector Solve, Triangular and Banded

```
CALL STBSV (UPLO, TRANS, DIAG, N, NCODA, SA, LDA, SX, INCX)
CALL DTBSV (UPLO, TRANS, DIAG, N, NCODA, DA, LDA, DX, INCX)
CALL CTBSV (UPLO, TRANS, DIAG, N, NCODA, CA, LDA, CX, INCX)
CALL ZTBSV (UPLO, TRANS, DIAG, N, NCODA, ZA, LDA, ZX, INCX)
```

For all data types, A is an $N \times N$ triangular matrix with `NCODA` codiagonals. The matrix is stored in band triangular storage mode. These subprograms solve x for one of the expressions: $x \leftarrow A^{-1} x$, $x \leftarrow (A^{T})^{-1} x$, or for complex data,

$$x \leftarrow \left(\overline{A}^{T}\right)^{-1} x$$

The matrix A is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `UPLO`, `TRANS`, and `DIAG` determine the part of the matrix used and the operation performed.

Rank-One Matrix Update, General and Real

```
CALL SGER (M, N, SALPHA, SX, INCX, SY, INCY, SA, LDA)
CALL DGER (M, N, DALPHA, DX, INCX, DY, INCY, DA, LDA
```

For all data types, A is an $M \times N$ matrix. These subprograms set $A \leftarrow A + \alpha xy^{T}$.

Rank-One Matrix Update, General, Complex, and Transpose

```
CALL CGERU (M, N, CALPHA, CX, INCX, CY, INCY, CA, LDA)
CALL ZGERU (M, N, ZALPHA, ZX, INCX, ZY, INCY, ZA, LDA)
```

For all data types, A is an $M \times N$ matrix. These subprograms set $A \leftarrow A + \alpha xy^{T}$.

Rank-One Matrix Update, General, Complex, and Conjugate Transpose

```
CALL CGERC (M, N, CALPHA, CX, INCX, CY, INCY, CA, LDA)
CALL ZGERC (M, N, ZALPHA, ZX, INCX, ZY, INCY, ZA, LDA)
```

For all data types, A is an $M \times N$ matrix. These subprograms set

$$A \leftarrow A + \alpha x\overline{y}^{T}$$

Rank-One Matrix Update, Hermitian and Conjugate Transpose

```
CALL CHER (UPLO, N, SALPHA, CX, INCX, CA, LDA)
CALL ZHER (UPLO, N, DALPHA, ZX, INCX, ZA, LDA)
```

For all data types, A is an $N \times N$ matrix. These subprograms set

$$A \leftarrow A + \alpha x\overline{x}^{T}$$

where A is Hermitian. The matrix A is either referenced by its upper or lower triangular part. The character flag `UPLO` determines the part used. CAUTION: Notice the scalar parameter α is real, and the data in the matrix and vector are complex.

Rank-Two Matrix Update, Hermitian and Conjugate Transpose

```
CALL CHER2 (UPLO, N, CALPHA, CX, INCX, CY, INCY, CA, LDA)
CALL ZHER2 (UPLO, N, ZALPHA, ZX, INCX, ZY, INCY, ZA, LDA)
```

For all data types, A is an $N \times N$ matrix. These subprograms set

$$A \leftarrow A + \alpha x\bar{y}^T + \bar{\alpha} y\bar{x}^T$$

where A is an Hermitian matrix. The matrix A is either referenced by its upper or lower triangular part. The character flag `UPLO` determines the part used.

Rank-One Matrix Update, Symmetric and Real

```
CALL SSYR (UPLO, N, SALPHA, SX, INCX, SA, LDA)
CALL DSYR (UPLO, N, DALPHA, DX, INCX, DA, LDA)
```

For all data types, A is an $N \times N$ matrix. These subprograms set $A \leftarrow A + \alpha xx^T$ where A is a symmetric matrix. The matrix A is either referenced by its upper or lower triangular part. The character flag `UPLO` determines the part used.

Rank-Two Matrix Update, Symmetric and Real

```
CALL SSYR2 (UPLO, N, SALPHA, SX, INCX, SY, INCY, SA, LDA)
CALL DSYR2 (UPLO, N, DALPHA, DX, INCX, DY, INCY, DA, LDA)
```

For all data types, A is an $N \times N$ matrix. These subprograms set $A \leftarrow A + \alpha xy^T +$ αyx^T where A is a symmetric matrix. The matrix A is referenced by its upper or lower triangular part. The character flag `UPLO` determines the part used.

Matrix-Matrix Multiply, General

```
CALL SGEMM (TRANSA, TRANSB, M, N, K, SALPHA, SA, LDA, SB,
            LDB, SBETA, SC, LDC)
CALL DGEMM (TRANSA, TRANSB, M, N, K, DALPHA, DA, LDA, DB,
            LDB, DBETA, DC, LDC)
CALL CGEMM (TRANSA, TRANSB, M, N, K, CALPHA, CA, LDA, CB,
            LDB, CBETA, CC, LDC)
CALL ZGEMM (TRANSA, TRANSB, M, N, K, ZALPHA, ZA, LDA, ZB,
            LDB, ZBETA, ZC, LDC)
```

For all data types, these subprograms set $C_{M \times N}$ to one of the expressions:

$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha A^T B + \beta C, C \leftarrow \alpha AB^T + \beta C, C \leftarrow \alpha A^T B^T + \beta C,$

or for complex data, $C \leftarrow \alpha A\bar{B}^T + \beta C, C \leftarrow \alpha \bar{A}^T B + \beta C, C \leftarrow \alpha A^T \bar{B}^T + \beta C,$

$C \leftarrow \alpha \bar{A}^T B^T + \beta C, C \leftarrow \alpha \bar{A}^T \bar{B}^T + \beta C$

The character flags `TRANSA` and `TRANSB` determine the operation to be performed. Each matrix product has dimensions that follow from the fact that C has dimension $M \times N$.

Matrix-Matrix Multiply, Symmetric

```
CALL SSYMM (SIDE, UPLO, M, N, SALPHA, SA, LDA, SB, LDB,
            SBETA, SC, LDC)
CALL DSYMM (SIDE, UPLO, M, N, DALPHA, DA, LDA, DB, LDB,
            DBETA, DC, LDC)
CALL CSYMM (SIDE, UPLO, M, N, CALPHA, CA, LDA, CB, LDB,
            CBETA, CC, LDC)
```

```
CALL ZSYMM (SIDE, UPLO, M, N, ZALPHA, ZA, LDA, ZB, LDB,
            ZBETA, ZC, LDC)
```

For all data types, these subprograms set $C_{M \times N}$ to one of the expressions: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is a symmetric matrix. The matrix A is referenced either by its upper or lower triangular part. The character flags `SIDE` and `UPLO` determine the part of the matrix used and the operation performed.

Matrix-Matrix Multiply, Hermitian

```
CALL CHEMM (SIDE, UPLO, M, N, CALPHA, CA, LDA, CB, LDB,
            CBETA, CC, LDC)
CALL ZHEMM (SIDE, UPLO, M, N, ZALPHA, ZA, LDA, ZB, LDB,
            ZBETA, ZC, LDC)
```

For all data types, these subprograms set $C_{M \times N}$ to one of the expressions: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is an Hermitian matrix. The matrix A is referenced either by its upper or lower triangular part. The character flags `SIDE` and `UPLO` determine the part of the matrix used and the operation performed.

Rank-*k* Update, Symmetric

```
CALL SSYRK (UPLO, TRANS, N, K, SALPHA, SA, LDA, SBETA, SC,
            LDC)
CALL DSYRK (UPLO, TRANS, N, K, DALPHA, DA, LDA, DBETA, DC,
            LDC)
CALL CSYRK (UPLO, TRANS, N, K, CALPHA, CA, LDA, CBETA, CC,
            LDC)
CALL ZSYRK (UPLO, TRANS, N, K, ZALPHA, ZA, LDA, ZBETA, ZC,
            LDC)
```

For all data types, these subprograms set $C_{M \times N}$ to one of the expressions: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$. The matrix C is referenced either by its upper or lower triangular part. The character flags `UPLO` and `TRANS` determine the part of the matrix used and the operation performed. In subprogram `CSYRK` and `ZSYRK`, only values `'N'` or `'T'` are allowed for `TRANS`; `'C'` is not acceptable.

Rank-*k* Update, Hermitian

```
CALL CHERK (UPLO, TRANS, N, K, SALPHA, CA, LDA, SBETA, CC,
            LDC)
CALL ZHERK (UPLO, TRANS, N, K, DALPHA, ZA, LDA, DBETA, ZC,
            LDC)
```

For all data types, these subprograms set $C_{N \times N}$ to one of the expressions:

$$C \leftarrow \alpha A\overline{A}^T + \beta C \text{ or } C \leftarrow \alpha \overline{A}^T A + \beta C$$

The matrix C is referenced either by its upper or lower triangular part. The character flags `UPLO` and `TRANS` determine the part of the matrix used and the operation performed. CAUTION: Notice the scalar parameters α and β are real, and the data in the matrices are complex. Only values `'N'` or `'C'` are allowed for `TRANS`; `'T'` is not acceptable.

Rank-2*k* Update, Symmetric

```
CALL SSYR2K (UPLO, TRANS, N, K, SALPHA, SA, LDA, SB, LDB,
             SBETA, SC, LDC)
CALL DSYR2K (UPLO, TRANS, N, K, DALPHA, DA, LDA, DB, LDB,
             DBETA, DC, LDC)
CALL CSYR2K (UPLO, TRANS, N, K, CALPHA, CA, LDA, CB, LDB,
             CBETA, CC, LDC)
CALL ZSYR2K (UPLO, TRANS, N, K, ZALPHA, ZA, LDA, ZB, LDB,
             ZBETA, ZC, LDC)
```

For all data types, these subprograms set $C_{N \times N}$ to one of the expressions:

$$C \leftarrow \alpha AB^T + \alpha\beta A^T + \beta\mathrm{C} \text{ or } C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$$

The matrix *C* is referenced either by its upper or lower triangular part. The character flags `UPLO` and `TRANS` determine the part of the matrix used and the operation performed. In subprogram `CSYR2K` and `ZSYR2K`, only values `'N'`or `'T'` are allowed for `TRANS`; `'C'`is not acceptable.

Rank-2*k* Update, Hermitian

```
CALL CHER2K (UPLO, TRANS, N, K, CALPHA, CA, LDA, CB, LDB,
             SBETA, CC, LDC)
CALL ZHER2K (UPLO, TRANS, N, K, ZALPHA, ZA, LDA, ZB, LDB,
             DBETA, ZC, LDC)
```

For all data types, these subprograms set $C_{N \times N}$ to one of the expressions:

$$C \leftarrow \alpha A\overline{B}^T + \overline{\alpha} B\overline{A}^T + \beta\mathrm{C} \text{ or } C \leftarrow \alpha \overline{A}^T B + \overline{\alpha}\overline{B}^T A + \beta C$$

The matrix *C* is referenced either by its upper or lower triangular part. The character flags `UPLO` and `TRANS` determine the part of the matrix used and the operation performed. CAUTION: Notice the scalar parameter β is real, and the data in the matrices are complex. In subprogram `CHER2K` and `ZHER2K`, only values `'N'` or `'C'`are allowed for `TRANS`; `'T'`is not acceptable.

Matrix-Matrix Multiply, Triangular

```
CALL STRMM (SIDE, UPLO, TRANSA, DIAG, M, N, SALPHA, SA,
            LDA, SB, LDB)
CALL DTRMM (SIDE, UPLO, TRANSA, DIAG, M, N, DALPHA, DA,
            LDA, DB, LDB)
CALL CTRMM (SIDE, UPLO, TRANSA, DIAG, M, N, CALPHA, CA,
            LDA, CB,LDB)
CALL ZTRMM (SIDE, UPLO, TRANSA, DIAG, M, N, ZALPHA, ZA,
            LDA, ZB, LDB)
```

For all data types, these subprograms set $B_{M \times N}$ to one of the_expressions:

$$B \leftarrow \alpha AB,\ B \leftarrow \alpha A^T B,\ B \leftarrow \alpha BA,\ B \leftarrow \alpha BA^T,$$

$$\text{or for complex data, } B \leftarrow \alpha \overline{A}^T B, \text{ or } B \leftarrow \alpha B\overline{A}^T$$

where *A* is a triangular matrix. The matrix *A* is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `SIDE`, `UPLO`, `TRANSA`, and `DIAG` determine the part of the matrix used and the operation performed.

Matrix-Matrix Solve, Triangular

```
CALL STRSM (SIDE, UPLO, TRANSA, DIAG, M, N, SALPHA, SA,
            LDA, SB, LDB)
CALL DTRSM (SIDE, UPLO, TRANSA, DIAG, M, N, DALPHA, DA,
            LDA, DB, LDB)
CALL CTRSM (SIDE, UPLO, TRANSA, DIAG, M, N, CALPHA, CA,
            LDA, CB, LDB)
CALL ZTRSM (SIDE, UPLO, TRANSA, DIAG, M, N, ZALPHA, ZA,
            LDA, ZB, LDB)
```

For all data types, these subprograms set $B_{M \times N}$ to one of the expressions:

$$B \leftarrow \alpha A^{-1}B,\ B \leftarrow \alpha BA^{-1},\ B \leftarrow \alpha\left(A^{-1}\right)^{\mathrm{T}}B,\ B \leftarrow \alpha B\left(A^{-1}\right)^{T},$$

$$\text{or for complex data, } B \leftarrow \alpha\left(\overline{A}^{T}\right)^{-1}B, \text{ or } B \leftarrow \alpha B\left(\overline{A}^{T}\right)^{-1}$$

where *A* is a triangular matrix. The matrix *A* is either referenced using its upper or lower triangular part and is unit or nonunit triangular. The character flags `SIDE`, `UPLO`, `TRANSA`, and `DIAG` determine the part of the matrix used and the operation performed.

Other Matrix/Vector Operations

This section describes a set of routines for matrix/vector operations. The matrix copy and conversion routines are summarized by the following table:

From	**To**			
	Real General	**Complex General**	**Real Band**	**Complex Band**
Real General	`CRGRG` p. 1058	`CRGCG` p. 1068	`CRGRB` p. 1063	
Complex General		`CCGCG` p. 1059		`CCGCB` p. 1065
Real Band	`CRBRG` p. 1064		`CRBRB` p. 1060	`CRBCB` p. 1070
Complex Band		`CCBCG` p. 1066		`CCBCB` p. 1061
Symmetric Full	`CSFRG` p. 1071			
Hermitian Full		`CHFCG` p. 1072		

Symmetric Band			CSBRB p. 1074	
Hermitian Band				CHBCB p. 1075

The matrix multiplication routines are summarized as follows:

AB	***A***			
B	**Real Rect.**	**Complex Rect.**	**Real Band**	**Complex Band**
Real Rectangular	`MRRRR` p. 1082			
Complex Rect.		`MCRCR` p. 1084		
Vector	`MURRV` p. 1090	`MUCRV` p. 1093	`MURBV` p. 1091	`MUCBV` p. 1094

The matrix norm routines are summarized as follows:

||*A*||	**Real Rectangular**	**Real Band**	**Complex Band**
∞-norm	`NRIRR` p. 1099		
1-norm	`NR1RR` p. 1100	`NR1RB` p. 1103	`NR1CB` p. 1104
Frobenius	`NR2RR` p. 1101		

CRGRG/DCRGRG (Single/Double precision)

Copy a real general matrix.

Usage

`CALL CRGRG (N, A, LDA, B, LDB)`

Arguments

N — Order of the matrices. (Input)

A — Matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Matrix of order `N` containing a copy of `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

Algorithm

The routine `CRGRG` copies the real $N \times N$ general matrix A into the real $N \times N$ general matrix B.

Example

A real 3 × 3 general matrix is copied into another real 3 × 3 general matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N
      PARAMETER  (LDA=3, LDB=3, N=3)
C
      REAL       A(LDA,N), B(LDB,N)
      EXTERNAL   CRGRG, WRRRN
C                                  Set values for  A
C                                  A = (   0.0   1.0   1.0  )
C                                      (  -1.0   0.0   1.0  )
C                                      (  -1.0  -1.0   0.0  )
C
      DATA A/0.0, 2* - 1.0, 1.0, 0.0, -1.0, 2*1.0, 0.0/
C                                  Copy real matrix A to real matrix B
      CALL CRGRG (N, A, LDA, B, LDB)
C                                  Print results
      CALL WRRRN (’B’, N, N, B, LDB, 0)
      END
```

Output

```
             B
        1        2        3
1   0.000    1.000    1.000
2  -1.000    0.000    1.000
3  -1.000   -1.000    0.000
```

CCGCG/DCCGCG (Single/Double precision)

Copy a complex general matrix.

Usage

`CALL CCGCG (N, A, LDA, B, LDB)`

Arguments

N — Order of the matrices `A` and `B`.   (Input)

A — Complex matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

B — Complex matrix of order `N` containing a copy of `A`.   (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine `CCGCG` copies the complex *N* × *N* general matrix *A* into the complex *N* × *N* general matrix *B*.

Example

A complex 3 × 3 general matrix is copied into another complex 3 × 3 general matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N
      PARAMETER  (LDA=3, LDB=3, N=3)
C
      COMPLEX    A(LDA,N), B(LDB,N)
      EXTERNAL   CCGCG, WRCRN
C                            Set values for  A
C                            A = (  0.0+0.0i  1.0+1.0i  1.0+1.0i  )
C                                ( -1.0-1.0i  0.0+0.0i  1.0+1.0i  )
C                                ( -1.0-1.0i -1.0-1.0i  0.0+0.0i  )
C
      DATA A/(0.0,0.0), 2*(-1.0,-1.0), (1.0,1.0), (0.0,0.0),
     &     (-1.0,-1.0), 2*(1.0,1.0), (0.0,0.0)/
C                                  Copy matrix A to matrix B
      CALL CCGCG (N, A, LDA, B, LDB)
C                                  Print results
      CALL WRCRN ('B', N, N, B, LDB, 0)
      END
```

Output

```
                          B
                1                2                3
1  ( 0.000, 0.000)  ( 1.000, 1.000)  ( 1.000, 1.000)
2  (-1.000,-1.000)  ( 0.000, 0.000)  ( 1.000, 1.000)
3  (-1.000,-1.000)  (-1.000,-1.000)  ( 0.000, 0.000)
```

CRBRB/DCRBRB (Single/Double precision)

Copy a real band matrix stored in band storage mode.

Usage

CALL CRBRB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)

Arguments

N — Order of the matrices A and B.   (Input)

A — Real band matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals in A.   (Input)

NUCA — Number of upper codiagonals in A.   (Input)

B — Real band matrix of order N containing a copy of A.   (Output)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

NLCB — Number of lower codiagonals in B. (Input)
NLCB must be at least as large as NLCA.

NUCB — Number of upper codiagonals in B. (Input)
NUCB must be at least as large as NUCA.

Algorithm

The routine CRBRB copies the real band matrix *A* in band storage mode into the real band matrix *B* in band storage mode.

Example

A real band matrix of order 3, in band storage mode with one upper codiagonal, and one lower codiagonal is copied into another real band matrix also in band storage mode.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLCA, NLCB, NUCA, NUCB
      PARAMETER  (LDA=3, LDB=3, N=3, NLCA=1, NLCB=1, NUCA=1, NUCB=1)
C
      REAL       A(LDA,N), B(LDB,N)
      EXTERNAL   CRBRB, WRRRN
C                                  Set values for  A (in band mode)
C                                  A = (  0.0  1.0   1.0  )
C                                      (  1.0  1.0   1.0  )
C                                      (  1.0  1.0   0.0  )
C
      DATA A/0.0, 7*1.0, 0.0/
C                                  Copy A to B
      CALL CRBRB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)
C                                  Print results
      CALL WRRRN (’B’, NUCB+NLCB+1, N, B, LDB, 0)
      END
```

Output

```
             B
        1        2        3
1   0.000    1.000    1.000
2   1.000    1.000    1.000
3   1.000    1.000    0.000
```

CCBCB/DCCBCB (Single/Double precision)

Copy a complex band matrix stored in complex band storage mode.

Usage

CALL CCBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)

Arguments

N — Order of the matrices A and B. (Input)

A — Complex band matrix of order N. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals in A. (Input)

NUCA — Number of upper codiagonals in A. (Input)

B — Complex matrix of order N containing a copy of A. (Output)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program. (Input)

NLCB — Number of lower codiagonals in B. (Input)
NLCB must be at least as large as NLCA.

NUCB — Number of upper codiagonals in B. (Input)
NUCB must be at least as large as NUCA.

Algorithm

The routine CCBCB copies the complex band matrix *A* in band storage mode into the complex band matrix *B* in band storage mode.

Example

A complex band matrix of order 3 in band storage mode with one upper codiagonal and one lower codiagonal is copied into another complex band matrix in band storage mode.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLCA, NLCB, NUCA, NUCB
      PARAMETER  (LDA=3, LDB=3, N=3, NLCA=1, NLCB=1, NUCA=1, NUCB=1)
C
      COMPLEX    A(LDA,N), B(LDB,N)
      EXTERNAL   CCBCB, WRCRN
C                        Set values for  A (in band mode)
C                        A = (  0.0+0.0i  1.0+1.0i  1.0+1.0i  )
C                            (  1.0+1.0i  1.0+1.0i  1.0+1.0i  )
C                            (  1.0+1.0i  1.0+1.0i  0.0+0.0i  )
C
      DATA A/(0.0,0.0), 7*(1.0,1.0), (0.0,0.0)/
C                                  Copy A to B
      CALL CCBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)
C                                  Print results
      CALL WRCRN (’B’, NUCB+NLCB+1, N, B, LDB, 0)
      END
```

Output

```
                          B
                1                 2                 3
1  ( 0.000, 0.000)  ( 1.000, 1.000)  ( 1.000, 1.000)
2  ( 1.000, 1.000)  ( 1.000, 1.000)  ( 1.000, 1.000)
3  ( 1.000, 1.000)  ( 1.000, 1.000)  ( 0.000, 0.000)
```

CRGRB/DCRGRB (Single/Double precision)

Convert a real general matrix to a matrix in band storage mode.

Usage

`CALL CRGRB (N, A, LDA, NLC, NUC, B, LDB)`

Arguments

N — Order of the matrices `A` and `B`.   (Input)

A — Real `N` by `N` matrix.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NLC — Number of lower codiagonals in `B`.   (Input)

NUC — Number of upper codiagonals in `B`.   (Input)

B — Real (`NUC` + 1 + `NLC`) by `N` array containing the band matrix in band storage mode.   (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine `CRGRB` converts the real general $N \times N$ matrix A with m_u = `NUC` upper codiagonals and m_l = `NLC` lower codiagonals into the real band matrix B of order N. The first m_u rows of B then contain the upper codiagonals of A, the next row contains the main diagonal of A, and the last m_l rows of B contain the lower codiagonals of A.

Example

A real 4 × 4 matrix with one upper codiagonal and three lower codiagonals is copied to a real band matrix of order 4 in band storage mode.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLC, NUC
      PARAMETER  (LDA=4, LDB=5, N=4, NLC=3, NUC=1)
C
      REAL       A(LDA,N), B(LDB,N)
      EXTERNAL   CRGRB, WRRRN
C                                  Set values for  A
C                                  A = (  1.0     2.0    0.0    0.0)
C                                      ( -2.0     1.0    3.0    0.0)
C                                      (  0.0    -3.0    1.0    4.0)
C                                      ( -7.0     0.0   -4.0    1.0)
C
      DATA A/1.0, -2.0, 0.0, -7.0, 2.0, 1.0, -3.0, 0.0, 0.0, 3.0, 1.0,
     &     -4.0, 0.0, 0.0, 4.0, 1.0/
C                                  Convert A to band matrix B
```

```
      CALL CRGRB (N, A, LDA, NLC, NUC, B, LDB)
C                                  Print results
      CALL WRRRN ('B', NUC+NLC+1, N, B, LDB, 0)
      END
```

Output

```
                B
         1       2       3       4
1    0.000   2.000   3.000   4.000
2    1.000   1.000   1.000   1.000
3   -2.000  -3.000  -4.000   0.000
4    0.000   0.000   0.000   0.000
5   -7.000   0.000   0.000   0.000
```

CRBRG/DCRBRG (Single/Double precision)

Convert a real matrix in band storage mode to a real general matrix.

Usage

CALL CRBRG (N, A, LDA, NLC, NUC, B, LDB)

Arguments

N — Order of the matrices A and B.   (Input)

A — Real (NUC + 1 + NLC) by N array containing the band matrix in band storage mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLC — Number of lower codiagonals in A.   (Input)

NUC — Number of upper codiagonals in A.   (Input)

B — Real N by N array containing the matrix.   (Output)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine CRBRG converts the real band matrix A of order N in band storage mode into the real $N \times N$ general matrix B with m_u = NUC upper codiagonals and m_l = NLC lower codiagonals. The first m_u rows of A are copied to the upper codiagonals of B, the next row of A is copied to the diagonal of B, and the last m_l rows of A are copied to the lower codiagonals of B.

Example

A real band matrix of order 3 in band storage mode with one upper codiagonal and one lower codiagonal is copied to a 3 × 3 real general matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLC, NUC
      PARAMETER  (LDA=3, LDB=3, N=3, NLC=1, NUC=1)
C
      REAL       A(LDA,N), B(LDB,N)
      EXTERNAL   CRBRG, WRRRN
C                                  Set values for  A (in band mode)
C                                  A = (  0.0     1.0    1.0)
C                                      (  4.0     3.0    2.0)
C                                      (  2.0     2.0    0.0)
C
      DATA A/0.0, 4.0, 2.0, 1.0, 3.0, 2.0, 1.0, 2.0, 0.0/
C                                  Convert band matrix A to matrix B
      CALL CRBRG (N, A, LDA, NLC, NUC, B, LDB)
C                                  Print results
      CALL WRRRN (’B’, N, N, B, LDB, 0)
      END
```

Output

```
             B
        1       2       3
1   4.000   1.000   0.000
2   2.000   3.000   1.000
3   0.000   2.000   2.000
```

CCGCB/DCCGCB (Single/Double precision)

Convert a complex general matrix to a matrix in complex band storage mode.

Usage

```
CALL CCGCB (N, A, LDA, NLC, NUC, B, LDB)
```

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Complex `N` by `N` array containing the matrix. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NLC — Number of lower codiagonals in `B`. (Input)

NUC — Number of upper codiagonals in `B`. (Input)

B — Complex (`NUC` + 1 + `NLC`) by `N` array containing the band matrix in band storage mode. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

Algorithm

The routine `CCGCB` converts the complex general matrix A of order N with m_u = `NUC` upper codiagonals and m_l = `NLC` lower codiagonals into the complex band matrix B of order N in band storage mode. The first m_u rows of B then contain the upper codiagonals of A, the next row contains the main diagonal of A, and the last m_l rows of B contain the lower codiagonals of A.

Example

A complex general matrix of order 4 with one upper codiagonal and three lower codiagonals is copied to a complex band matrix of order 4 in band storage mode.

```
C                                 Declare variables
      INTEGER    LDA, LDB, N, NLC, NUC
      PARAMETER  (LDA=4, LDB=5, N=4, NLC=3, NUC=1)
C
      COMPLEX    A(LDA,N), B(LDB,N)
      EXTERNAL   CCGCB, WRCRN
C                     Set values for  A
C                     A = (  1.0+0.0i   2.0+1.0i  0.0+0.0i  0.0+0.0i )
C                         ( -2.0+1.0i   1.0+0.0i  3.0+2.0i  0.0+0.0i )
C                         (  0.0+0.0i  -3.0+2.0i  1.0+0.0i  4.0+3.0i )
C                         ( -7.0+1.0i   0.0+0.0i -4.0+3.0i  1.0+0.0i )
C
      DATA A/(1.0,0.0), (-2.0,1.0), (0.0,0.0), (-7.0,1.0), (2.0,1.0),
     &     (1.0,0.0), (-3.0,2.0), (0.0,0.0), (0.0,0.0), (3.0,2.0),
     &     (1.0,0.0), (-4.0,3.0), (0.0,0.0), (0.0,0.0), (4.0,3.0),
     &     (1.0,0.0)/
C                                 Convert A to band matrix B
      CALL CCGCB (N, A, LDA, NLC, NUC, B, LDB)
C                                 Print results
      CALL WRCRN (’B’, NUC+NLC+1, N, B, LDB, 0)
      END
```

Output

```
                                 B
                 1                2                3                4
1  ( 0.000, 0.000)  ( 2.000, 1.000)  ( 3.000, 2.000)  ( 4.000, 3.000)
2  ( 1.000, 0.000)  ( 1.000, 0.000)  ( 1.000, 0.000)  ( 1.000, 0.000)
3  (-2.000, 1.000)  (-3.000, 2.000)  (-4.000, 3.000)  ( 0.000, 0.000)
4  ( 0.000, 0.000)  ( 0.000, 0.000)  ( 0.000, 0.000)  ( 0.000, 0.000)
5  (-7.000, 1.000)  ( 0.000, 0.000)  ( 0.000, 0.000)  ( 0.000, 0.000)
```

CCBCG/DCCBCG (Single/Double precision)

Convert a complex matrix in band storage mode to a complex matrix in full storage mode.

Usage

`CALL CCBCG (N, A, LDA, NLC, NUC, B, LDB)`

Arguments

N — Order of the matrices A and B.   (Input)

A — Complex (NUC + 1 + NLC) by N matrix containing the band matrix in band mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLC — Number of lower codiagonals in A.   (Input)

NUC — Number of upper codiagonals in A.   (Input)

B — Complex N by N matrix containing the band matrix in full mode.   (Output)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine CCBCG converts the complex band matrix A of order N with m_u = NUC upper codiagonals and m_l = NLC lower codiagonals into the $N \times N$ complex general matrix B. The first m_u rows of A are copied to the upper codiagonals of B, the next row of A is copied to the diagonal of B, and the last m_l rows of A are copied to the lower codiagonals of B.

Example

A complex band matrix of order 4 in band storage mode with one upper codiagonal and three lower codiagonals is copied into a 4 × 4 complex general matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLC, NUC
      PARAMETER  (LDA=5, LDB=4, N=4, NLC=3, NUC=1)
C
      COMPLEX    A(LDA,N), B(LDB,N)
      EXTERNAL   CCBCG, WRCRN
C                      Set values for  A (in band mode)
C                      A = (  0.0+0.0i  2.0+1.0i  3.0+2.0i  4.0+3.0i  )
C                          (  1.0+0.0i  1.0+0.0i  1.0+0.0i  1.0+0.0i  )
C                          ( -2.0+1.0i -3.0+2.0i -4.0+3.0i  0.0+0.0i  )
C                          (  0.0+0.0i  0.0+0.0i  0.0+0.0i  0.0+0.0i  )
C                          ( -7.0+1.0i  0.0+0.0i  0.0+0.0i  0.0+0.0i  )
C
      DATA A/(0.0,0.0), (1.0,0.0), (-2.0,1.0), (0.0,0.0), (-7.0,1.0),
     &     (2.0,1.0), (1.0,0.0), (-3.0,2.0), 2*(0.0,0.0), (3.0,2.0),
     &     (1.0,0.0), (-4.0,3.0), 2*(0.0,0.0), (4.0,3.0), (1.0,0.0),
     &     3*(0.0,0.0)/
C                                  Convert band matrix A to matrix B
      CALL CCBCG (N, A, LDA, NLC, NUC, B, LDB)
C                                  Print results
      CALL WRCRN ('B', N, N, B, LDB, 0)
      END
```

Output

```
                                   B
                 1                 2                 3                 4
1  ( 1.000, 0.000)  ( 2.000, 1.000)  ( 0.000, 0.000)  ( 0.000, 0.000)
2  (-2.000, 1.000)  ( 1.000, 0.000)  ( 3.000, 2.000)  ( 0.000, 0.000)
3  ( 0.000, 0.000)  (-3.000, 2.000)  ( 1.000, 0.000)  ( 4.000, 3.000)
4  (-7.000, 1.000)  ( 0.000, 0.000)  (-4.000, 3.000)  ( 1.000, 0.000)
```

CRGCG/DCRGCG (Single/Double precision)

Copy a real general matrix to a complex general matrix.

Usage

`CALL CRGCG (N, A, LDA, B, LDB)`

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Real matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

B — Complex matrix of order `N` containing a copy of `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

Comments

The matrices `A` and `B` may be the same.

Algorithm

The routine `CRGCG` copies a real $N \times N$ matrix to a complex $N \times N$ matrix.

Example

A 3 × 3 real matrix is copied to a 3 × 3 complex matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDB, N
      PARAMETER  (LDA=3, LDB=3, N=3)
C
      REAL       A(LDA,N)
      COMPLEX    B(LDB,N)
      EXTERNAL   CRGCG, WRCRN
C                                 Set values for  A
C                                 A = (  2.0     1.0    3.0 )
C                                     (  4.0     1.0    0.0 )
C                                     ( -1.0     2.0    0.0 )
C
      DATA A/2.0, 4.0, -1.0, 1.0, 1.0, 2.0, 3.0, 0.0, 0.0/
C                                 Convert real A to complex B
```

```
      CALL CRGCG (N, A, LDA, B, LDB)
C                                  Print results
      CALL WRCRN ('B', N, N, B, LDB, 0)
      END
```

Output

```
                         B
                1                 2                 3
1  ( 2.000, 0.000)  ( 1.000, 0.000)  ( 3.000, 0.000)
2  ( 4.000, 0.000)  ( 1.000, 0.000)  ( 0.000, 0.000)
3  (-1.000, 0.000)  ( 2.000, 0.000)  ( 0.000, 0.000)
```

CRRCR/DCRRCR (Single/Double precision)

Copy a real rectangular matrix to a complex rectangular matrix.

Usage

`CALL CRRCR (NRA, NCA, A, LDA, NRB, NCB, B, LDB)`

Arguments

NRA — Number of rows in `A`. (Input)

NCA — Number of columns in `A`. (Input)

A — Real `NRA` by `NCA` rectangular matrix. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NRB — Number of rows in `B`. (Input)
It must be the same as `NRA`.

NCB — Number of columns in `B`. (Input)
It must be the same as `NCA`.

B — Complex `NRB` by `NCB` rectangular matrix containing a copy of `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

Comments

The matrices `A` and `B` may be the same.

Algorithm

The routine `CRRCR` copies a real rectangular matrix to a complex rectangular matrix.

Example

A 3 × 2 real matrix is copied to a 3 × 2 complex matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDB, NCA, NCB, NRA, NRB
      PARAMETER  (LDA=3, LDB=3, NCA=2, NCB=2, NRA=3, NRB=3)
C
      REAL       A(LDA,NCA)
      COMPLEX    B(LDB,NCB)
      EXTERNAL   CRRCR, WRCRN
C                                 Set values for  A
C                                 A = (  1.0     4.0  )
C                                     (  2.0     5.0  )
C                                     (  3.0     6.0  )
C
      DATA A/1.0, 2.0, 3.0, 4.0, 5.0, 6.0/
C                                 Convert real A to complex B
      CALL CRRCR (NRA, NCA, A, LDA, NRB, NCB, B, LDB)
C                                 Print results
      CALL WRCRN ('B', NRB, NCB, B, LDB, 0)
      END
```

Output

```
                B
                 1                 2
1  ( 1.000, 0.000)  ( 4.000, 0.000)
2  ( 2.000, 0.000)  ( 5.000, 0.000)
3  ( 3.000, 0.000)  ( 6.000, 0.000)
```

CRBCB/DCRBCB (Single/Double precision)

Convert a real matrix in band storage mode to a complex matrix in band storage mode.

Usage

CALL CRBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)

Arguments

N — Order of the matrices A and B.   (Input)

A — Real band matrix of order N.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals in A.   (Input)

NUCA — Number of upper codiagonals in A.   (Input)

***B* —** Complex matrix of order N containing a copy of A.   (Output)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program.   (Input)

NLCB — Number of lower codiagonals in B.   (Input)
NLCB must be at least as large as NLCA.

NUCB — Number of upper codiagonals in `B`. (Input)
`NUCB` must be at least as large as `NUCA`.

Algorithm

The routine `CRBCB` converts a real band matrix in band storage mode with `NUCA` upper codiagonals and `NLCA` lower codiagonals into a complex band matrix in band storage mode with `NUCB` upper codiagonals and `NLCB` lower codiagonals.

Example

A real band matrix of order 3 in band storage mode with one upper codiagonal and one lower codiagonal is copied into another complex band matrix in band storage mode.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLCA, NLCB, NUCA, NUCB
      PARAMETER  (LDA=3, LDB=3, N=3, NLCA=1, NLCB=1, NUCA=1, NUCB=1)
C
      REAL       A(LDA,N)
      COMPLEX    B(LDB,N)
      EXTERNAL   CRBCB, WRCRN
C                                  Set values for  A (in band mode)
C                                  A = (  0.0     1.0    1.0)
C                                      (  1.0     1.0    1.0)
C                                      (  1.0     1.0    0.0)
C
      DATA A/0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0/
C                                  Convert real band matrix A
C                                  to complex band matrix B
      CALL CRBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB)
C                                  Print results
      CALL WRCRN (’B’, NUCB+NLCB+1, N, B, LDB, 0)
      END
```

Output

```
                            B
                1                 2                 3
1  ( 0.000, 0.000)  ( 1.000, 0.000)  ( 1.000, 0.000)
2  ( 1.000, 0.000)  ( 1.000, 0.000)  ( 1.000, 0.000)
3  ( 1.000, 0.000)  ( 1.000, 0.000)  ( 0.000, 0.000)
```

CSFRG/DCSFRG (Single/Double precision)

Extend a real symmetric matrix defined in its upper triangle to its lower triangle.

Usage

`CALL CSFRG (N, A, LDA)`

Arguments

N – Order of the matrix `A`. (Input)

A — `N` by `N` symmetric matrix of order `N` to be filled out. (Input/Output)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine `CSFRG` converts an $N \times N$ matrix A in symmetric mode into a general matrix by filling in the lower triangular portion of A using the values defined in its upper triangular portion.

Example

The lower triangular portion of a real 3 × 3 symmetric matrix is filled with the values defined in its upper triangular portion.

```
C                                  Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
C
      REAL       A(LDA,N)
      EXTERNAL   CSFRG, WRRRN
C                                  Set values for  A
C                                  A = (   0.0   3.0   4.0  )
C                                      (         1.0   5.0  )
C                                      (               2.0  )
C
      DATA A/3*0.0, 3.0, 1.0, 0.0, 4.0, 5.0, 2.0/
C                                  Fill the lower portion of A
      CALL CSFRG (N, A, LDA)
C                                  Print results
      CALL WRRRN (’A’, N, N, A, LDA, 0)
      END
```

Output

```
            A
        1       2       3
1   0.000   3.000   4.000
2   3.000   1.000   5.000
3   4.000   5.000   2.000
```

CHFCG/DCHFCG (Single/Double precision)

Extend a complex Hermitian matrix defined in its upper triangle to its lower triangle.

Usage

`CALL CHFCG (N, A, LDA)`

Arguments

N — Order of the matrix.   (Input)

A — Complex Hermitian matrix of order `N`. (Input/Output)
On input, the upper triangle of `A` defines a Hermitian matrix. On output, the lower triangle of `A` is defined so that `A` is Hermitian.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

Comments

Informational errors

Type	Code	
3	1	The matrix is not Hermitian. It has a diagonal entry with a small imaginary part.
4	2	The matrix is not Hermitian. It has a diagonal entry with an imaginary part.

Algorithm

The routine `CHFCG` converts an $N \times N$ complex matrix A in Hermitian mode into a complex general matrix by filling in the lower triangular portion of A using the values defined in its upper triangular portion.

Example

A complex 3×3 Hermitian matrix defined in its upper triangle is extended to its lower triangle.

```
C                                 Declare variables
      INTEGER    LDA, N
      PARAMETER  (LDA=3, N=3)
C
      COMPLEX    A(LDA,N)
      EXTERNAL   CHFCG, WRCRN
C                                 Set values for  A
C                           A = (  1.0+0.0i  1.0+1.0i  1.0+2.0i  )
C                               (            2.0+0.0i  2.0+2.0i  )
C                               (                      3.0+0.0i  )
C
      DATA A/(1.0,0.0), 2*(0.0,0.0), (1.0,1.0), (2.0,0.0), (0.0,0.0),
     &     (1.0,2.0), (2.0,2.0), (3.0,0.0)/
C                                 Fill in lower Hermitian matrix
      CALL CHFCG (N, A, LDA)
C                                 Print results
      CALL WRCRN (’A’, N, N, A, LDA, 0)
      END
```

Output

```
                            A
                 1                2                3
1  ( 1.000, 0.000)  ( 1.000, 1.000)  ( 1.000, 2.000)
2  ( 1.000,-1.000)  ( 2.000, 0.000)  ( 2.000, 2.000)
3  ( 1.000,-2.000)  ( 2.000,-2.000)  ( 3.000, 0.000)
```

CSBRB/DCSBRB (Single/Double precision)

Copy a real symmetric band matrix stored in band symmetric storage mode to a real band matrix stored in band storage mode.

Usage

```
CALL CSBRB (N, A, LDA, NUCA, B, LDB, NLCB, NUCB)
```

Arguments

N — Order of the matrices `A` and `B`.   (Input)

A — Real band symmetric matrix of order `N`.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NUCA — Number of codiagonals in `A`.   (Input)

B — Real band matrix of order `N` containing a copy of `A`.   (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

NLCB — Number of lower codiagonals in `B`.   (Input)
`NLCB` must be at least as large as `NUCA`.

NUCB — Number of upper codiagonals in `B`.   (Input)
`NUCB` must be at least as large as `NUCA`.

Algorithm

The routine `CSBRB` copies a real matrix *A* stored in symmetric band mode to a matrix B stored in band mode. The lower codiagonals of *B* are set using the values from the upper codiagonals of *A*.

Example

A real matrix of order 4 in band symmetric storage mode with 2 upper codiagonals is copied to a real matrix in band storage mode with 2 upper codiagonals and 2 lower codiagonals.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NLCB, NUCA, NUCB
      PARAMETER  (N=4, NUCA=2, LDA=NUCA+1, NLCB=NUCA, NUCB=NUCA,
     &           LDB=NLCB+NUCB+1)
C
      REAL       A(LDA,N), B(LDB,N)
      EXTERNAL   CSBRB, WRRRN
C                           Set values for  A, in band mode
C                           A = (  0.0  0.0  2.0  1.0 )
C                               (  0.0  2.0  3.0  1.0 )
C                               (  1.0  2.0  3.0  4.0 )
C
      DATA A/2*0.0, 1.0, 0.0, 2.0, 2.0, 2.0, 3.0, 3.0, 1.0, 1.0, 4.0/
```

```
C                                 Copy A to B
      CALL CSBRB (N, A, LDA, NUCA, B, LDB, NLCB, NUCB)
C                                 Print results
      CALL WRRRN ('B', NLCB+NUCB+1, N, B, LDB, 0)
      END
```

Output

```
               B
        1       2       3       4
1   0.000   0.000   2.000   1.000
2   0.000   2.000   3.000   1.000
3   1.000   2.000   3.000   4.000
4   2.000   3.000   1.000   0.000
5   2.000   1.000   0.000   0.000
```

CHBCB/DCHBCB (Single/Double precision)

Copy a complex Hermitian band matrix stored in band Hermitian storage mode to a complex band matrix stored in band storage mode.

Usage

`CALL CHBCB (N, A, LDA, NUCA, B, LDB, NLCB, NUCB)`

Arguments

N — Order of the matrices `A` and `B`. (Input)

A — Complex band Hermitian matrix of order `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NUCA — Number of codiagonals in `A`. (Input)

B — Complex band matrix of order `N` containing a copy of `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

NLCB — Number of lower codiagonals in `B`. (Input)
`NLCB` must be at least as large as `NUCA`.

NUCB — Number of upper codiagonals in `B`. (Input)
`NUCB` must be at least as large as `NUCA`.

Comments

Informational errors

Type	Code	
3	1	An element on the diagonal has a complex part that is near zero, the complex part is set to zero.
4	1	An element on the diagonal has a complex part that is not zero.

Algorithm

The routine CSBRB copies a complex matrix *A* stored in Hermitian band mode to a matrix *B* stored in complex band mode. The lower codiagonals of *B* are filled using the values in the upper codiagonals of *A*.

Example

A complex Hermitian matrix of order 3 in band Hermitian storage mode with one upper codiagonal is copied to a complex matrix in band storage mode.

```
C                                 Declare variables
      INTEGER    LDA, LDB, N, NLCB, NUCA, NUCB
      PARAMETER  (N=3, NUCA=1, LDA=NUCA+1, NLCB=NUCA, NUCB=NUCA,
     &           LDB=NLCB+NUCB+1)
C
      COMPLEX    A(LDA,N), B(LDB,N)
      EXTERNAL   CHBCB, WRCRN
C                                 Set values for  A (in band mode)
C                           A = (  0.0+0.0i -1.0+1.0i -2.0+2.0i  )
C                               (  1.0+0.0i  1.0+0.0i  1.0+0.0i  )
C
      DATA A/(0.0,0.0), (1.0,0.0), (-1.0,1.0), (1.0,0.0), (-2.0,2.0),
     &     (1.0,0.0)/
C                                 Copy a complex Hermitian band matrix
C                                 to a complex band matrix
      CALL CHBCB (N, A, LDA, NUCA, B, LDB, NLCB, NUCB)
C                                 Print results
      CALL WRCRN ('B', NLCB+NUCB+1, N, B, LDB, 0)
      END
```

Output

```
                            B
                 1                2                3
1  ( 0.000, 0.000)  (-1.000, 1.000)  (-2.000, 2.000)
2  ( 1.000, 0.000)  ( 1.000, 0.000)  ( 1.000, 0.000)
3  (-1.000,-1.000)  (-2.000,-2.000)  ( 0.000, 0.000)
```

TRNRR/DTRNRR (Single/Double precision)

Transpose a rectangular matrix.

Usage

CALL TRNRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB)

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA matrix in full storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NRB — Number of rows of `B`. (Input)
`NRB` must be equal to `NCA`.

NCB — Number of columns of `B`. (Input)
`NCB` must be equal to `NRA`.

B — Real `NRB` by `NCB` matrix in full storage mode containing the transpose of `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

Comments

If `LDA` = `LDB` and `NRA` = `NCA`, then `A` and `B` can occupy the same storage locations; otherwise, `A` and `B` must be stored separately.

Algorithm

The routine `TRNRR` computes the transpose $B = A^T$ of a real rectangular matrix A.

Example

Transpose the 5 × 3 real rectangular matrix A into the 3 × 5 real rectangular matrix B.

```
C                                  Declare variables
      INTEGER    LDA, LDB, NCA, NCB, NRA, NRB
      PARAMETER  (LDA=5, LDB=3, NCA=3, NCB=5, NRA=5, NRB=3)
C
      REAL       A(LDA,NCA), B(LDB,NCB)
      EXTERNAL   TRNRR, WRRRN
C                                  Set values for A
C                                  A = ( 11.0  12.0  13.0 )
C                                      ( 21.0  22.0  23.0 )
C                                      ( 31.0  32.0  33.0 )
C                                      ( 41.0  42.0  43.0 )
C                                      ( 51.0  52.0  53.0 )
C
      DATA A/11.0, 21.0, 31.0, 41.0, 51.0, 12.0, 22.0, 32.0, 42.0,
     &     52.0, 13.0, 23.0, 33.0, 43.0, 53.0/
C                                  B = transpose(A)
      CALL TRNRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB)
C                                  Print results
      CALL WRRRN ('B = trans(A)', NRB, NCB, B, LDB, 0)
      RETURN
      END
```

Output

```
              B = trans(A)
        1       2       3       4       5
1   11.00   21.00   31.00   41.00   51.00
```

```
2   12.00   22.00   32.00   42.00   52.00
3   13.00   23.00   33.00   43.00   53.00
```

MXTXF/DMXTXF (Single/Double precision)

Compute the transpose product of a matrix, A^TA.

Usage

`CALL MXTXF (NRA, NCA, A, LDA, NB, B, LDB)`

Arguments

NRA — Number of rows in `A`.   (Input)

NCA — Number of columns in `A`.   (Input)

A — Real `NRA` by `NCA` rectangular matrix.   (Input)
The transpose product of `A` is to be computed.

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NB — Order of the matrix `B`.   (Input)
`NB` must be equal to `NCA`.

B — Real `NB` by `NB` symmetric matrix containing the transpose product A^TA. (Output)

LDB — Leading dimension of *B* exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine `MXTXF` computes the real general matrix $B = A^TA$ given the real rectangular matrix *A*.

Example

Multiply the transpose of a 3 × 4 real matrix by itself. The output matrix will be a 4 × 4 real symmetric matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDB, NB, NCA, NRA
      PARAMETER  (LDA=3, LDB=4, NB=4, NCA=4, NRA=3)
C
      REAL       A(LDA,NCA), B(LDB,NB)
      EXTERNAL   MXTXF, WRRRN
C                                 Set values for A
C                                 A = ( 3.0  1.0  4.0  2.0 )
C                                     ( 0.0  2.0  1.0 -1.0 )
C                                     ( 6.0  1.0  3.0  2.0 )
C
      DATA A/3.0, 0.0, 6.0, 1.0, 2.0, 1.0, 4.0, 1.0, 3.0, 2.0, -1.0,
     &     2.0/
C                                 Compute B = trans(A)*A
      CALL MXTXF (NRA, NCA, A, LDA, NB, B, LDB)
C                                 Print results
      CALL WRRRN (’B = trans(A)*A’, NB, NB, B, LDB, 0)
      END
```

Output

```
         B = trans(A)*A
        1        2        3        4
1   45.00     9.00    30.00    18.00
2    9.00     6.00     9.00     2.00
3   30.00     9.00    26.00    13.00
4   18.00     2.00    13.00     9.00
```

MXTYF/DMXTYF (Single/Double precision)

Multiply the transpose of matrix *A* by matrix *B*, $A^T B$.

Usage

```
CALL MXTYF (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC,
            C, LDC)
```

Arguments

NRA — Number of rows in `A`.   (Input)

NCA — Number of columns in `A`.   (Input)

A — Real `NRA` by `NCA` matrix.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NRB — Number of rows in `B`.   (Input)
`NRB` must be the same as `NRA`.

NCB — Number of columns in `B`.   (Input)

B — Real `NRB` by `NCB` matrix.   (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

NRC — Number of rows of `C`.   (Input)
`NRC` must be equal to `NCA`.

NCC — Number of columns of `C`.   (Input)
`NCC` must be equal to `NCB`.

C — Real `NCA` by `NCB` matrix containing the transpose product $A^T B$.   (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

The routine MXTYF computes the real general matrix $C = A^T B$ given the real rectangular matrices A and B.

Example

Multiply the transpose of a 3 × 4 real matrix by a 3 × 3 real matrix. The output matrix will be a 4 × 3 real matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, LDC, NCA, NCB, NCC, NRA, NRB, NRC
      PARAMETER  (LDA=3, LDB=3, LDC=4, NCA=4, NCB=3, NCC=3, NRA=3,
     &           NRB=3, NRC=4)
C
      REAL       A(LDA,NCA), B(LDB,NCB), C(LDC,NCC)
      EXTERNAL   MXTYF, WRRRN
C                                  Set values for A
C                                  A = ( 1.0  0.0  2.0  0.0 )
C                                      ( 3.0  4.0 -1.0  0.0 )
C                                      ( 2.0  1.0  2.0  1.0 )
C
C                                  Set values for B
C                                  B = ( -1.0  2.0  0.0 )
C                                      (  3.0  0.0 -1.0 )
C                                      (  0.0  5.0  2.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
      DATA B/-1.0, 3.0, 0.0, 2.0, 0.0, 5.0, 0.0, -1.0, 2.0/
C                                  Compute C = trans(A)*B
      CALL MXTYF (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC, C, LDC)
C                                  Print results
      CALL WRRRN ('C = trans(A)*B', NRC, NCC, C, LDC, 0)
      END
```

Output

```
      C = trans(A)*B
        1       2       3
1    8.00   12.00    1.00
2   12.00    5.00   -2.00
3   -5.00   14.00    5.00
4    0.00    5.00    2.00
```

MXYTF/DMXYTF (Single/Double precision)

Multiply a matrix A by the transpose of a matrix B, AB^T.

Usage

```
CALL MXYTF (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC,
            C, LDC)
```

Arguments

NRA — Number of rows in `A`. (Input)

NCA — Number of columns in `A`. (Input)

A — Real `NRA` by `NCA` rectangular matrix. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NRB — Number of rows in `B`. (Input)

NCB — Number of columns in `B`. (Input)
`NCB` must be the same as `NCA`.

B — Real `NRB` by `NCB` rectangular matrix. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

NRC — Number of rows of `C`. (Input)
`NRC` must be equal to `NRA`.

NCC — Number of columns of `C`. (Input)
`NCC` must be equal to `NRB`.

C — Real `NRC` by `NCC` rectangular matrix containing the transpose product AB^T. (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)

Algorithm

The routine `MXYTF` computes the real general matrix $C = AB^T$ given the real rectangular matrices A and B.

Example

Multiply a 3 × 4 real matrix by the transpose of a 3 × 4 real matrix. The output matrix will be a 3 × 3 real matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, LDC, NCA, NCB, NCC, NRA, NRB, NRC
      PARAMETER  (LDA=3, LDB=3, LDC=3, NCA=4, NCB=4, NCC=3, NRA=3,
     &           NRB=3, NRC=3)
C
      REAL       A(LDA,NCA), B(LDB,NCB), C(LDC,NCC)
      EXTERNAL   MXYTF, WRRRN
C                                  Set values for A
C                                  A = ( 1.0  0.0  2.0  0.0 )
C                                      ( 3.0  4.0 -1.0  0.0 )
C                                      ( 2.0  1.0  2.0  1.0 )
C
C                                  Set values for B
C                                  B = ( -1.0  2.0  0.0  2.0 )
```

```
C                                    (  3.0  0.0 -1.0 -1.0 )
C                                    (  0.0  5.0  2.0  5.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
      DATA B/-1.0, 3.0, 0.0, 2.0, 0.0, 5.0, 0.0, -1.0, 2.0, 2.0, -1.0,
     &     5.0/
C                                Compute C = A*trans(B)
      CALL MXYTF (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC, C, LDC)
C                                Print results
      CALL WRRRN ('C = A*trans(B)', NRC, NCC, C, LDC, 0)
      END
```

Output

```
      C = A*trans(B)
        1       2       3
1   -1.00    1.00    4.00
2    5.00   10.00   18.00
3    2.00    3.00   14.00
```

MRRRR/DMRRRR (Single/Double precision)

Multiply two real rectangular matrices, *AB*.

Usage

```
CALL MRRRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC,
            C, LDC)
```

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA matrix in full storage mode. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NRB — Number of rows of B. (Input)
NRB must be equal to NCA.

NCB — Number of columns of B. (Input)

B — Real NRB by NCB matrix in full storage mode. (Input)

LDB — Leading dimension of B exactly as specified in the dimension statement of the calling program. (Input)

NRC — Number of rows of C. (Input)
NRC must be equal to NRA.

NCC — Number of columns of C. (Input)
NCC must be equal to NCB.

C — Real `NRC` by `NCC` matrix containing the product *AB* in full storage mode. (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)

Algorithm

Given the real rectangular matrices *A* and *B*, `MRRRR` computes the real rectangular matrix *C* = *AB*.

Example

Multiply a 3 × 4 real matrix by a 4 × 3 real matrix. The output matrix will be a 3 × 3 real matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, LDC, NCA, NCB, NCC, NRA, NRB, NRC
      PARAMETER  (LDA=3, LDB=4, LDC=3, NCA=4, NCB=3, NCC=3, NRA=3,
     &           NRB=4, NRC=3)
C
      REAL       A(LDA,NCA), B(LDB,NCB), C(LDC,NCC)
      EXTERNAL   MRRRR, WRRRN
C                                  Set values for A
C                                  A = ( 1.0  0.0  2.0  0.0 )
C                                      ( 3.0  4.0 -1.0  0.0 )
C                                      ( 2.0  1.0  2.0  1.0 )
C
C                                  Set values for B
C                                  B = ( -1.0  0.0  2.0 )
C                                      (  3.0  5.0  2.0 )
C                                      (  0.0  0.0 -1.0 )
C                                      (  2.0 -1.0  5.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
      DATA B/-1.0, 3.0, 0.0, 2.0, 0.0, 5.0, 0.0, -1.0, 2.0, 2.0, -1.0,
     &     5.0/
C                                  Compute C = A*B
      CALL MRRRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC, C, LDC)
C                                  Print results
      CALL WRRRN (’C = A*B’, NRC, NCC, C, LDC, 0)
      END
```

Output

```
          C = A*B
        1       2       3
1   -1.00    0.00    0.00
2    9.00   20.00   15.00
3    3.00    4.00    9.00
```

MCRCR/DMCRCR (Single/Double precision)

Multiply two complex rectangular matrices, *AB*.

Usage

```
CALL MCRCR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC,
            C, LDC)
```

Arguments

NRA — Number of rows of `A`.   (Input)

NCA — Number of columns of `A`.   (Input)

A — Complex `NRA` by `NCA` rectangular matrix.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NRB — Number of rows of `B`.   (Input)
`NRB` must be equal to `NCA`.

NCB — Number of columns of `B`.   (Input)

B — Complex `NRB` by `NCB` rectangular matrix.   (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

NRC — Number of rows of `C`.   (Input)
`NRC` must be equal to `NRA`.

NCC — Number of columns of `C`.   (Input)
`NCC` must be equal to `NCB`.

C — Complex `NRC` by `NCC` rectangular matrix containing the product `A * B`. (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program.   (Input)

Algorithm

Given the complex rectangular matrices *A* and *B*, `MCRCR` computes the complex rectangular matrix *C* = *AB*.

Example

Multiply a 3 × 4 complex matrix by a 4 × 3 complex matrix. The output matrix will be a 3 × 3 complex matrix.

```
C                                 Declare variables
      INTEGER    LDA, LDB, LDC, NCA, NCB, NCC, NRA, NRB, NRC
      PARAMETER  (LDA=3, LDB=4, LDC=3, NCA=4, NCB=3, NCC=3, NRA=3,
     &           NRB=4, NRC=3)
```

```
C
      COMPLEX    A(LDA,NCA), B(LDB,NCB), C(LDC,NCC)
      EXTERNAL   MCRCR, WRCRN
C                                 Set values for A
C             A = ( 1.0 + 1.0i  -1.0+ 2.0i  0.0 + 1.0i  0.0 - 2.0i )
C                 ( 3.0 + 7.0i  6.0 - 4.0i  2.0 - 1.0i  0.0 + 1.0i )
C                 ( 1.0 + 0.0i  1.0 - 2.0i  -2.0+ 0.0i  0.0 + 0.0i )
C
C                                 Set values for B
C             B = ( 2.0 + 1.0i  3.0 + 2.0i  3.0 + 1.0i )
C                 ( 2.0 - 1.0i  4.0 - 2.0i  5.0 - 3.0i )
C                 ( 1.0 + 0.0i  0.0 - 1.0i  0.0 + 1.0i )
C                 ( 2.0 + 1.0i  1.0 + 2.0i  0.0 - 1.0i )
C
      DATA A/(1.0,1.0), (3.0,7.0), (1.0,0.0), (-1.0,2.0), (6.0,-4.0),
     &     (1.0,-2.0), (0.0,1.0), (2.0,-1.0), (-2.0,0.0), (0.0,-2.0),
     &     (0.0,1.0), (0.0,0.0)/
      DATA B/(2.0,1.0), (2.0,-1.0), (1.0,0.0), (2.0,1.0), (3.0,2.0),
     &     (4.0,-2.0), (0.0,-1.0), (1.0,2.0), (3.0,1.0), (5.0,-3.0),
     &     (0.0,1.0), (0.0,-1.0)/
C                                 Compute C = A*B
      CALL MCRCR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC, C, LDC)
C                                 Print results
      CALL WRCRN ('C = A*B', NRC, NCC, C, LDC, 0)
      END
```

Output

```
                     C = A*B
                  1                 2                 3
1  (  3.00,  5.00)  (  6.00, 13.00)  (  0.00, 17.00)
2  (  8.00,  4.00)  (  8.00, -2.00)  ( 22.00,-12.00)
3  (  0.00, -4.00)  (  3.00, -6.00)  (  2.00,-14.00)
```

HRRRR/DHRRRR (Single/Double precision)

Compute the Hadamard product of two real rectangular matrices.

Usage

```
CALL HRRRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC,
            C, LDC)
```

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA rectangular matrix. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NRB — Number of rows of B. (Input)
NRB must be equal to NRA.

NCB — Number of columns of `B`. (Input)
`NCB` must be equal to `NCA`.

B — Real `NRB` by `NCB` rectangular matrix. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

NRC — Number of rows of `C`. (Input)
`NRC` must be equal to `NRA`.

NCC — Number of columns of `C`. (Input)
`NCC` must be equal to `NCA`.

C — Real `NRC` by `NCC` rectangular matrix containing the Hadamard product of `A` and `B`. (Output)
If `A` is not needed, then `C` can share the same storage locations as `A`. Similarly, if `B` is not needed, then `C` can share the same storage locations as `B`.

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)

Algorithm

The routine `HRRRR` computes the Hadamard product of two real matrices A and B and returns a real matrix C, where $C_{ij} = A_{ij}B_{ij}$.

Example

Compute the Hadamard product of two 4 × 4 real matrices. The output matrix will be a 4 × 4 real matrix.

```
C                                Declare variables
      INTEGER    LDA, LDB, LDC, NCA, NCB, NCC, NRA, NRB, NRC
      PARAMETER  (LDA=4, LDB=4, LDC=4, NCA=4, NCB=4, NCC=4, NRA=4,
     &           NRB=4, NRC=4)
C
      REAL       A(LDA,NCA), B(LDB,NCB), C(LDC,NCC)
      EXTERNAL   HRRRR, WRRRN
C                                Set values for A
C                                A = ( -1.0  0.0 -3.0  8.0 )
C                                    (  2.0  1.0  7.0  2.0 )
C                                    (  3.0 -2.0  2.0 -6.0 )
C                                    (  4.0  1.0 -5.0 -8.0 )
C
C                                Set values for B
C                                B = (  2.0  3.0  0.0 -10.0 )
C                                    (  1.0 -1.0  4.0   2.0 )
C                                    ( -1.0 -2.0  7.0   1.0 )
C                                    (  2.0  1.0  9.0   0.0 )
C
      DATA A/-1.0, 2.0, 3.0, 4.0, 0.0, 1.0, -2.0, 1.0, -3.0, 7.0, 2.0,
     &     -5.0, 8.0, 2.0, -6.0, -8.0/
      DATA B/2.0, 1.0, -1.0, 2.0, 3.0, -1.0, -2.0, 1.0, 0.0, 4.0, 7.0,
     &     9.0, -10.0, 2.0, 1.0, 0.0/
C                                Compute Hadamard product of A and B
      CALL HRRRR (NRA, NCA, A, LDA, NRB, NCB, B, LDB, NRC, NCC, C, LDC)
```

```
C                                  Print results
      CALL WRRRN ('C = A (*) B', NRC, NCC, C, LDC, 0)
      END
```

Output

```
            C = A (*) B
        1       2       3       4
1   -2.00    0.00    0.00  -80.00
2    2.00   -1.00   28.00    4.00
3   -3.00    4.00   14.00   -6.00
4    8.00    1.00  -45.00    0.00
```

BLINF/DBLINF (Single/Double precision)

Compute the bilinear form $x^T Ay$.

Usage

`BLINF(NRA, NCA, A, LDA, X, Y)`

Arguments

NRA — Number of rows of `A`.   (Input)

NCA — Number of columns of `A`.   (Input)

A — Real `NRA` by `NCA` matrix.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

X — Real vector of length `NRA`.   (Input)

Y — Real vector of length `NCA`.   (Input)

BLINF — The value of $x^T Ay$ is returned in `BLINF`.   (Output)

Comments

The quadratic form can be computed by calling `BLINF` with the vector `X` in place of the vector `Y`.

Algorithm

Given the real rectangular matrix A and two vectors x and y, `BLINF` computes the bilinear form $x^T Ay$.

Example

Compute the bilinear form $x^T Ay$, where x is a vector of length 5, A is a 5×2 matrix and y is a vector of length 2.

```
C                                  Declare variables
      INTEGER    LDA, NCA, NRA
      PARAMETER  (LDA=5, NCA=2, NRA=5)
C
      INTEGER    NOUT
      REAL       A(LDA,NCA), BLINF, VALUE, X(NRA), Y(NCA)
      EXTERNAL   BLINF, UMACH
C                                 Set values for A
C                                 A = ( -2.0  2.0 )
C                                     (  3.0 -6.0 )
C                                     ( -4.0  7.0 )
C                                     (  1.0 -8.0 )
C                                     (  0.0 10.0 )
C                                 Set values for X
C                                 X = (  1.0 -2.0  3.0 -4.0 -5.0 )
C                                 Set values for Y
C                                 Y = ( -6.0  3.0 )
C
      DATA A/-2.0, 3.0, -4.0, 1.0, 0.0, 2.0, -6.0, 7.0, -8.0, 10.0/
      DATA X/1.0, -2.0, 3.0, -4.0, -5.0/
      DATA Y/-6.0, 3.0/
C                                 Compute bilinear form
      VALUE = BLINF(NRA,NCA,A,LDA,X,Y)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The bilinear form trans(x)*A*y = ’, VALUE
      END
```

Output

```
The bilinear form trans(x)*A*y =     195.000
```

POLRG/DPOLRG (Single/Double precision)

Evaluate a real general matrix polynomial.

Usage

`CALL POLRG (N, A, LDA, NCOEF, COEF, B, LDB)`

Arguments

N — Order of the matrix `A`.   (Input)

A — `N` by `N` matrix for which the polynomial is to be computed.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NCOEF — Number of coefficients.   (Input)

COEF — Vector of length `NCOEF` containing the coefficients of the polynomial in order of increasing power.   (Input)

B — `N` by `N` matrix containing the value of the polynomial evaluated at `A`. (Output)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

Comments

Automatic workspace usage is

POLRG N * N units, or
DPOLRG 2 * N * N units.

Workspace may be explicitly provided, if desired, by use of P2LRG/DP2LRG. The reference is

```
CALL P2LRG (N, A, LDA, NCOEF, COEF, B, LDB, WORK)
```

The additional argument is

WORK — Work vector of length N * N.

Algorithm

Let m = NCOEF and c = COEF.

The routine POLRG computes the matrix polynomial

$$B = \sum_{k=1}^{m} c_k A^{k-1}$$

using Horner's scheme

$$B = \left(\ldots \left(\left(c_m A + c_{m-1} I \right) A + c_{m-2} I \right) A + \ldots + c_1 I \right)$$

where I is the $N \times N$ identity matrix.

Example

This example evaluates the matrix polynomial $3I + A + 2A^2$, where A is a 3×3 matrix.

```
C                                  Declare variables
      INTEGER    LDA, LDB, N, NCOEF
      PARAMETER  (N=3, NCOEF=3, LDA=N, LDB=N)
C
      REAL       A(LDA,N), B(LDB,N), COEF(NCOEF)
      EXTERNAL   POLRG, WRRRN
C                                  Set values of A and COEF
C
C                                  A = (  1.0    3.0    2.0  )
C                                      ( -5.0    1.0    7.0  )
C                                      (  1.0    5.0   -4.0  )
C
C                                  COEF = (3.0, 1.0, 2.0)
C
      DATA A/1.0, -5.0, 1.0, 3.0, 1.0, 5.0, 2.0, 7.0, -4.0/
      DATA COEF/3.0, 1.0, 2.0/
C
C                                  Evaluate B = 3I + A + 2*A**2
      CALL POLRG (N, A, LDA, NCOEF, COEF, B, LDB)
C                                  Print B
      CALL WRRRN ('B = 3I + A + 2*A**2', N, N, B, LDB, 0)
      END
```

Output

```
     B = 3I + A + 2*A**2
          1        2       3
1   -20.0     35.0    32.0
2   -11.0     46.0   -55.0
3   -55.0    -19.0   105.0
```

MURRV/DMURRV (Single/Double precision)

Multiply a real rectangular matrix by a vector.

Usage

```
CALL MURRV (NRA, NCA, A, LDA, NX, X, IPATH, NY, Y)
```

Arguments

NRA — Number of rows of A.   (Input)

NCA — Number of columns of A.   (Input)

A — Real NRA by NCA rectangular matrix.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NX — Length of the vector X.   (Input)
NX must be equal to NCA if IPATH is equal to 1. NX must be equal to NRA if IPATH is equal to 2.

X — Real vector of length NX.   (Input)

IPATH — Integer flag.   (Input)
IPATH = 1 means the product Y = A * X is computed. IPATH = 2 means the product Y = trans(A) * X is computed, where trans(A) is the transpose of A.

NY — Length of the vector Y.   (Input)
NY must be equal to NRA if IPATH is equal to 1. NY must be equal to NCA if IPATH is equal to 2.

Y — Real vector of length NY containing the product A * X if IPATH is equal to 1 and the product trans(A) * X if IPATH is equal to 2.   (Output)

Algorithm

If IPATH = 1, MURRV computes $y = Ax$, where A is a real general matrix and x and y are real vectors. If IPATH = 2, MURRV computes $y = A^T x$.

Example

Multiply a 3 × 3 real matrix by a real vector of length 3. The output vector will be a real vector of length 3.

```
C                                  Declare variables
      INTEGER    LDA, NCA, NRA, NX, NY
      PARAMETER  (LDA=3, NCA=3, NRA=3, NX=3, NY=3)
C
      INTEGER    IPATH
      REAL       A(LDA,NCA), X(NX), Y(NY)
      EXTERNAL   MURRV, WRRRN
C                                  Set values for A and X
C                                  A = ( 1.0  0.0  2.0 )
C                                      ( 0.0  3.0  0.0 )
C                                      ( 4.0  1.0  2.0 )
C
C                                  X = ( 1.0  2.0  1.0 )
C
C
      DATA A/1.0, 0.0, 4.0, 0.0, 3.0, 1.0, 2.0, 0.0, 2.0/
      DATA X/1.0, 2.0, 1.0/
C                                  Compute y = Ax
      IPATH = 1
      CALL MURRV (NRA, NCA, A, LDA, NX, X, IPATH, NY, Y)
C                                  Print results
      CALL WRRRN ('y = Ax', 1, NY, Y, 1, 0)
      END
```

Output

```
        y = Ax
    1       2       3
3.000   6.000   8.000
```

MURBV/DMURBV (Single/Double precision)

Multiply a real band matrix in band storage mode by a real vector.

Usage

`CALL MURBV (N, A, LDA, NLCA, NUCA, NX, X, IPATH, NY, Y)`

Arguments

N — Order of the matrix.   (Input)

A — Real `NLCA` + `NUCA` + 1 by `N` band matrix stored in band mode.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals in `A`.   (Input)

NUCA — Number of upper codiagonals in `A`.   (Input)

NX — Length of the vector `X`.   (Input)
`NX` must be equal to `N`.

X — Real vector of length `NX`.   (Input)

IPATH — Integer flag.   (Input)
IPATH = 1 means the product Y = A * X is computed. IPATH = 2 means the product Y = trans(A) * X is computed, where trans(A) is the transpose of A.

NY — Length of vector Y.   (Input)
NY must be equal to N.

Y — Real vector of length NY containing the product A * X if IPATH is equal to 1 and the product trans(A) * X if IPATH is equal to 2.   (Output)

Algorithm

If IPATH = 1, MURBV computes $y = Ax$, where A is a real band matrix and x and y are real vectors. If IPATH = 2, MURBV computes $y = A^T x$.

Example

Multiply a real band matrix of order 6, with two upper codiagonals and two lower codiagonals stored in band mode, by a real vector of length 6. The output vector will be a real vector of length 6.

```
C                                 Declare variables
      INTEGER    LDA, N, NLCA, NUCA, NX, NY
      PARAMETER  (LDA=5, N=6, NLCA=2, NUCA=2, NX=6, NY=6)
C
      INTEGER    IPATH
      REAL       A(LDA,N), X(NX), Y(NY)
      EXTERNAL   MURBV, WRRRN
C                                 Set values for A (in band mode)
C                                 A = ( 0.0  0.0  1.0  2.0  3.0  4.0 )
C                                     ( 0.0  1.0  2.0  3.0  4.0  5.0 )
C                                     ( 1.0  2.0  3.0  4.0  5.0  6.0 )
C                                     (-1.0 -2.0 -3.0 -4.0 -5.0  0.0 )
C                                     (-5.0 -6.0 -7.0 -8.0  0.0  0.0 )
C
C                                 Set values for X
C                                 X = (-1.0  2.0 -3.0  4.0 -5.0  6.0 )
C
      DATA A/0.0, 0.0, 1.0, -1.0, -5.0, 0.0, 1.0, 2.0, -2.0, -6.0,
     &     1.0, 2.0, 3.0, -3.0, -7.0, 2.0, 3.0, 4.0, -4.0, -8.0, 3.0,
     &     4.0, 5.0, -5.0, 0.0, 4.0, 5.0, 6.0, 0.0, 0.0/
      DATA X/-1.0, 2.0, -3.0, 4.0, -5.0, 6.0/
C                                 Compute y = Ax
      IPATH = 1
      CALL MURBV (N, A, LDA, NLCA, NUCA, NX, X, IPATH, NY, Y)
C                                 Print results
      CALL WRRRN (’y = Ax’, 1, NY, Y, 1, 0)
      END
```

Output

```
                     y = Ax
    1       2       3       4       5       6
-2.00    7.00  -11.00   17.00   10.00   29.00
```

MUCRV/DMUCRV (Single/Double precision)

Multiply a complex rectangular matrix by a complex vector.

Usage

CALL MUCRV (NRA, NCA, A, LDA, NX, X, IPATH, NY, Y)

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Complex NRA by NCA rectangular matrix. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

NX — Length of the vector X. (Input)
NX must be equal to NCA if IPATH is equal to 1. NX must be equal to NRA if IPATH is equal to 2.

X — Complex vector of length NX. (Input)

IPATH — Integer flag. (Input)
IPATH = 1 means the product Y = A * X is computed. IPATH = 2 means the product Y = trans(A) * X is computed, where trans(A) is the transpose of A.

NY — Length of the vector Y. (Input)
NY must be equal to NRA if IPATH is equal to 1. NY must be equal to NCA if IPATH is equal to 2.

Y — Complex vector of length NY containing the product A * X if IPATH is equal to 1 and the product trans(A) * X if IPATH is equal to 2. (Output)

Algorithm

If IPATH = 1, MUCRV computes $y = Ax$, where A is a complex general matrix and x and y are complex vectors. If IPATH = 2, MUCRV computes $y = A^T x$.

Example

Multiply a 3 × 3 complex matrix by a complex vector of length 3. The output vector will be a complex vector of length 3.

```
C                                 Declare variables
      INTEGER    LDA, NCA, NRA, NX, NY
      PARAMETER  (LDA=3, NCA=3, NRA=3, NX=3, NY=3)
C
      INTEGER    IPATH
      COMPLEX    A(LDA,NCA), X(NX), Y(NY)
      EXTERNAL   MUCRV, WRCRN
C
C                                 Set values for A and X
```

```
C             A = ( 1.0 + 2.0i  3.0 + 4.0i  1.0 + 0.0i )
C                 ( 2.0 + 1.0i  3.0 + 2.0i  0.0 - 1.0i )
C                 ( 2.0 - 1.0i  1.0 + 0.0i  0.0 + 1.0i )
C
C             X = ( 1.0 - 1.0i  2.0 - 2.0i  0.0 - 1.0i )
C
      DATA A/(1.0,2.0), (2.0,1.0), (2.0,-1.0), (3.0,4.0), (3.0,2.0),
     &     (1.0,0.0), (1.0,0.0), (0.0,-1.0), (0.0,1.0)/
      DATA X/(1.0,-1.0), (2.0,-2.0), (0.0,-1.0)/
C                                Compute y = Ax
      IPATH = 1
      CALL MUCRV (NRA, NCA, A, LDA, NX, X, IPATH, NY, Y)
C                                Print results
      CALL WRCRN ('y = Ax', 1, NY, Y, 1, 0)
      END
```

Output

```
                     y = Ax
              1                 2                 3
( 17.00,  2.00)  ( 12.00, -3.00)  (  4.00, -5.00)
```

MUCBV/DMUCBV (Single/Double precision)

Multiply a complex band matrix in band storage mode by a complex vector.

Usage

```
CALL MUCBV (N, A, LDA, NLCA, NUCA, NX, X, IPATH, NY, Y)
```

Arguments

N — Order of the matrix.   (Input)

A — Complex NLCA + NUCA + 1 by N band matrix stored in band mode.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals in A.   (Input)

NUCA — Number of upper codiagonals in A.   (Input)

NX — Length of the vector X.   (Input)
NX must be equal to N.

X — Complex vector of length NX.   (Input)

IPATH — Integer flag. (Input)
`IPATH` = 1 means the product `Y` = `A * X` is computed. `IPATH` = 2 means the product `Y` = trans(`A`) `* X` is computed, where trans(`A`) is the transpose of `A`.

NY — Length of vector `Y`. (Input)
`NY` must be equal to `N`.

Y — Complex vector of length `NY` containing the product `A * X` if `IPATH` is equal to 1 and the product trans(`A`) `* X` if `IPATH` is equal to 2. (Output)

Algorithm

If `IPATH` = 1, `MUCBV` computes $y = Ax$, where A is a complex band matrix and x and y are complex vectors. If `IPATH` = 2, `MUCBV` computes $y = A^T x$.

Example

Multiply the transpose of a complex band matrix of order 4, with one upper codiagonal and two lower codiagonals stored in band mode, by a complex vector of length 3. The output vector will be a complex vector of length 3.

```
C                                  Declare variables
      INTEGER    LDA, N, NLCA, NUCA, NX, NY
      PARAMETER  (LDA=4, N=4, NLCA=2, NUCA=1, NX=4, NY=4)
C
      INTEGER    IPATH
      COMPLEX    A(LDA,N), X(NX), Y(NY)
      EXTERNAL   MUCBV, WRCRN
C                                    Set values for A (in band mode)
C            A = (  0.0+ 0.0i   1.0+ 2.0i   3.0+ 4.0i   5.0+ 6.0i )
C                ( -1.0- 1.0i  -1.0- 1.0i  -1.0- 1.0i  -1.0- 1.0i )
C                ( -1.0+ 2.0i  -1.0+ 3.0i  -2.0+ 1.0i   0.0+ 0.0i )
C                (  2.0+ 0.0i   0.0+ 2.0i   0.0+ 0.0i   0.0+ 0.0i )
C
C                                   Set values for X
C            X = ( 3.0 + 4.0i  0.0 + 0.0i  1.0 + 2.0i  -2.0 - 1.0i )
C
      DATA A/(0.0,0.0), (-1.0,-1.0), (-1.0,2.0), (2.0,0.0), (1.0,2.0),
     &     (-1.0,-1.0), (-1.0,3.0), (0.0,2.0), (3.0,4.0), (-1.0,-1.0),
     &     (-2.0,1.0), (0.0,0.0), (5.0,6.0), (-1.0,-1.0), (0.0,0.0),
     &     (0.0,0.0)/
      DATA X/(3.0,4.0), (0.0,0.0), (1.0,2.0), (-2.0,-1.0)/
C                                  Compute y = Ax
      IPATH = 2
      CALL MUCBV (N, A, LDA, NLCA, NUCA, NX, X, IPATH, NY, Y)
C                                  Print results
      CALL WRCRN (’y = Ax’, 1, NY, Y, 1, 0)
      END
```

Output

```
                           y = Ax
              1                 2                 3                 4
(  3.00, -3.00)  (-10.00,  7.00)  (  6.00, -3.00)  ( -6.00, 19.00)
```

ARBRB/DARBRB (Single/Double precision)

Add two band matrices, both in band storage mode.

Usage

```
CALL ARBRB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB, C,
            LDC, NLCC, NUCC)
```

Arguments

N — Order of the matrices `A`, `B` and `C`.   (Input)

A — `N` by `N` band matrix with `NLCA` lower codiagonals and `NUCA` upper codiagonals stored in band mode with dimension (`NLCA` + `NUCA` + 1) by `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals of `A`.   (Input)

NUCA — Number of upper codiagonals of `A`.   (Input)

B — `N` by `N` band matrix with `NLCB` lower codiagonals and `NUCB` upper codiagonals stored in band mode with dimension (`NLCB` + `NUCB` + 1) by `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program.   (Input)

NLCB — Number of lower codiagonals of `B`.   (Input)

NUCB — Number of upper codiagonals of `B`.   (Input)

C — `N` by `N` band matrix with `NLCC` lower codiagonals and `NUCC` upper codiagonals containing the sum `A` + `B` in band mode with dimension (`NLCC` + `NUCC` + 1) by `N`.   (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program.   (Input)

NLCC — Number of lower codiagonals of `C`.   (Input)
`NLCC` must be at least as large as max(`NLCA`, `NLCB`).

NUCC — Number of upper codiagonals of `C`.   (Input)
`NUCC` must be at least as large as max(`NUCA`, `NUCB`).

Algorithm

The routine `ARBRB` adds two real matrices stored in band mode, returning a real matrix stored in band mode.

Example

Add two real matrices of order 4 stored in band mode. Matrix *A* has one upper codiagonal and one lower codiagonal. Matrix *B* has no upper codiagonals and two lower codiagonals. The output matrix *C*, has one upper codiagonal and two lower codiagonals.

```
C                                 Declare variables
      INTEGER    LDA, LDB, LDC, N, NLCA, NLCB, NLCC, NUCA, NUCB, NUCC
      PARAMETER  (LDA=3, LDB=3, LDC=4, N=4, NLCA=1, NLCB=2, NLCC=2,
     &           NUCA=1, NUCB=0, NUCC=1)
C
      INTEGER    NBC
      REAL       A(LDA,N), B(LDB,N), C(LDC,N)
      EXTERNAL   ARBRB, WRRRN
C                                 Set values for  A (in band mode)
C                                 A = (  0.0     2.0    3.0   -1.0)
C                                     (  1.0     1.0    1.0    1.0)
C                                     (  0.0     3.0    4.0    0.0)
C
C                                 Set values for  B (in band mode)
C                                 B = (  3.0     3.0    3.0    3.0)
C                                     (  1.0    -2.0    1.0    0.0)
C                                     ( -1.0     2.0    0.0    0.0)
C
      DATA A/0.0, 1.0, 0.0, 2.0, 1.0, 3.0, 3.0, 1.0, 4.0, -1.0, 1.0,
     &     0.0/
      DATA B/3.0, 1.0, -1.0, 3.0, -2.0, 2.0, 3.0, 1.0, 0.0, 3.0, 0.0,
     &     0.0/
C                                 Add A and B to obtain C (in band
C                                                         mode)
      CALL ARBRB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB, C, LDC,
     &            NLCC, NUCC)
C                                 Print results
      NBC = NLCC + NUCC + 1
      CALL WRRRN ('C = A+B', NBC, N, C, LDC, 0)
      END
```

Output

```
             C = A+B
        1       2       3       4
1   0.000   2.000   3.000  -1.000
2   4.000   4.000   4.000   4.000
3   1.000   1.000   5.000   0.000
4  -1.000   2.000   0.000   0.000
```

ACBCB/DACBCB (Single/Double precision)

Add two complex band matrices, both in band storage mode.

Usage

```
CALL ACBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB, C,
            LDC, NLCC, NUCC)
```

Arguments

N — Order of the matrices `A`, `B` and `C`. (Input)

A — `N` by `N` complex band matrix with `NLCA` lower codiagonals and `NUCA` upper codiagonals stored in band mode with dimension (`NLCA` + `NUCA` + 1) by `N`. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of `A`. (Input)

NUCA — Number of upper codiagonals of `A`. (Input)

B — `N` by `N` complex band matrix with `NLCB` lower codiagonals and `NUCB` upper codiagonals stored in band mode with dimension (`NLCB` + `NUCB` + 1) by `N`. (Input)

LDB — Leading dimension of `B` exactly as specified in the dimension statement of the calling program. (Input)

NLCB — Number of lower codiagonals of `B`. (Input)

NUCB — Number of upper codiagonals of `B`. (Input)

C — `N` by `N` complex band matrix with `NLCC` lower codiagonals and `NUCC` upper codiagonals containing the sum `A` + `B` in band mode with dimension (`NLCC` + `NUCC` + 1) by `N`. (Output)

LDC — Leading dimension of `C` exactly as specified in the dimension statement of the calling program. (Input)

NLCC — Number of lower codiagonals of `C`. (Input)
`NLCC` must be at least as large as max(`NLCA`, `NLCB`).

NUCC — Number of upper codiagonals of `C`. (Input)
`NUCC` must be at least as large as max(`NUCA`, `NUCB`).

Algorithm

The routine `ACBCB` adds two complex matrices stored in band mode, returning a complex matrix stored in band mode.

Example

Add two complex matrices of order 4 stored in band mode. Matrix *A* has two upper codiagonals and no lower codiagonals. Matrix *B* has no upper codiagonals and two lower codiagonals. The output matrix *C* has two upper codiagonals and two lower codiagonals.

```
C                                  Declare variables
      INTEGER    LDA, LDB, LDC, N, NLCA, NLCB, NLCC, NUCA, NUCB, NUCC
      PARAMETER  (LDA=3, LDB=3, LDC=5, N=3, NLCA=0, NLCB=2, NLCC=2,
     &           NUCA=2, NUCB=0, NUCC=2)
C
```

```
      INTEGER    NBC
      COMPLEX    A(LDA,N), B(LDB,N), C(LDC,N)
      EXTERNAL   ACBCB, WRCRN
C                                 Set values for A (in band mode)
C                  A = ( 0.0 + 0.0i  0.0 + 0.0i  3.0 - 2.0i )
C                      ( 0.0 + 0.0i  -1.0+ 3.0i  6.0 + 0.0i )
C                      ( 1.0 + 4.0i  5.0 - 2.0i  3.0 + 1.0i )
C
C                                 Set values for B (in band mode)
C                  B = ( 3.0 + 1.0i  4.0 + 1.0i  7.0 - 1.0i )
C                      ( -1.0- 4.0i  9.0 + 3.0i  0.0 + 0.0i )
C                      ( 2.0 - 1.0i  0.0 + 0.0i  0.0 + 0.0i )
C
      DATA A/(0.0,0.0), (0.0,0.0), (1.0,4.0), (0.0,0.0), (-1.0,3.0),
     &     (5.0,-2.0), (3.0,-2.0), (6.0,0.0), (3.0,1.0)/
      DATA B/(3.0,1.0), (-1.0,-4.0), (2.0,-1.0), (4.0,1.0), (9.0,3.0),
     &     (0.0,0.0), (7.0,-1.0), (0.0,0.0), (0.0,0.0)/
C                                 Compute C = A+B
      CALL ACBCB (N, A, LDA, NLCA, NUCA, B, LDB, NLCB, NUCB, C, LDC,
     &            NLCC, NUCC)
C                                 Print results
      NBC = NLCC + NUCC + 1
      CALL WRCRN ('C = A+B', NBC, N, C, LDC, 0)
      END
```

Output

```
                         C = A+B
                1                 2                 3
1  (  0.00,  0.00)  (  0.00,  0.00)  (  3.00, -2.00)
2  (  0.00,  0.00)  ( -1.00,  3.00)  (  6.00,  0.00)
3  (  4.00,  5.00)  (  9.00, -1.00)  ( 10.00,  0.00)
4  ( -1.00, -4.00)  (  9.00,  3.00)  (  0.00,  0.00)
5  (  2.00, -1.00)  (  0.00,  0.00)  (  0.00,  0.00)
```

NRIRR/DNRIRR (Single/Double precision)

Compute the infinity norm of a real matrix.

Usage

`CALL NRIRR (NRA, NCA, A, LDA, ANORM)`

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA matrix whose infinity norm is to be computed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

ANORM — Real scalar containing the infinity norm of A. (Output)

Algorithm

The routine NRIRR computes the infinity norm of a real rectangular matrix *A*. If *m* = NRA and *n* = NCA, then the ∞-norm of *A* is

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} \left| A_{ij} \right|$$

This is the maximum of the sums of the absolute values of the row elements.

Example

Compute the infinity norm of a 3 × 4 real rectangular matrix.

```
C                                 Declare variables
      INTEGER    LDA, NCA, NRA
      PARAMETER  (LDA=3, NCA=4, NRA=3)
C
      INTEGER    NOUT
      REAL       A(LDA,NCA), ANORM
      EXTERNAL   NRIRR, UMACH
C
C                                 Set values for A
C                                 A = ( 1.0  0.0  2.0  0.0 )
C                                     ( 3.0  4.0 -1.0  0.0 )
C                                     ( 2.0  1.0  2.0  1.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
C                                 Compute the infinity norm of A
      CALL NRIRR (NRA, NCA, A, LDA, ANORM)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ' The infinity norm of A is ', ANORM
      END
```

Output

```
The infinity norm of A is     8.00000
```

NR1RR/DNR1RR (Single/Double precision)

Compute the 1-norm of a real matrix.

Usage

```
CALL NR1RR (NRA, NCA, A, LDA, ANORM)
```

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA matrix whose 1-norm is to be computed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

ANORM — Real scalar containing the 1-norm of A. (Output)

Algorithm

The routine NR1RR computes the 1-norm of a real rectangular matrix *A*. If *m* = NRA and *n* = NCA, then the 1-norm of *A* is

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} \left| A_{ij} \right|$$

This is the maximum of the sums of the absolute values of the column elements.

Example

Compute the 1-norm of a 3 × 4 real rectangular matrix.

```
C                              Declare variables
      INTEGER    LDA, NCA, NRA
      PARAMETER  (LDA=3, NCA=4, NRA=3)
C
      INTEGER    NOUT
      REAL       A(LDA,NCA), ANORM
      EXTERNAL   NR1RR, UMACH
C
C                  Set values for A
C                          A = ( 1.0  0.0  2.0  0.0 )
C                              ( 3.0  4.0 -1.0  0.0 )
C                              ( 2.0  1.0  2.0  1.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
C                                  Compute the L1 norm of A
      CALL NR1RR (NRA, NCA, A, LDA, ANORM)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The 1-norm of A is ’, ANORM
      END
```

Output

```
The 1-norm of A is     6.00000
```

NR2RR/DNR2RR (Single/Double precision)

Compute the Frobenius norm of a real rectangular matrix.

Usage

```
CALL NR2RR (NRA, NCA, A, LDA, ANORM)
```

Arguments

NRA — Number of rows of A. (Input)

NCA — Number of columns of A. (Input)

A — Real NRA by NCA rectangular matrix. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement of the calling program. (Input)

ANORM — Frobenius norm of A. (Output)

Algorithm

The routine NR2RR computes the Frobenius norm of a real rectangular matrix *A*. If *m* = NRA and *n* = NCA, then the Frobenius norm of *A* is

$$\|A\|_2 = \left[\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2 \right]^{1/2}$$

Example

Compute the Frobenius norm of a 3×4 real rectangular matrix.

```
C                                  Declare variables
      INTEGER    LDA, NCA, NRA
      PARAMETER  (LDA=3, NCA=4, NRA=3)
C
      INTEGER    NOUT
      REAL       A(LDA,NCA), ANORM
      EXTERNAL   NR2RR, UMACH
C
C                                  Set values for A
C                                  A = ( 1.0  0.0  2.0  0.0 )
C                                      ( 3.0  4.0 -1.0  0.0 )
C                                      ( 2.0  1.0  2.0  1.0 )
C
      DATA A/1.0, 3.0, 2.0, 0.0, 4.0, 1.0, 2.0, -1.0, 2.0, 0.0, 0.0,
     &     1.0/
C
C                                  Compute Frobenius norm of A
      CALL NR2RR (NRA, NCA, A, LDA, ANORM)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The Frobenius norm of A is ’, ANORM
      END
```

Output

```
The Frobenius norm of A is     6.40312
```

NR1RB/DNR1RB (Single/Double precision)

Compute the 1-norm of a real band matrix in band storage mode.

Usage

`CALL NR1RB (N, A, LDA, NLCA, NUCA, ANORM)`

Arguments

N — Order of the matrix. (Input)

A — Real (`NUCA` + `NLCA` + 1) by `N` array containing the `N` by `N` band matrix in band storage mode. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

NLCA — Number of lower codiagonals of `A`. (Input)

NUCA — Number of upper codiagonals of `A`. (Input)

ANORM — Real scalar containing the 1-norm of `A`. (Output)

Algorithm

The routine `NR1RB` computes the 1-norm of a real band matrix *A*. The 1-norm of a matrix *A* is

$$\|A\|_1 = \max_{1 \le j \le N} \sum_{i=1}^{N} \left| A_{ij} \right|$$

This is the maximum of the sums of the absolute values of the column elements.

Example

Compute the 1-norm of a 4 × 4 real band matrix stored in band mode.

```
C                                  Declare variables
      INTEGER    LDA, N, NLCA, NUCA
      PARAMETER  (LDA=4, N=4, NLCA=2, NUCA=1)
C
      INTEGER    NOUT
      REAL       A(LDA,N), ANORM
      EXTERNAL   NR1RB, UMACH
C
C                                  Set values for A (in band mode)
C                                  A = (  0.0  2.0  2.0  3.0  )
C                                      ( -2.0 -3.0 -4.0 -1.0  )
C                                      (  2.0  1.0  0.0  0.0  )
C                                      (  0.0  1.0  0.0  0.0  )
C
      DATA A/0.0, -2.0, 2.0, 0.0, 2.0, -3.0, 1.0, 1.0, 2.0, -4.0, 0.0,
     &     0.0, 3.0, -1.0, 2*0.0/
C                                  Compute the L1 norm of A
      CALL NR1RB (N, A, LDA, NLCA, NUCA, ANORM)
```

```
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The 1-norm of A is ’, ANORM
      END
```

Output

```
The 1-norm of A is     7.00000
```

NR1CB/DNR1CB (Single/Double precision)

Compute the 1-norm of a complex band matrix in band storage mode.

Usage

`CALL NR1CB (N, A, LDA, NLCA, NUCA, ANORM)`

Arguments

N — Order of the matrix.   (Input)

A — Complex (`NUCA` + `NLCA` + 1) by `N` array containing the `N` by `N` band matrix in band storage mode.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

NLCA — Number of lower codiagonals of `A`.   (Input)

NUCA — Number of upper codiagonals of `A`.   (Input)

ANORM — Real scalar containing the 1-norm of `A`.   (Output)

Algorithm

The routine `NR1CB` computes the 1-norm of a complex band matrix *A*. The 1-norm of a complex matrix *A* is

$$\|A\|_1 = \max_{1 \le j \le N} \sum_{i=1}^{N} \left[\left| \Re A_{ij} \right| + \left| \Im A_{ij} \right| \right]$$

Example

Compute the 1-norm of a complex matrix of order 4 in band storage mode.

```
C                                 Declare variables
      INTEGER    LDA, N, NLCA, NUCA
      PARAMETER  (LDA=4, N=4, NLCA=2, NUCA=1)
C
      INTEGER    NOUT
      REAL       ANORM
      COMPLEX    A(LDA,N)
      EXTERNAL   NR1CB, UMACH
C
C                                 Set values for A (in band mode)
C                        A = (  0.0+0.0i  2.0+3.0i -1.0+1.0i -2.0-1.0i )
```

```
C                          ( -2.0+3.0i  1.0+0.0i -4.0-1.0i  0.0-4.0i )
C                          (  2.0+2.0i  4.0+6.0i  3.0+2.0i  0.0+0.0i )
C                          (  0.0-1.0i  2.0+1.0i  0.0+0.0i  0.0+0.0i )
C
      DATA A/(0.0,0.0), (-2.0,3.0), (2.0,2.0), (0.0,-1.0), (2.0,3.0),
     &     (1.0,0.0), (4.0,6.0), (2.0,1.0), (-1.0,1.0), (-4.0,-1.0),
     &     (3.0,2.0), (0.0,0.0), (-2.0,-1.0), (0.0,-4.0), (0.0,0.0),
     &     (0.0,0.0)/
C                                  Compute the L1 norm of A
      CALL NR1CB (N, A, LDA, NLCA, NUCA, ANORM)
C                                  Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ' The 1-norm of A is ', ANORM
      END
```

Output

```
The 1-norm of A is     19.0000
```

DISL2/DDISL2 (Single/Double precision)

Compute the Euclidean (2-norm) distance between two points.

Usage

DISL2(N, X, INCX, Y, INCY)

Arguments

N — Length of the vectors X and Y.   (Input)

X — Vector of length max(N * |INCX|, 1).   (Input)

INCX — Displacement between elements of X.   (Input)
The I-th element of X is X(1 + (I − 1) * INCX) if INCX is greater than or equal to zero or X(1 + (I − N) * INCX) if INCX is less than zero.

Y — Vector of length max(N * |INCY|, 1).   (Input)

INCY — Displacement between elements of Y.   (Input)
The I-th element of Y is Y(1 + (I − 1) * INCY) if INCY is greater than or equal to zero or Y(1 + (I − N) * INCY) if INCY is less than zero.

DISL2 — Euclidean (2-norm) distance between the points X and Y.   (Output)

Algorithm

The function DISL2 computes the Euclidean (2-norm) distance between two points *x* and *y*. The Euclidean distance is defined to be

$$\left[\sum_{i=1}^{N}\left(x_i - y_i\right)^2\right]^{1/2}$$

Example

Compute the Euclidean (2-norm) distance between two vectors of length 4.

```
C                                 Declare variables
      INTEGER    INCX, INCY, N
      PARAMETER  (INCX=1, INCY=1, N=4)
C
      INTEGER    NOUT
      REAL       DISL2, VAL, X(N), Y(N)
      EXTERNAL   DISL2, UMACH
C
C                                 Set values for X and Y
C                                 X = ( 1.0 -1.0  0.0  2.0 )
C
C                                 Y = ( 4.0  2.0  1.0 -3.0 )
C
      DATA X/1.0, -1.0, 0.0, 2.0/
      DATA Y/4.0, 2.0, 1.0, -3.0/
C                                 Compute L2 distance
      VAL = DISL2(N,X,INCX,Y,INCY)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The 2-norm distance is ’, VAL
      END
```

Output

```
The 2-norm distance is     6.63325
```

DISL1/DDISL1 (Single/Double precision)

Compute the 1-norm distance between two points.

Usage

`DISL1(N, X, INCX, Y, INCY)`

Arguments

N — Length of the vectors `X` and `Y`.   (Input)

X — Vector of length max(`N` * |`INCX`|, 1).   (Input)

INCX — Displacement between elements of `X`.   (Input)
The `I`-th element of `X` is `X`(1 + (`I` – 1) * `INCX`) if `INCX` is greater than or equal to zero or `X`(1 + (`I` – `N`) * `INCX`) if `INCX` is less than zero.

Y — Vector of length max(`N` * |`INCY`|, 1).   (Input)

INCY — Displacement between elements of `Y`.   (Input)
The `I`-th element of `Y` is `Y`(1 + (`I` – 1) * `INCY`) if `INCY` is greater than or equal to zero or `Y`(1 + (`I` – `N`) * `INCY`) if `INCY` is less than zero.

DISL1 — 1-norm distance between the points `X` and `Y`.   (Output)

Algorithm

The function `DISL1` computes the 1-norm distance between two points *x* and *y*. The 1-norm distance is defined to be

$$\sum_{i=1}^{N} \left| x_i - y_i \right|$$

Example

Compute the 1-norm distance between two vectors of length 4.

```
C                                 Declare variables
      INTEGER    INCX, INCY, N
      PARAMETER  (INCX=1, INCY=1, N=4)
C
      INTEGER    NOUT
      REAL       DISL1, VAL, X(N), Y(N)
      EXTERNAL   DISL1, UMACH
C
C                                 Set values for X and Y
C                                 X = ( 1.0 -1.0  0.0  2.0 )
C
C                                 Y = ( 4.0  2.0  1.0 -3.0 )
C
      DATA X/1.0, -1.0, 0.0, 2.0/
      DATA Y/4.0, 2.0, 1.0, -3.0/
C                                 Compute L1 distance
      VAL = DISL1(N,X,INCX,Y,INCY)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’ The 1-norm distance is ’, VAL
      END
```

Output

```
The 1-norm distance is     12.0000
```

DISLI/DDISLI (Single/Double precision)

Compute the infinity norm distance between two points.

Usage

`DISLI(N, X, INCX, Y, INCY)`

Arguments

N — Length of the vectors `X` and `Y`. (Input)

X — Vector of length max(`N` * |`INCX`|, 1). (Input)

INCX — Displacement between elements of `X`. (Input)
The `I`-th element of `X` is `X`(1 + (`I` – 1) `*INCX`) if `INCX` is greater than or equal to zero or `X`(1 + (`I` – `N`) `*` `INCX`) if `INCX` is less than zero.

Y — Vector of length max(N * |INCY|, 1). (Input)

INCY — Displacement between elements of Y. (Input)
The I-th element of Y is Y(1 + (I – 1) * INCY) if INCY is greater than or equal to zero or Y(1 + (I – N) * INCY) if INCY is less than zero.

DISLI — Infinity norm distance between the points X and Y. (Output)

Algorithm

The function DISLI computes the 1-norm distance between two points *x* and *y*. The 1norm distance is defined to be

$$\max_{1 \le i \le N} \left| x_i - y_i \right|$$

Example

Compute the ∞-norm distance between two vectors of length 4.

```
C                                 Declare variables
      INTEGER    INCX, INCY, N
      PARAMETER  (INCX=1, INCY=1, N=4)
C
      INTEGER    NOUT
      REAL       DISLI, VAL, X(N), Y(N)
      EXTERNAL   DISLI, UMACH
C
C                                 Set values for X and Y
C                                 X = ( 1.0 -1.0  0.0  2.0 )
C
C                                 Y = ( 4.0  2.0  1.0 -3.0 )
C
      DATA X/1.0, -1.0, 0.0, 2.0/
      DATA Y/4.0, 2.0, 1.0, -3.0/
C                                 Compute L-infinity distance
      VAL = DISLI(N,X,INCX,Y,INCY)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ' The infinity-norm distance is ', VAL
      END
```

Output

```
The infinity-norm distance is     5.00000
```

VCONR/DVCONR (Single/Double precision)

Compute the convolution of two real vectors.

Usage

```
CALL VCONR (NX, X, NY, Y, NZ, Z)
```

Arguments

NX — Length of the vector `X`. (Input)

X — Vector of length `NX`. (Input)

NY — Length of the vector `Y`. (Input)

Y — Vector of length `NY`. (Input)

NZ — Length of the vector `Z`. (Input)
`NZ` must be at least `NX` + `NY` – 1.

Z — Vector of length `NZ` containing the convolution `Z` = `X * Y`. (Output)

Comments

Automatic workspace usage is

`VCONR` 12 * (`NX` + `NY` – 1) + 15 units, or
`DVCONR` 24 * (`NX` + `NY` – 1) + 30 units.

Workspace may be explicitly provided, if desired, by use of `V2ONR`/`DV2ONR`. The reference is

```
CALL V2ONR (NX, X, NY, Y, NZ, Z, XWK, YWK, ZWK, WK)
```

The additional arguments are as follows:

XWK — Complex work array of length `NX` + `NY` – 1.

YWK — Complex work array of length `NX` + `NY` – 1.

ZWK — Complex work array of length `NX` + `NY` – 1.

WK — Real work array of length 6 * (`NX` + `NY` – 1) + 15.

Algorithm

The routine `VCONR` computes the convolution *z* of two real vectors *x* and *y*. Let n_x = `NX`, n_y = `NY` and n_z = `NZ`. The vector *z* is defined to be

$$z_j = \sum_{k=1}^{n_x} x_{j-k+1} y_k \quad \text{for } j = 1, 2, \ldots, n_z$$

where $n_z = n_x + n_y - 1$. If the index $j - k + 1$ is outside the range 1, 2, …, n_x, then x_{j-k+1} is taken to be zero.

The fast Fourier transform is used to compute the convolution. Define the complex vector *u* of length $n_z = n_x + n_y - 1$ to be

$$u = \left(x_1, x_2, \ldots, x_{n_x}, 0, \ldots, 0\right)$$

The complex vector *v*, also of length n_z, is defined similarly using *y*. Then, by the Fourier convolution theorem,

$$\hat{w}_i = \hat{u}_i \hat{v}_i \quad \text{for } i = 1, 2, \ldots, n_z$$

where the $\hat{u}$ indicates the Fourier transform of *u* computed via IMSL routine FFTCF (page 772). IMSL routine FFTCB (page 774) is used to compute the complex vector *w* from $\hat{w}$. The vector *z* is then found by taking the real part of the vector *w*.

Example

In this example, the convolution of a vector *x* of length 8 and a vector *y* of length 3 is computed. The resulting vector *z* is of length 8 + 3 – 1 = 10. (The vector *y* is sometimes called a *filter*.)

```
      INTEGER    NX, NY, NZ
      PARAMETER  (NX=8, NY=3, NZ=NX+NY-1)
C
      REAL       X(NX), Y(NY), Z(NZ)
      EXTERNAL   VCONR, WRRRN
C                                 Set values for X
C                     X = (1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0)
C                                 Set values for Y
C                     Y = (0.0  0.0  1.0)
C
      DATA X/1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0/
      DATA Y/0.0, 0.0, 1.0/
C                                 Compute vector convolution
C                                 Z = X * Y
      CALL VCONR (NX, X, NY, Y, NZ, Z)
C                                 Print results
      CALL WRRRN (’Z = X (*) Y’, 1, NZ, Z, 1, 0)
      END
```

Output

```
                                  Z = X (*) Y
    1      2      3       4       5       6       7       8       9      10
0.000  0.000  1.000   2.000   3.000   4.000   5.000   6.000   7.000   8.000
```

VCONC/DVCONC (Single/Double precision)

Compute the convolution of two complex vectors.

Usage

CALL VCONC (NX, X, NY, Y, NZ, Z)

Arguments

NX — Length of the vector X.   (Input)

X — Complex vector of length NX.   (Input)

NY — Length of the vector Y.   (Input)

Y — Complex vector of length NY.   (Input)

NZ — Length of the vector `Z`. (Input)
`NZ` must be at least `NX` + `NY` – 1.

Z — Complex vector of length `NZ` containing the convolution `Z` = `X * Y`. (Output)

Comments

Automatic workspace usage is

`VCONC` 10 `*` (`NX` + `NY` –1) + 15 units, or
`DVCONC` 20 `*` (`NX` + `NY` –1) + 30 units.

Workspace may be explicitly provided, if desired, by use of `V2ONC`/`DV2ONC`. The reference is

```
CALL V2ONC (NX, X, NY, Y, NZ, Z, XWK, YWK, WK)
```

The additional arguments are as follows:

XWK — Complex work array of length `NX` + `NY` – 1.

YWK — Complex work array of length `NX` + `NY` – 1.

WK — Real work arrary of length 6 `*` (`NX` + `NY` –1) + 15.

Algorithm

The routine `VCONC` computes the convolution *z* of two complex vectors *x* and *y*. Let n_x = `NX`, then n_y = `NY` and n_z = `NZ`. The vector *z* is defined to be

$$z_j = \sum_{k=1}^{n_x} x_{j-k+1} y_k \quad \text{for } j = 1, 2, \ldots, n_z$$

where $n_z = n_x + n_y - 1$. If the index $j - k + 1$ is outside the range 1, 2, …, n_x, then x_{j-k+1} is taken to be zero.

The fast Fourier transform is used to compute the convolution. Define the complex vector *u* of length $n_z = n_x + n_y - 1$ to be

$$u = \left(x_1, x_2, \ldots, x_{n_z}, 0, \ldots, 0\right)$$

The complex vector *v*, also of length n_z, is defined similarly using *y*. Then, by the Fourier convolution theorem,

$$\hat{z}_i = \hat{u}_i \hat{v}_i \quad \text{for } i = 1, 2, \ldots, n_z$$

where the $\hat{u}$ indicates the Fourier transform of *u* computed using IMSL routine `FFTCF` (page 754). The complex vector *z* is computed from $\hat{w}$ via IMSL routine `FFTCB` (page 756).

Example

In this example, the convolution of a vector *x* of length 4 and a vector *y* of length 3 is computed. The resulting vector *z* is of length 4 + 3 – 1 = 6. (The vector *y* is sometimes called a *filter*.)

```
      INTEGER    NX, NY, NZ
      PARAMETER  (NX=4, NY=3, NZ=NX+NY-1)
C
      COMPLEX    X(NX), Y(NY), Z(NZ)
      EXTERNAL   VCONC, WRCRN
C                                 Set values for X
C                X = ( 1.0+2.0i 3.0+4.0i 5.0+6.0i 7.0+8.0i )
C                                 Set values for Y
C                Y = (0.0+0i 0.0+0i 1.0+0i )
C
      DATA X/(1.0,2.0), (3.0,4.0), (5.0,6.0), (7.0,8.0)/
      DATA Y/(0.0,0.0), (0.0,0.0), (1.0,1.0)/
C                                 Compute vector convolution
C                                 Z = X * Y
      CALL VCONC (NX, X, NY, Y, NZ, Z)
C                                 Print results
      CALL WRCRN ('Z = X (*) Y', 1, NZ, Z, 1, 0)
      END
```

Output

```
                           Z = X (*) Y
              1                2                3                4
(  0.00,  0.00)  (  0.00,  0.00)  ( -1.00,  3.00)  ( -1.00,  7.00)

              5                6
( -1.00, 11.00)  ( -1.00, 15.00)
```

Extended Precision Arithmetic

This section describes a set of routines for mixed precision arithmetic. The routines are designed to allow the computation and use of the full quadruple precision result from the multiplication of two double precision numbers. An array called the accumulator stores the result of this multiplication. The result of the multiplication is added to the current contents of the accumulator. It is also possible to add a double precision number to the accumulator or to store a double precision approximation in the accumulator.

The mixed double precision arithmetic routines are described below. The accumulator array, `QACC`, is a double precision array of length 2. Double precision variables are denoted by `DA` and `DB`. Available operations are:

Initialize a real accumulator, `QACC` ← `DA`.

`CALL DQINI (DA, QACC)`

Store a real accumulator, `DA` ← `QACC`.

`CALL DQSTO (QACC, DA)`

Add to a real accumulator, `QACC` ← `QACC` + `DA`.

CALL DQADD (DA, QACC)

Add a product to a real accumulator, QACC ← QACC + DA*DB.

CALL DQMUL (DA, DB, QACC)

There are also mixed double complex arithmetic versions of the above routines. The accumulator, ZACC, is a double precision array of length 4. Double complex variables are denoted by ZA and ZB. Available operations are:

Initialize a complex accumulator, ZACC ← ZA.

CALL ZQINI (ZA, ZACC)

Store a complex accumulator, ZA ← ZACC.

CALL ZQSTO (ZACC, ZA)

Add to a complex accumulator, ZACC ← ZACC + ZA.

CALL ZQADD (ZA, ZACC)

Add a product to a complex accumulator, ZACC ← ZACC + ZA * ZB.

CALL ZQMUL (ZA, ZB, ZACC)

Example

In this example, the value of 1.0D0/3.0D0 is computed in quadruple precision using Newton's method. Four iterations of

$$x_{k+1} = x_k + \left(x_k - a x_k^2\right)$$

with $a = 3$ are taken. The error $ax - 1$ is then computed. The results are accurate to approximately twice the usual double precision accuracy, as given by the IMSL routine DMACH(4), page 1173. Since DMACH is machine dependent, the actual accuracy obtained is also machine dependent.

```
      INTEGER    I, NOUT
      DOUBLE PRECISION A, DACC(2), DMACH, ERROR, SACC(2), X(2), X1, X2
      EXTERNAL   DMACH, DQADD, DQINI, DQMUL, DQSTO, UMACH
C
      CALL UMACH (2, NOUT)
      A = 3.0D0
      CALL DQINI (1.0001D0/A, X)
C                                 Compute X(K+1) = X(K) - A*X(K)*X(K)
C                                 + X(K)
      DO 10  I=1, 4
         X1 = X(1)
         X2 = X(2)
C                                 Compute X + X
         CALL DQADD (X1, X)
         CALL DQADD (X2, X)
C                                 Compute X*X
         CALL DQINI (0.0D0, DACC)
         CALL DQMUL (X1, X1, DACC)
         CALL DQMUL (X1, X2, DACC)
         CALL DQMUL (X1, X2, DACC)
         CALL DQMUL (X2, X2, DACC)
```

```
C                                 Compute -A*(X*X)
         CALL DQINI (0.0D0, SACC)
         CALL DQMUL (-A, DACC(1), SACC)
         CALL DQMUL (-A, DACC(2), SACC)
C                                 Compute -A*(X*X) + (X + X)
         CALL DQADD (SACC(1), X)
         CALL DQADD (SACC(2), X)
   10 CONTINUE
C                                 Compute A*X - 1
      CALL DQINI (0.0D0, SACC)
      CALL DQMUL (A, X(1), SACC)
      CALL DQMUL (A, X(2), SACC)
      CALL DQADD (-1.0D0, SACC)
      CALL DQSTO (SACC, ERROR)
C                                 ERROR should be less than MACHEPS**2
      WRITE (NOUT,99999) ERROR, ERROR/DMACH(4)**2
C
99999 FORMAT (’  A*X - 1 = ’, D15.7, ’ = ’, F10.5, ’*MACHEPS**2’)
      END
```

Output

```
A*X - 1 =   0.6162976D-32 =    0.12500*MACHEPS**2
```

Chapter 10: Utilities

Routines

10.1. Print

10.2. Permute

10.3. Sort

10.4. Search

WRRRN/DWRRRN (Single/Double precision)

Print a real rectangular matrix with integer row and column labels.

Usage

```
CALL WRRRN (TITLE, NRA, NCA, A, LDA, ITRING)
```

Arguments

TITLE — Character string specifying the title. (Input)
TITLE set equal to a blank character(s) suppresses printing of the title. Use “% /” within the title to create a new line. Long titles are automatically wrapped.

NRA — Number of rows. (Input)

NCA — Number of columns. (Input)

A — NRA by NCA matrix to be printed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

ITRING — Triangle option. (Input)

ITRING	Action
0	Full matrix is printed.
1	Upper triangle of A is printed, including the diagonal.
2	Upper triangle of A excluding the diagonal of A is printed.
−1	Lower triangle of A is printed, including the diagonal.
−2	Lower triangle of A excluding the diagonal of A is printed.

Comments

1. A single D, E, or F format is chosen automatically in order to print 4 significant digits for the largest element of A in absolute value. Routine WROPT (page 1130) can be used to change the default format.

2. Horizontal centering, a method for printing large matrices, paging, printing a title on each page, and many other options can be selected by invoking WROPT.

3. A page width of 78 characters is used. Page width and page length can be reset by invoking PGOPT (page 1137).

4. Output is written to the unit specified by UMACH (page 1201).

Algorithm

Routine WRRRN prints a real rectangular matrix with the rows and columns labeled 1, 2, 3, and so on. WRRRN can restrict printing to the elements of the upper or lower triangles of matrices via the ITRING option. Generally, ITRING ≠ 0 is used with symmetric matrices.

In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set NRA to the length of the array and set NCA = 1. For a row vector, set NRA = 1 and set NCA to the length of the array. In both cases, set LDA = NRA and set ITRING = 0.

Example

The following example prints all of a 3 × 4 matrix A where $a_{ij} = i + j/10$.

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, J
      REAL       A(LDA,NCA)
      EXTERNAL   WRRRN
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = I + J*0.1
   10    CONTINUE
   20 CONTINUE
C                                  Write A matrix.
      CALL WRRRN ('A', NRA, NCA, A, LDA, ITRING)
      END
```

Output

```
                  A
        1       2       3       4
1   1.100   1.200   1.300   1.400
2   2.100   2.200   2.300   2.400
3   3.100   3.200   3.300   3.400
```

WRRRL/DWRRRL (Single/Double precision)

Print a real rectangular matrix with a given format and labels.

Usage

```
CALL WRRRL (TITLE, NRA, NCA, A, LDA, ITRING, FMT, RLABEL,
            CLABEL)
```

Arguments

TITLE — Character string specifying the title.   (Input)
TITLE set equal to a blank character(s) suppresses printing of the title.

NRA — Number of rows.   (Input)

NCA — Number of columns.   (Input)

A — NRA by NCA matrix to be printed.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

ITRING — Triangle option.   (Input)

ITRING	**Action**
0	Full matrix is printed.
1	Upper triangle of A is printed, including the diagonal.
2	Upper triangle of A excluding the diagonal of A is printed.
–1	Lower triangle of A is printed, including the diagonal.
–2	Lower triangle of A excluding the diagonal of A is printed.

FMT — Character string containing formats. (Input)
If FMT is set to a blank character(s), the format used is specified by WROPT (page 1130). Otherwise, FMT must contain exactly one set of parentheses and one or more edit descriptors. For example, FMT = ′(F10.3)′ specifies this F format for the entire matrix. FMT = ′(2E10.3, 3F10.3)′ specifies an E format for columns 1 and 2 and an F format for columns 3, 4 and 5. If the end of FMT is encountered and if some columns of the matrix remain, format control continues with the first format in FMT. Even though the matrix A is real, an I format can be used to print the integer part of matrix elements of A. The most useful formats are special formats, called the “V and W formats,” that can be used to specify pretty formats automatically. Set FMT = ′(V10.4)′ if you want a single D, E, or F format selected automatically with field width 10 and with 4 significant digits. Set FMT = ′(W10.4)′ if you want a single D, E, F, or I format selected automatically with field width 10 and with 4 significant digits. While the V format prints trailing zeroes and a trailing decimal point, the W format does not. See Comment 4 for general descriptions of the V and W formats. FMT may contain only D, E, F, G, I, V, or W edit descriptors, e.g., the X descriptor is not allowed.

RLABEL — CHARACTER * (*) vector of labels for rows of A. (Input)
If rows are to be numbered consecutively 1, 2, …, NRA, use RLABEL(1) = ′NUMBER′. If no row labels are desired, use RLABEL(1) = ′NONE′. Otherwise, RLABEL is a vector of length NRA containing the labels.

CLABEL — CHARACTER * (*) vector of labels for columns of A. (Input)
If columns are to be numbered consecutively 1, 2, …, NCA, use CLABEL(1) = ′NUMBER′. If no column labels are desired, use CLABEL(1) = ′NONE′. Otherwise, CLABEL(1) is the heading for the row labels, and either CLABEL(2) must be ′NUMBER′or ′NONE′, or CLABEL must be a vector of length NCA + 1 with CLABEL(1 + *j*) containing the column heading for the *j*-th column.

Comments

1. Automatic workspace is used only if all of the following three conditions are met: (1) FMT contains V or W edit descriptors. (2) FMT is not a single V or W format with no repetition factor. (3) WROPT has previously been invoked with IOPT = –2 and ISET = 0. In this case, workspace usage is

   WRRRL 10 * NCA character units, or
   DWRRRL 10 * NCA character units.

   Workspace may be explicitly provided, if desired, by use of W2RRL/DW2RRL. The reference is

   ```
   CALL W2RRL (TITLE, NRA, NCA, A, LDA, ITRING, FMT,
               RLABEL, CLABEL, CHWK)
   ```

   The additional argument is

CHWK — CHARACTER * 10 work vector of length NCA. This workspace is referenced only if all three conditions indicated at the beginning of this comment are met. Otherwise, CHWK is not referenced and can be a CHARACTER * 10 vector of length one.

2. The output appears in the following form:

```
TITLE
          CLABEL(1)   CLABEL(2)   CLABEL(3)   CLABEL(4)
          RLABEL(1)       xxxxx       xxxxx       xxxxx
          RLABEL(2)       xxxxx       xxxxx       xxxxx
```

3. Use “% /” within titles or labels to create a new line. Long titles or labels are automatically wrapped.

4. For printing numbers whose magnitudes are unknown, the G format in FORTRAN is useful; however, the decimal points will generally not be aligned when printing a column of numbers. The V and W formats are special formats used by this routine to select a D, E, F, or I format so that the decimal points will be aligned. The V and W formats are specified as *Vn.d* and *Wn.d*. Here, n is the field width and d is the number of significant digits generally printed. Valid values for n are 3, 4,…, 40. Valid values for d are 1, 2, …, $n - 2$. If FMT specifies one format and that format is a V or W format, all elements of the matrix A are examined to determine one FORTRAN format for printing. If FMT specifies more than one format, FORTRAN formats are generated separately from each V or W format.

5. A page width of 78 characters is used. Page width and page length can be reset by invoking PGOPT (page 1137).

6. Horizontal centering, method for printing large matrices, paging, method for printing NaN (not a number), printing a title on each page, and many other options can be selected by invoking WROPT (page 1130).

7. Output is written to the unit specified by UMACH (page 1201).

Algorithm

Routine WRRRL prints a real rectangular matrix (stored in A) with row and column labels (specified by RLABEL and CLABEL, respectively) according to a given format (stored in FMT). WRRRL can restrict printing to the elements of upper or lower triangles of matrices via the ITRING option. Generally, ITRING ≠ 0 is used with symmetric matrices.

In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set NRA to the length of the array and set NCA = 1. For a row vector, set NRA = 1 and set NCA to the length of the array. In both cases, set LDA = NRA, and set ITRING = 0.

Example

The following example prints all of a 3 × 4 matrix A where $a_{ij} = (i + j/10)10^{j-3}$.

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, J
      REAL       A(LDA,NCA)
      CHARACTER  CLABEL(5)*5, FMT*8, RLABEL(3)*5
      EXTERNAL   WRRRL
C
      DATA FMT/'(W10.6)'/
      DATA CLABEL/'     ', 'Col 1', 'Col 2', 'Col 3', 'Col 4'/
      DATA RLABEL/'Row 1', 'Row 2', 'Row 3'/
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = (I+J*0.1)*10.0**(J-3)
   10    CONTINUE
   20 CONTINUE
C                                  Write A matrix.
      CALL WRRRL ('A', NRA, NCA, A, LDA, ITRING, FMT, RLABEL, CLABEL)
      END
```

Output

```
                          A
            Col 1       Col 2       Col 3        Col 4
Row 1       0.011       0.120       1.300       14.000
Row 2       0.021       0.220       2.300       24.000
Row 3       0.031       0.320       3.300       34.000
```

WRIRN

Print an integer rectangular matrix with integer row and column labels.

Usage

CALL WRIRN (TITLE, NRMAT, NCMAT, MAT, LDMAT, ITRING)

Arguments

TITLE — Character string specifying the title.   (Input)
TITLE set equal to a blank character(s) suppresses printing of the title. Use “% /” within the title to create a new line. Long titles are automatically wrapped.

NRMAT — Number of rows.   (Input)

NCMAT — Number of columns.   (Input)

MAT — NRMAT by NCMAT matrix to be printed.   (Input)

LDMAT — Leading dimension of MAT exactly as specified in the dimension statement in the calling program.   (Input)

ITRING — Triangle option.   (Input)

ITRING	Action
0	Full matrix is printed.
1	Upper triangle of MAT is printed, including the diagonal.
2	Upper triangle of MAT excluding the diagonal of MAT is printed.
−1	Lower triangle of MAT is printed, including the diagonal.
–2	Lower triangle of MAT excluding the diagonal of MAT is printed.

Comments

1. All the entries in MAT are printed using a single I format. The field width is determined by the largest absolute entry.
2. Horizontal centering, a method for printing large matrices, paging, printing a title on each page, and many other options can be selected by invoking WROPT (page 1130).
3. A page width of 78 characters is used. Page width and page length can be reset by invoking PGOPT (page 1137).
4. Output is written to the unit specified by UMACH (page 1201).

Algorithm

Routine WRIRN prints an integer rectangular matrix with the rows and columns labeled 1, 2, 3, and so on. WRIRN can restrict printing to elements of the upper and lower triangles of matrices via the ITRING option. Generally, ITRING ≠ 0 is used with symmetric matrices.

In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set NRMAT to the length of the array and set NCMAT = 1. For a row vector, set NRMAT = 1 and set NCMAT to the length of the array. In both cases, set LDMAT = NRMAT and set ITRING = 0:

Example

The following example prints all of a 3 × 4 matrix A = MAT where $a_{ij} = 10i + j$.

```
      INTEGER    ITRING, LDMAT, NCMAT, NRMAT
      PARAMETER  (ITRING=0, LDMAT=10, NCMAT=4, NRMAT=3)
C
      INTEGER    I, J, MAT(LDMAT,NCMAT)
      EXTERNAL   WRIRN
C
      DO 20  I=1, NRMAT
         DO 10  J=1, NCMAT
            MAT(I,J) = I*10 + J
   10    CONTINUE
   20 CONTINUE
C                                 Write MAT matrix.
      CALL WRIRN ('MAT', NRMAT, NCMAT, MAT, LDMAT, ITRING)
      END
```

Output

```
         MAT
     1    2    3    4
1   11   12   13   14
2   21   22   23   24
3   31   32   33   34
```

WRIRL

Print an integer rectangular matrix with a given format and labels.

Usage

```
CALL WRIRL (TITLE, NRMAT, NCMAT, MAT, LDMAT, ITRING, FMT,
            RLABEL, CLABEL)
```

Arguments

TITLE — Character string specifying the title. (Input)
`TITLE` set equal to a blank character(s) suppresses printing of the title.

NRMAT — Number of rows. (Input)

NCMAT — Number of columns. (Input)

MAT — `NRMAT` by `NCMAT` matrix to be printed. (Input)

LDMAT — Leading dimension of `MAT` exactly as specified in the dimension statement in the calling program. (Input)

ITRING — Triangle option. (Input)

`ITRING`	**Action**
0	Full matrix is printed.
1	Upper triangle of `MAT` is printed, including the diagonal.
2	Upper triangle of `MAT` excluding the diagonal of `MAT` is printed.
–1	Lower triangle of `MAT` is printed, including the diagonal.
–2	Lower triangle of `MAT` excluding the diagonal of `MAT` is printed.

FMT — Character string containing formats. (Input)
If `FMT` is set to a blank character(s), the format used is a single `I` format with field width determined by the largest absolute entry. Otherwise, `FMT` must contain exactly one set of parentheses and one or more `I` edit descriptors. For example, `FMT` = `′(I10)′` specifies this `I` format for the entire matrix. `FMT` = `′(2I10, 3I5)′` specifies an `I10` format for columns 1 and 2 and an `I5` format for columns 3, 4 and 5. If the end of `FMT` is encountered and if some columns of the matrix remain, format control continues with the first format in `FMT`. `FMT` may only contain the `I` edit descriptor, e.g., the `X` edit descriptor is not allowed.

RLABEL — `CHARACTER * (*)` vector of labels for rows of `MAT`. (Input)
If rows are to be numbered consecutively 1, 2, …, `NRMAT`, use `RLABEL`(1) = `′NUMBER′`. If no row labels are desired, use

RLABEL(1) = 'NONE'. Otherwise, RLABEL is a vector of length NRMAT containing the labels.

CLABEL — CHARACTER * (*) vector of labels for columns of MAT. (Input)
If columns are to be numbered consecutively 1, 2, …, NCMAT, use CLABEL(1) = 'NUMBER'. If no column labels are desired, use CLABEL(1) = 'NONE'. Otherwise, CLABEL(1) is the heading for the row labels, and either CLABEL(2) must be 'NUMBER' or 'NONE', or CLABEL must be a vector of length

NCMAT + 1 with CLABEL(1 + *j*) containing the column heading for the *j*-th column.

Comments

1. The output appears in the following form:

```
TITLE
             CLABEL(1)   CLABEL(2)   CALBEL(3)   CLABEL 4)
             RLABEL(1)       xxxxx       xxxxx       xxxxx
             RLABEL(2)       xxxxx       xxxxx       xxxxx
```

2. Use "% /" within titles or labels to create a new line. Long titles or labels are automatically wrapped.

3. A page width of 78 characters is used. Page width and page length can be reset by invoking PGOPT (page 1137).

4. Horizontal centering, a method for printing large matrices, paging, printing a title on each page, and many other options can be selected by invoking WROPT (page 1130).

5. Output is written to the unit specified by UMACH (page 1201).

Algorithm

Routine WRIRL prints an integer rectangular matrix (stored in MAT) with row and column labels (specified by RLABEL and CLABEL, respectively), according to a given format (stored in FMT). WRIRL can restrict printing to the elements of upper or lower triangles of matrices via the ITRING option. Generally, ITRING ≠ 0 is used with symmetric matrices. In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set NRMAT to the length of the array and set NCMAT = 1. For a row vector, set NRMAT = 1 and set NCMAT to the length of the array. In both cases, set LDMAT = NRMAT, and set ITRING = 0.

Example

The following example prints all of a 3 × 4 matrix A = MAT where $a_{ij} = 10i + j$.

```
INTEGER    ITRING, LDMAT, NCMAT, NRMAT
```

```
      PARAMETER  (ITRING=0, LDMAT=10, NCMAT=4, NRMAT=3)
C
      INTEGER    I, J, MAT(LDMAT,NCMAT)
      CHARACTER  CLABEL(5)*5, FMT*8, RLABEL(3)*5
      EXTERNAL   WRIRL
C
      DATA FMT/'(I2)'/
      DATA CLABEL/'     ', 'Col 1', 'Col 2', 'Col 3', 'Col 4'/
      DATA RLABEL/'Row 1', 'Row 2', 'Row 3'/
C
      DO 20  I=1, NRMAT
         DO 10  J=1, NCMAT
            MAT(I,J) = I*10 + J
   10    CONTINUE
   20 CONTINUE
C                                 Write MAT matrix.
      CALL WRIRL ('MAT', NRMAT, NCMAT, MAT, LDMAT, ITRING, FMT,
     &            RLABEL, CLABEL)
      END
```

Output

```
                   MAT
        Col 1  Col 2  Col 3  Col 4
Row 1     11     12     13     14
Row 2     21     22     23     24
Row 3     31     32     33     34
```

WRCRN/DWRCRN (Single/Double precision)

Print a complex rectangular matrix with integer row and column labels.

Usage

CALL WRCRN (TITLE, NRA, NCA, A, LDA, ITRING)

Arguments

TITLE — Character string specifying the title. (Input)
TITLE set equal to a blank character(s) suppresses printing of the title. Use “% /” within the title to create a new line. Long titles are automatically wrapped.

NRA — Number of rows. (Input)

NCA — Number of columns. (Input)

A — Complex NRA by NCA matrix to be printed. (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program. (Input)

ITRING — Triangle option. (Input)

ITRING	**Action**
0	Full matrix is printed.
1	Upper triangle of A is printed, including the diagonal.

2 Upper triangle of A excluding the diagonal of A is printed.

−1 Lower triangle of A is printed, including the diagonal.

−2 Lower triangle of A excluding the diagonal of A is printed.

Comments

1. A single D, E, or F format is chosen automatically in order to print 4 significant digits for the largest real or imaginary part in absolute value of all the complex numbers in A. Routine WROPT (page 1130) can be used to change the default format.

2. Horizontal centering, a method for printing large matrices, paging, method for printing NaN (not a number), and printing a title on each page can be selected by invoking WROPT.

3. A page width of 78 characters is used. Page width and page length can be reset by invoking subroutine PGOPT (page 1137).

4. Output is written to the unit specified by UMACH (page 1201).

Algorithm

Routine WRCRN prints a complex rectangular matrix with the rows and columns labeled 1, 2, 3, and so on. WRCRN can restrict printing to the elements of the upper or lower triangles of matrices via the ITRING option. Generally, ITRING ≠ 0 is used with Hermitian matrices.

In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set NRA to the length of the array, and set NCA = 1. For a row vector, set NRA = 1, and set NCA to the length of the array. In both cases, set LDA = NRA, and set ITRING = 0.

Example

This example prints all of a 3 × 4 complex matrix A with elements

$$a_{mn} = m + ni, \text{ where } i = \sqrt{-1}$$

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, J
      COMPLEX    A(LDA,NCA), CMPLX
      INTRINSIC  CMPLX
      EXTERNAL   WRCRN
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = CMPLX(I,J)
   10    CONTINUE
   20 CONTINUE
C                                Write A matrix.
      CALL WRCRN ('A', NRA, NCA, A, LDA, ITRING)
      END
```

Output

```
                                  A
                 1                2                3                4
1  ( 1.000, 1.000)  ( 1.000, 2.000)  ( 1.000, 3.000)  ( 1.000, 4.000)
2  ( 2.000, 1.000)  ( 2.000, 2.000)  ( 2.000, 3.000)  ( 2.000, 4.000)
3  ( 3.000, 1.000)  ( 3.000, 2.000)  ( 3.000, 3.000)  ( 3.000, 4.000)
```

WRCRL/DWRCRL (Single/Double precision)

Print a complex rectangular matrix with a given format and labels.

Usage

```
CALL WRCRL (TITLE, NRA, NCA, A, LDA, ITRING, FMT, RLABEL,
            CLABEL)
```

Arguments

TITLE — Character string specifying the title.   (Input)
TITLE set equal to a blank character(s) suppresses printing of the title.

NRA — Number of rows.   (Input)

NCA — Number of columns.   (Input)

A — Complex NRA by NCA matrix to be printed.   (Input)

LDA — Leading dimension of A exactly as specified in the dimension statement in the calling program.   (Input)

ITRING — Triangle option.   (Input)

ITRING	**Action**
0	Full matrix is printed.
1	Upper triangle of A is printed, including the diagonal.
2	Upper triangle of A excluding the diagonal of A is printed.
–1	Lower triangle of A is printed, including the diagonal.
–2	Lower triangle of A excluding the diagonal of A is printed.

FMT — Character string containing formats.   (Input)
If FMT is set to a blank character(s), the format used is specified by WROPT (page 1130). Otherwise, FMT must contain exactly one set of parentheses and one or more edit descriptors. Because a complex number consists of two parts (a real and an imaginary part), two edit descriptors are used for printing a single complex number. FMT = ′(E10.3, F10.3)′ specifies an E format for the real part and an F format for the imaginary part. FMT = ′(F10.3)′ uses an F format for both the real and imaginary parts. If the end of FMT is encountered and if all columns of the matrix have not been printed, format control continues with the first format in FMT. Even though the matrix A is complex, an I format can be used to print the integer parts of the real and imaginary components of each complex number. The most useful formats are special formats, called the “V and W formats,” that can be used to specify pretty formats automatically. Set

FMT = ′(V10.4)′ if you want a single D, E, or F format selected automatially with field width 10 and with 4 significant digits. Set FMT = ′(W10.4)′ if you want a single D, E, F, or I format selected automatically with field width 10 and with 4 significant digits. While the V format prints trailing zeroes and a trailing decimal point, the W format does not. See Comment 4 for general descriptions of the V and W formats. FMT may contain only D, E, F, G, I, V, or W edit descriptors, e.g., the X descriptor is not allowed.

RLABEL — CHARACTER * (*) vector of labels for rows of A. (Input)
If rows are to be numbered consecutively 1, 2, …, NRA, use RLABEL(1) = ′NUMBER′. If no row labels are desired, use RLABEL(1) = ′NONE′. Otherwise, RLABEL is a vector of length NRA containing the labels.

CLABEL — CHARACTER * (*) vector of labels for columns of A. (Input)
If columns are to be numbered consecutively 1, 2, …, NCA, use CLABEL(1) = ′NUMBER′. If no column labels are desired, use CLABEL(1) = ′NONE′. Otherwise, CLABEL(1) is the heading for the row labels, and either CLABEL(2) must be ′NUMBER′ or ′NONE′, or CLABEL must be a vector of length NCA + 1 with CLABEL(1 + *j*) containing the column heading for the *j*-th column.

Comments

1. Automatic workspace is used only if all of the following three conditions are met: (1) FMT contains V or W edit descriptors. (2) FMT is not a single V or W format with no repetition factor. (3) WROPT has previously been invoked with IOPT = –2 and ISET = 0. In this case, workspace usage is

   WRCRL 20 * NCA character units, or
   DWRCRL 20 * NCA character units.

   Workspace may be explicitly provided, if desired, by use of W2CRL/DW2CRL. The reference is

   ```
   CALL W2CRL (TITLE, NRA, NCA, A, LDA, ITRING, FMT,
                RLABEL, CLABEL, CHWK)
   ```

   The additional argument is

   CHWK — CHARACTER * 10 work vector of length 2 * NCA. This workspace is referenced only if all three conditions indicated at the beginning of this comment are met. Otherwise, CHWK is not referenced and can be a CHARACTER * 10 vector of length one.

2. The output appears in the following form:

```
TITLE
         CLABEL(1)        CLABEL(2)        CLABEL(3)        CLABEL(4)
         RLABEL(1)  (xxxxx,xxxxx)  (xxxxx,xxxxx)  (xxxxx,xxxxx)
         RLABEL(2)  (xxxxx,xxxxx)  (xxxxx,xxxxx)  (xxxxx,xxxxx)
```

3. Use “% /” within titles or labels to create a new line. Long titles or labels are automatically wrapped.

4. For printing numbers whose magnitudes are unknown, the `G` format in FORTRAN is useful; however, the decimal points will generally not be aligned when printing a column of numbers. The `V` and `W` formats are special formats used by this routine to select a `D`, `E`, `F`, or `I` format so that the decimal points will be aligned. The `V` and `W` formats are specified as *Vn.d* and *Wn.d*. Here, *n* is the field width, and *d* is the number of significant digits generally printed. Valid values for *n* are 3, 4, …, 40. Valid values for *d* are 1, 2, …, *n* – 2. If `FMT` specifies one format and that format is a `V` or `W` format, all elements of the matrix `A` are examined to determine one FORTRAN format for printing. If `FMT` specifies more than one format, FORTRAN formats are generated separately from each `V` or `W` format.

5. A page width of 78 characters is used. Page width and page length can be reset by invoking `PGOPT` (page 1137).

6. Horizontal centering, a method for printing large matrices, paging, method for printing NaN (not a number), printing a title on each page, and may other options can be selected by invoking `WROPT` (page 1130).

7. Output is written to the unit specified by `UMACH` (page 1201).

Algorithm

Routine `WRCRL` prints a complex rectangular matrix (stored in *A*) with row and column labels (specified by `RLABEL` and `CLABEL`, respectively) according to a given format (stored in `FMT`). Routine `WRCRL` can restrict printing to the elements of upper or lower triangles of matrices via the `ITRING` option. Generally, the `ITRING` ≠ 0 is used with Hermitian matrices.

In addition, one-dimensional arrays can be printed as column or row vectors. For a column vector, set `NRA` to the length of the array, and set `NCA` = 1. For a row vector, set `NRA` = 1, and set `NCA` to the length of the array. In both cases, set `LDA` = `NRA`, and set `ITRING` = 0.

Example

The following example prints all of a 3 × 4 matrix *A* with elements

$$a_{mn} = (m+.123456) + ni, \text{ where } i = \sqrt{-1}$$

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, J
      COMPLEX    A(LDA,NCA), CMPLX
      CHARACTER  CLABEL(5)*5, FMT*8, RLABEL(3)*5
      INTRINSIC  CMPLX
      EXTERNAL   WRCRL
C
```

```
      DATA FMT/'(W12.6)'/
      DATA CLABEL/'    ', 'Col 1', 'Col 2', 'Col 3', 'Col 4'/
      DATA RLABEL/'Row 1', 'Row 2', 'Row 3'/
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = CMPLX(I,J) + 0.123456
   10    CONTINUE
   20 CONTINUE
C                                 Write A matrix.
      CALL WRCRL ('A', NRA, NCA, A, LDA, ITRING, FMT, RLABEL, CLABEL)
      END
```

Output

```
                               A
                             Col 1                          Col 2
Row 1  (     1.12346,     1.00000)  (     1.12346,     2.00000)
Row 2  (     2.12346,     1.00000)  (     2.12346,     2.00000)
Row 3  (     3.12346,     1.00000)  (     3.12346,     2.00000)

                             Col 3                          Col 4
Row 1  (     1.12346,     3.00000)  (     1.12346,     4.00000)
Row 2  (     2.12346,     3.00000)  (     2.12346,     4.00000)
Row 3  (     3.12346,     3.00000)  (     3.12346,     4.00000)
```

WROPT

Set or retrieve an option for printing a matrix.

Usage

`CALL WROPT (IOPT, ISET, ISCOPE)`

Arguments

IOPT — Indicator of option type. (Input)

`IOPT`	**Description of Option Type**
–1, 1	Horizontal centering or left justification of matrix to be printed
–2, 2	Method for printing large matrices
–3, 3	Paging
–4, 4	Method for printing NaN (not a number), and negative and positive machine infinity.
–5, 5	Title option
–6, 6	Default format for real and complex numbers
–7, 7	Spacing between columns
–8, 8	Maximum horizontal space reserved for row labels
–9, 9	Indentation of continuation lines for row labels
–10, 10	Hot zone option for determining line breaks for row labels
–11, 11	Maximum horizontal space reserved for column labels
–12, 12	Hot zone option for determining line breaks for column labels
–13, 13	Hot zone option for determining line breaks for titles

IOPT	Description of Option Type
–14, 14	Option for the label that appears in the upper left hand corner that can be used as a heading for the row numbers or a label for the column headings for WR**N routines
–15, 15	Option for skipping a line between invocations of WR**N routines, provided a new page is not to be issued
–16, 16	Option for vertical alignment of the matrix values relative to the associated row labels that occupy more than one line
0	Reset all the current settings saved in internal variables back to their last setting made with an invocation of WROPT with ISCOPE = 1. (This option is used internally by routines printing a matrix and is not useful otherwise.)

If IOPT is negative, ISET and ISCOPE are input and are saved in internal variables. If IOPT is positive, ISET is output and receives the currently active setting for the option (if ISCOPE = 0) or the last global setting for the option (if ISCOPE = 1). If IOPT = 0, ISET and ISCOPE are not referenced.

ISET — Setting for option selected by IOPT. (Input, if IOPT is negative; output, if IOPT is positive; not referenced if IOPT = 0)

IOPT	ISET	Meaning
–1, 1	0	Matrix is left justified
	1	Matrix is centered horizontally on page
–2, 2	0	A complete row is printed before the next row is printed. Wrapping is used if necessary.
	m	Here, *m* is a positive integer. Let n_1 be the maximum number of columns beginning with column 1 that fit across the page (as determined by the widths of the printing formats). First, columns 1 through n_1 are printed for rows 1 through *m*. Let n_2 be the maximum number of columns beginning with column $n_1 + 1$ that fit across the page. Second, columns $n_1 + 1$ through $n_1 + n_2$ are printed for rows 1 through *m*. This continues until the last columns are printed for rows 1 through *m*. Printing continues in this fashion for the next *m* rows, etc.

IOPT	ISET	Meaning
–3, 3	–2	Printing begins on the next line, and no paging occurs.
	–1	Paging is on. Every invocation of a WR*** routine begins on a new page, and paging occurs within each invocation as is needed
	0	Paging is on. The first invocation of a WR*** routine begins on a new page, and subsequent paging occurs as is needed. With this option, every invocation of a WR*** routine ends with a call to WROPT to reset this option to *k*, a positive integer giving the number of lines printed on the current page.
	k	Here, *k* is a positive integer. Paging is on, and *k* lines have been printed on the current page. If *k* is less than the page length IPAGE (see PGOPT, page 1137), then IPAGE – *k* lines are printed before a new page instruction is issued. If *k* is greater than or equal to IPAGE, then the first invocation of a WR*** routine begins on a new page. In any case, subsequent paging occurs as is needed. With this option, every invocation of a WR*** routine ends with a call to WROPT to reset the value of *k*.
–4, 4	0	NaN is printed as a series of decimal points, negative machine infinity is printed as a series of minus signs, and positive machine infinity is printed as a series of plus signs.
	1	NaN is printed as a series of blank characters, negative machine infinity is printed as a series of minus signs, and positive machine infinity is printed as a series of plus signs.
	2	NaN is printed as “NaN,” negative machine infinity is printed as “-Inf” and positive machine infinity is printed as “Inf.”
	3	NaN is printed as a series of blank characters, negative machine infinity is printed as “-Inf,” and positive machine infinity is printed as “Inf.”
–5, 5	0	Title appears only on first page.
	1	Title appears on the first page and all continuation pages.

IOPT	ISET	Meaning
–6, 6	0	Format is (W10.4). See Comment 2.
	1	Format is (W12.6). See Comment 2.
	2	Format is (1PE12.5).
	3	Format is *Vn*.4 where the field width *n* is determined. See Comment 2.
	4	Format is *Vn*.6 where the field width *n* is determined. See Comment 2.
	5	Format is 1PE*n.d* where $n = d + 7$, and $d + 1$ is the maximum number of significant digits.
–7, 7	k_1	Number of characters left blank between columns. k_1 must be between 0 and 5, inclusively.
–8, 8	k_2	Maximum width (in characters) reserved for row labels. $k_2 = 0$ means use the default.
–9, 9	k_3	Number of characters used to indent continuation lines for row labels. k_3 must be between 0 and 10, inclusively.
–10, 10	k_4	Width (in characters) of the hot zone where line breaks in row labels can occur. $k_4 = 0$ means use the default. k_4 must not exceed 50.
–11, 11	k_5	Maximum width (in characters) reserved for column labels. $k_5 = 0$ means use the default.
–12, 12	k_6	Width (in characters) of the hot zone where line breaks in column labels can occur. $k_6 = 0$ means use the default. k_6 must not exceed 50.
–13, 13	k_7	Width (in characters) of the hot zone where line breaks in titles can occur. k_7 must be between 1 and 50, inclusively.
–14	0	There is no label in the upper left hand corner.
	1	The label in the upper left hand corner is "Component" if a row vector or column vector is printed; the label is "Row/Column" if both the number of rows and columns are greater than one; otherwise, there is no label.
–15	0	A blank line is printed on each invocation of a WR**N routine before the matrix title provided a new page is not to be issued.
	1	A blank line is not printed on each invocation of a WR**N routine before the matrix title.

IOPT	**ISET**	**Meaning**
–16, 16	0	The matrix values are aligned vertically with the last line of the associated row label for the case IOPT = 2 and ISET is positive.
	1	The matrix values are aligned vertically with the first line of the associated row label.

ISCOPE — Indicator of the scope of the option. (Input if IOPT is nonzero; not referenced if IOPT = 0)

ISCOPE	**Action**
0	Setting is temporarily active for the next invocation of a WR*** matrix printing routine.
1	Setting is active until it is changed by another invocation of WROPT.

Comments

1. This program can be invoked repeatedly before using a WR*** routine to print a matrix. The matrix printing routines retrieve these settings to determine the printing options. It is not necessary to call WROPT if a default value of a printing option is desired. The defaults are as follows.

IOPT	**Default Value for ISET**	**Meaning**
1	0	Left justified
2	1000000	Number lines before wrapping
3	–2	No paging
4	2	NaN is printed as “NaN,” negative machine infinity is printed as “-Inf” and positive machine infinity is printed as “Inf.”
5	0	Title only on first page.
6	3	Default format is *Vn*.4.
7	2	2 spaces between columns.
8	0	Maximum row label width MAXRLW = 2 * IPAGEW/3 if matrix has one column; MAXRLW = IPAGEW/4 otherwise.
9	3	3 character indentation of row labels continued beyond one line.
10	0	Width of row label hot zone is MAXRLW/3 characters.

IOPT	Default Value for ISET	Meaning
11	0	Maximum column label width MAXCLW = min{max (NW + NW/2, 15), 40} for integer and real matrices, where NW is the field width for the format corresponding to the particular column. MAXCLW = min{max(NW + NW/2, 15), 83} for complex matrices, where NW is the sum of the two field widths for the formats corresponding to the particular column plus 3.
12	0	Width of column label hot zone is MAXCLW/3 characters.
13	10	Width of hot zone for titles is 10 characters.
14	0	There is no label in the upper left hand corner.
15	0	Blank line is printed.
16	0	The matrix values are aligned vertically with the last line of the associated row label.

For IOPT = 8, the default depends on the current value for the page width, IPAGEW (see PGOPT, page 1137).

2. The V and W formats are special formats that can be used to select a D, E, F, or I format so that the decimal points will be aligned. The V and W formats are specified as *Vn.d* and *Wn.d*. Here, *n* is the field width and *d* is the number of significant digits generally printed. Valid values for *n* are 3, 4, …, 40. Valid values for *d* are 1, 2, …, *n* – 2. While the V format prints trailing zeroes and a trailing decimal point, the W format does not.

Algorithm

Routine WROPT allows the user to set or retrieve an option for printing a matrix. The options controlled by WROPT include the following: horizontal centering, a method for printing large matrices, paging, method for printing NaN (not a number) and positive and negative machine infinities, printing titles, default formats for numbers, spacing between columns, maximum widths reserved for row and column labels, indentation of row labels that continue beyond one line, widths of hot zones for breaking of labels and titles, the default heading for row labels, whether to print a blank line between invocations of routines, and vertical alignment of matrix entries with respect to row labels continued beyond one line. (NaN and positive and negative machine infinities can be retrieved by AMACH and DMACH, page 1201, that are documented in the section “Machine-Dependent Constants” in the Reference Material.) Options can be set globally

(`ISCOPE` = 1) or temporarily for the next call to a printing routine (`ISCOPE` = 0).

Example

The following example illustrates the effect of `WROPT` when printing a 3 × 4 real matrix A with `WRRRN` (page 1116) where $a_{ij} = i + j/10$. The first call to `WROPT` sets horizontal printing so that the matrix is first printed horizontally centered on the page. In the next invocation of `WRRRN`, the left-justification option has been set via routine `WROPT` so the matrix is left justified when printed. Finally, because the scope of left justification was only for the next call to a printing routine, the last call to `WRRRN` results in horizontally centered printing.

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, IOPT, ISCOPE, ISET, J
      REAL       A(LDA,NCA)
      EXTERNAL   WROPT, WRRRN
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = I + J*0.1
   10    CONTINUE
   20 CONTINUE
C                                  Activate centering option.
C                                  Scope is global.
      IOPT   = -1
      ISET   = 1
      ISCOPE = 1
C
      CALL WROPT (IOPT, ISET, ISCOPE)
C                                  Write A matrix.
      CALL WRRRN ('A', NRA, NCA, A, LDA, ITRING)
C                                  Activate left justification.
C                                  Scope is local.
      IOPT   = -1
      ISET   = 0
      ISCOPE = 0
      CALL WROPT (IOPT, ISET, ISCOPE)
      CALL WRRRN ('A', NRA, NCA, A, LDA, ITRING)
      CALL WRRRN ('A', NRA, NCA, A, LDA, ITRING)
      END
```

Output

```
                  A
        1       2       3       4
1   1.100   1.200   1.300   1.400
2   2.100   2.200   2.300   2.400
3   3.100   3.200   3.300   3.400

                  A
        1       2       3       4
1   1.100   1.200   1.300   1.400
2   2.100   2.200   2.300   2.400
3   3.100   3.200   3.300   3.400
```

```
                  A
        1        2        3        4
1   1.100    1.200    1.300    1.400
2   2.100    2.200    2.300    2.400
3   3.100    3.200    3.300    3.400
```

PGOPT

Set or retrieve page width and length for printing.

Usage

`CALL PGOPT (IOPT, IPAGE)`

Arguments

IOPT — Page attribute option. (Input)

IOPT	**Description of Attribute**
–1, 1	Page width.
–2, 2	Page length.

Negative values of `IOPT` indicate the setting `IPAGE` is input. Positive values of `IOPT` indicate the setting `IPAGE` is output.

IPAGE — Value of page attribute. (Input, if `IOPT` is negative; output, if `IOPT` is positive.)

IOPT	**Description of Attribute**	**Settings for IPAGE**
–1, 1	Page width (in characters)	10, 11, …
–2, 2	Page length (in lines)	10, 11, …

Algorithm

Routine `PGOPT` is used to set or retrieve the page width or the page length for routines that perform printing.

Example

The following example illustrates the use of `PGOPT` to set the page width at 20 characters. Routine `WRRRN` (page 1116) is then used to print a 3 × 4 matrix *A* where $a_{ij}= i + j/10$.

```
      INTEGER    ITRING, LDA, NCA, NRA
      PARAMETER  (ITRING=0, LDA=10, NCA=4, NRA=3)
C
      INTEGER    I, IOPT, IPAGE, J
      REAL       A(LDA,NCA)
      EXTERNAL   PGOPT, WRRRN
C
      DO 20  I=1, NRA
         DO 10  J=1, NCA
            A(I,J) = I + J*0.1
   10    CONTINUE
   20 CONTINUE
C                                  Set page width.
      IOPT  = -1
      IPAGE = 20
      CALL PGOPT (IOPT, IPAGE)
C                                  Print the matrix A.
      CALL WRRRN (’A’, NRA, NCA, A, LDA, ITRING)
      END
```

Output

```
         A
        1       2
1   1.100   1.200
2   2.100   2.200
3   3.100   3.200

        3       4
1   1.300   1.400
2   2.300   2.400
3   3.300   3.400
```

PERMU/DPERMU (Single/Double precision)

Rearrange the elements of an array as specified by a permutation.

Usage

`CALL PERMU (N, X, IPERMU, IPATH, XPERMU)`

Arguments

N — Length of the arrays `X` and `XPERMU`.   (Input)

X — Real vector of length `N` containing the array to be permuted.   (Input)

IPERMU — Integer vector of length `N` containing a permutation `IPERMU`(1), …, `IPERMU`(`N`) of the integers 1, …, `N`.   (Input)

IPATH — Integer flag.   (Input)
`IPATH` = 1 means `IPERMU` represents a forward permutation, i.e., `X`(`IPERMU`(`I`)) is moved to `XPERMU`(`I`). `IPATH` = 2 means `IPERMU` represents a backward permutation, i.e., `X`(`I`) is moved to `XPERMU`(`IPERMU`(`I`)).

XPERMU — Real vector of length `N` containing the array `X` permuted.   (Output)
If `X` is not needed, `X` and `XPERMU` can share the same storage locations.

Algorithm

Routine `PERMU` rearranges the elements of an array according to a permutation vector. It has the option to do both forward and backward permutations.

Example

This example rearranges the array *X* using `IPERMU`; forward permutation is performed.

```
C                                 Declare variables
      INTEGER    IPATH, N
      PARAMETER  (IPATH=1, N=4)
C
      INTEGER    IPERMU(N), J, NOUT
      REAL       X(N), XPERMU(N)
      EXTERNAL   PERMU, UMACH
C                                 Set values for  X, IPERMU
C
C                           X = ( 5.0  6.0  1.0  4.0 )
C                           IPERMU = ( 3 1 4 2 )
C
      DATA X/5.0, 6.0, 1.0, 4.0/, IPERMU/3, 1, 4, 2/
C                                 Permute X into XPERMU
      CALL PERMU (N, X, IPERMU, IPATH, XPERMU)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print results
      WRITE (NOUT,99999) (XPERMU(J),J=1,N)
C
99999 FORMAT ('  The output vector is:', /, 10(1X,F10.2))
      END
```

Output

```
The Output vector is:
1.00       5.00       4.00       6.00
```

PERMA/DPERMA (Single/Double precision)

Permute the rows or columns of a matrix.

Usage

`CALL PERMA (NRA, NCA, A, LDA, IPERMU, IPATH, APER, LDAPER)`

Arguments

NRA — Number of rows.   (Input)

NCA — Number of columns.   (Input)

A — `NRA` by `NCA` matrix to be permuted.   (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program.   (Input)

IPERMU — Vector of length `K` containing a permutation `IPERMU`(1), …, `IPERMU`(`K`) of the integers 1, …, `K` where `K` = `NRA` if the rows of `A` are to be permuted and `K` = `NCA` if the columns of `A` are to be permuted.   (Input)

IPATH — Option parameter.   (Input)
IPATH = 1 means the rows of A will be permuted. IPATH = 2 means the columns of A will be permuted.

APER — NRA by NCA matrix containing the permuted matrix.   (Output)
If A is not needed, A and APER can share the same storage locations.

LDAPER — Leading dimension of APER exactly as specified in the dimension statement of the calling program.   (Input)

Comments

Automatic workspace usage is

PERMA NCA units, or
DPERMA 2 * NCA units.

Workspace may be explicitly provided, if desired, by use of P2RMA/DP2RMA. The reference is

```
CALL P2RMA (NRA, NCA, A, LDA, IPERMU, IPATH, APER, LDAPER,
            WORK)
```

The additional argument is

WORK — Real work vector of length NCA.

Algorithm

Routine PERMA interchanges the rows or columns of a matrix using a permutation vector such as the one obtained from routines SVRBP (page 1146) or SVRGP (page 1142).

The routine PERMA permutes a column (row) at a time by calling PERMU (page 1138). This process is continued until all the columns (rows) are permuted. On completion, let B = APER and p_i = IPERMU(I), then

$$B_{ij} = A_{p_i j}$$

for all i, j.

Example

This example permutes the columns of a matrix A.

```
C                                  Declare variables
      INTEGER    IPATH, LDA, LDAPER, NCA, NRA
      PARAMETER  (IPATH=2, LDA=3, LDAPER=3, NCA=5, NRA=3)
C
      INTEGER    I, IPERMU(5), J, NOUT
      REAL       A(LDA,NCA), APER(LDAPER,NCA)
      EXTERNAL   PERMA, UMACH
C                                  Set values for  A, IPERMU
C                                  A = ( 3.0  5.0  1.0  2.0  4.0 )
C                                      ( 3.0  5.0  1.0  2.0  4.0 )
C                                      ( 3.0  5.0  1.0  2.0  4.0 )
C
```

```
C                                  IPERMU = ( 3 4 1 5 2 )
C
      DATA A/3*3.0, 3*5.0, 3*1.0, 3*2.0, 3*4.0/, IPERMU/3, 4, 1, 5, 2/
C                                  Perform column permutation on A,
C                                  giving APER
      CALL PERMA (NRA, NCA, A, LDA, IPERMU, IPATH, APER, LDAPER)
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Print results
      WRITE (NOUT,99999) ((APER(I,J),J=1,NCA),I=1,NRA)
C
99999 FORMAT ('  The output matrix is:', /, 3(5F8.1,/))
      END
```

Output

```
The Output matrix is:
1.0     2.0     3.0     4.0     5.0
1.0     2.0     3.0     4.0     5.0
1.0     2.0     3.0     4.0     5.0
```

SVRGN/DSVRGN (Single/Double precision)

Sort a real array by algebraically increasing value.

Usage

```
CALL SVRGN (N, RA, RB)
```

Arguments

N — Number of elements in the array to be sorted. (Input)

RA — Vector of length `N` containing the array to be sorted. (Input)

RB — Vector of length `N` containing the sorted array. (Output)
If `RA` is not needed, `RA` and `RB` can share the same storage locations.

Algorithm

Routine `SVRGN` sorts the elements of an array, A, into ascending order by algebraic value. The array A is divided into two parts by picking a central element T of the array. The first and last elements of A are compared with T and exchanged until the three values appear in the array in ascending order. The elements of the array are rearranged until all elements greater than or equal to the central element appear in the second part of the array and all those less than or equal to the central element appear in the first part. The upper and lower subscripts of one of the segments are saved, and the process continues iteratively on the other segment. When one segment is finally sorted, the process begins again by retrieving the subscripts of another unsorted portion of the array. On completion, $A_j \le A_i$ for $j < i$. For more details, see Singleton (1969), Griffin and Redish (1970), and Petro (1970).

Example

This example sorts the 10-element array `RA` algebraically.

```
C                                 Declare variables
      PARAMETER  (N=10)
      REAL       RA(N), RB(N)
C                                 Set values for  RA
C     RA = ( -1.0  2.0  -3.0  4.0  -5.0  6.0  -7.0  8.0  -9.0  10.0 )
C
      DATA RA/-1.0, 2.0, -3.0, 4.0, -5.0, 6.0, -7.0, 8.0, -9.0, 10.0/
C                                 Sort RA by algebraic value into RB
      CALL SVRGN (N, RA, RB)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99999) (RB(J),J=1,N)
C
99999 FORMAT ('  The output vector is:', /, 10(1X,F5.1))
      END
```

Output

```
The Output vector is:
-9.0  -7.0  -5.0  -3.0  -1.0   2.0   4.0   6.0   8.0  10.0
```

SVRGP/DSVRGP (Single/Double precision)

Sort a real array by algebraically increasing value and return the permutation that rearranges the array.

Usage

`CALL SVRGP (N, RA, RB, IPERM)`

Arguments

N — Number of elements in the array to be sorted.   (Input)

RA — Vector of length `N` containing the array to be sorted.   (Input)

RB — Vector of length `N` containing the sorted array.   (Output)
If `RA` is not needed, `RA` and `RB` can share the same storage locations.

IPERM — Vector of length `N`.   (Input/Output)
On input, `IPERM` should be initialized to the values 1, 2, …, `N`. On output, `IPERM` contains a record of permutations made on the vector `RA`.

Comments

For wider applicability, integers (1, 2, …, `N`) that are to be associated with `RA(I)` for `I` = 1, 2, …, `N` may be entered into `IPERM(I)` in any order. Note that these integers must be unique.

Algorithm

Routine SVRGP sorts the elements of an array, *A*, into ascending order by algebraic value, keeping a record in *P* of the permutations to the array *A*. That is, the elements of *P* are moved in the same manner as are the elements in *A* as *A* is being sorted. The routine SVRGP uses the algorithm discussed in SVRGN (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array RA algebraically.

```
C                                 Declare variables
      PARAMETER  (N=10)
      REAL       RA(N), RB(N)
      INTEGER    IPERM(N)
C                                 Set values for  RA and IPERM
C     RA    = ( 10.0  -9.0  8.0  -7.0  6.0  5.0  4.0  -3.0  -2.0  -1.0 )
C
C     IPERM = ( 1  2  3  4  5  6  7  8  9  10)
C
      DATA RA/10.0, -9.0, 8.0, -7.0, 6.0, 5.0, 4.0, -3.0, -2.0, -1.0/
      DATA IPERM/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
C                                 Sort RA by algebraic value into RB
      CALL SVRGP (N, RA, RB, IPERM)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99998) (RB(J),J=1,N)
      WRITE (NOUT, 99999) (IPERM(J),J=1,N)
C
99998 FORMAT (’  The output vector is:’, /, 10(1X,F5.1))
99999 FORMAT (’  The permutation vector is:’, /, 10(1X,I5))
      END
```

Output

```
The output vector is:
-9.0  -7.0  -3.0  -2.0  -1.0   4.0   5.0   6.0   8.0  10.0

The permutation vector is:
2     4     8     9    10     7     6     5     3     1
```

SVIGN

Sort an integer array by algebraically increasing value.

Usage

CALL SVIGN (N, IA, IB)

Arguments

N — Number of elements in the array to be sorted.   (Input)

IA — Integer vector of length N containing the array to be sorted.   (Input)

IB — Integer vector of length `N` containing the sorted array. (Output)
If `IA` is not needed, `IA` and `IB` can share the same storage locations.

Algorithm

Routine `SVIGN` sorts the elements of an integer array, *A*, into ascending order by algebraic value. The routine `SVIGN` uses the algorithm discussed in `SVRGN` (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array `IA` algebraically.

```
C                                 Declare variables
      PARAMETER  (N=10)
      INTEGER    IA(N), IB(N)
C                                 Set values for  IA
C     IA = ( -1  2  -3  4  -5  6  -7  8  -9  10 )
C
      DATA IA/-1, 2, -3, 4, -5, 6, -7, 8, -9, 10/
C                                 Sort IA by algebraic value into IB
      CALL SVIGN (N, IA, IB)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99999) (IB(J),J=1,N)
C
99999 FORMAT ('  The output vector is:', /, 10(1X,I5))
      END
```

Output

```
The Output vector is:
-9    -7    -5    -3    -1     2     4     6     8    10
```

SVIGP

Sort an integer array by algebraically increasing value and return the permutation that rearranges the array.

Usage

`CALL SVIGP (N, IA, IB, IPERM)`

Arguments

N — Number of elements in the array to be sorted. (Input)

IA — Integer vector of length `N` containing the array to be sorted. (Input)

IB — Integer vector of length `N` containing the sorted array. (Output)
If `IA` is not needed, `IA` and `IB` can share the same storage locations.

IPERM — Vector of length `N`. (Input/Output)
On input, `IPERM` should be initialized to the values 1, 2, …, `N`. On output, `IPERM` contains a record of permutations made on the vector `IA`.

Comments

For wider applicability, integers (1, 2, …, N) that are to be associated with IA(I) for I = 1, 2, …, N may be entered into IPERM(I) in any order. Note that these integers must be unique.

Algorithm

Routine SVIGP sorts the elements of an integer array, *A*, into ascending order by algebraic value, keeping a record in *P* of the permutations to the array *A*. That is, the elements of *P* are moved in the same manner as are the elements in *A* as *A* is being sorted. The routine SVIGP uses the algorithm discussed in SVRGN (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array IA algebraically.

```
C                                 Declare variables
      PARAMETER  (N=10)
      INTEGER    IA(N), IB(N), IPERM(N)
C                                 Set values for  IA and IPERM
C     IA    = ( 10  -9  8  -7  6  5  4  -3  -2  -1 )
C
C     IPERM = ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 )
C
      DATA IA/10, -9, 8, -7, 6, 5, 4, -3, -2, -1/
      DATA IPERM/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
C                                 Sort IA by algebraic value into IB
      CALL SVIGP (N, IA, IB, IPERM)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99998) (IB(J),J=1,N)
      WRITE (NOUT, 99999) (IPERM(J),J=1,N)
C
99998 FORMAT (’ The output vector is:’, /, 10(1X,I5))
99999 FORMAT (’ The permutation vector is:’, /, 10(1X,I5))
      END
```

Output

```
The Output vector is:
-9    -7    -3    -2    -1     4     5     6     8    10

The permutation vector is:
2     4     8     9    10     7     6     5     3     1
```

SVRBN/DSVRBN (Single/Double precision)

Sort a real array by nondecreasing absolute value.

Usage

CALL SVRBN (N, RA, RB)

Arguments

N — Number of elements in the array to be sorted.   (Input)

RA — Vector of length `N` containing the array to be sorted.   (Input)

RB — Vector of length `N` containing the sorted array.   (Output)
If `RA` is not needed, `RA` and `RB` can share the same storage locations.

Algorithm

Routine `SVRBN` sorts the elements of an array, *A*, into ascending order by absolute value. The routine `SVRBN` uses the algorithm discussed in `SVRGN` (page 1141). On completion, $|A_j| \le |A_i|$ for $j < i$.

Example

This example sorts the 10-element array `RA` by absolute value.

```
C                                 Declare variables
      PARAMETER  (N=10)
      REAL       RA(N), RB(N)
C                                 Set values for  RA
C        RA = ( -1.0  3.0  -4.0  2.0  -1.0  0.0  -7.0  6.0  10.0  -7.0 )
C
      DATA RA/-1.0, 3.0, -4.0, 2.0, -1.0, 0.0, -7.0, 6.0, 10.0, -7.0/
C                                 Sort RA by absolute value into RB
      CALL SVRBN (N, RA, RB)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99999) (RB(J),J=1,N)
C
99999 FORMAT ('  The output vector is :', /, 10(1X,F5.1))
      END
```

Output

```
The Output vector is :
0.0  -1.0  -1.0   2.0   3.0  -4.0   6.0  -7.0  -7.0  10.0
```

SVRBP/DSVRBP (Single/Double precision)

Sort a real array by nondecreasing absolute value and return the permutation that rearranges the array.

Usage

`CALL SVRBP (N, RA, RB, IPERM)`

Arguments

N — Number of elements in the array to be sorted.   (Input)

RA — Vector of length `N` containing the array to be sorted.   (Input)

RB — Vector of length N containing the sorted array. (Output)
If RA is not needed, RA and RB can share the same storage locations.

IPERM — Vector of length N. (Input/Output)
On input, IPERM should be initialized to the values 1, 2, …, N. On output, IPERM contains a record of permutations made on the vector IA.

Comments

For wider applicability, integers (1, 2, …, N) that are to be associated with RA(I) for I = 1, 2, …, N may be entered into IPERM(I) in any order. Note that these integers must be unique.

Algorithm

Routine SVRBP sorts the elements of an array, A, into ascending order by absolute value, keeping a record in P of the permutations to the array A. That is, the elements of P are moved in the same manner as are the elements in A as A is being sorted. The routine SVRBP uses the algorithm discussed in SVRGN (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array RA by absolute value.

```
C                                 Declare variables
      PARAMETER  (N=10)
      REAL       RA(N), RB(N)
      INTEGER    IPERM(N)
C                                 Set values for  RA and IPERM
C     RA     = ( 10.0  9.0  8.0  7.0  6.0  5.0  -4.0  3.0  -2.0  1.0 )
C
C     IPERM = ( 1  2  3  4  5  6  7  8  9  10 )
C
      DATA RA/10.0, 9.0, 8.0, 7.0, 6.0, 5.0, -4.0, 3.0, -2.0, 1.0/
      DATA IPERM/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
C                                 Sort RA by absolute value into RB
      CALL SVRBP (N, RA, RB, IPERM)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99998) (RB(J),J=1,N)
      WRITE (NOUT, 99999) (IPERM(I),I=1,N)
C
99998 FORMAT ('  The output vector is:', /, 10(1X,F5.1))
99999 FORMAT ('  The permutation vector is:', /, 10(1X,I5))
      END
```

Output

```
The output vector is:
1.0  -2.0   3.0  -4.0   5.0   6.0   7.0   8.0   9.0  10.0
The permutation vector is:
10     9     8     7     6     5     4     3     2     1
```

SVIBN

Sort an integer array by nondecreasing absolute value.

Usage

`CALL SVIBN (N, IA, IB)`

Arguments

N — Number of elements in the array to be sorted. (Input)

IA — Integer vector of length `N` containing the array to be sorted. (Input)

IB — Integer vector of length `N` containing the sorted array. (Output)
If `IA` is not needed, `IA` and `IB` can share the same storage locations.

Algorithm

Routine `SVIBN` sorts the elements of an integer array, *A*, into ascending order by absolute value. This routine `SVIBN` uses the algorithm discussed in `SVRGN` (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array `IA` by absolute value.

```
C                                 Declare variables
      PARAMETER  (N=10)
      INTEGER    IA(N), IB(N)
C                                 Set values for  IA
C     IA = ( -1  3  -4  2  -1  0  -7  6  10  -7)
C
      DATA IA/-1, 3, -4, 2, -1, 0, -7, 6, 10, -7/
C                                 Sort IA by absolute value into IB
      CALL SVIBN (N, IA, IB)
C                                 Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99999) (IB(J),J=1,N)
C
99999 FORMAT ('  The output vector is:', /, 10(1X,I5))
      END
```

Output

```
The Output vector is:
0    -1    -1     2     3    -4     6    -7    -7    10
```

SVIBP

Sort an integer array by nondecreasing absolute value and return the permutation that rearranges the array.

Usage

CALL SVIBP (N, IA, IB, IPERM)

Arguments

N — Number of elements in the array to be sorted. (Input)

IA — Integer vector of length N containing the array to be sorted. (Input)

IB — Integer vector of length N containing the sorted array. (Output)
If IA is not needed, IA and IB can share the same storage locations.

IPERM — Vector of length N. (Input/Output)
On input, IPERM should be initialized to the values 1, 2, …, N. On output, IPERM contains a record of permutations made on the vector IA.

Comments

For wider applicability, integers (1, 2, …, N) that are to be associated with IA(I) for I = 1, 2, …, N may be entered into IPERM(I) in any order. Note that these integers must be unique.

Algorithm

Routine SVIBP sorts the elements of an integer array, *A*, into ascending order by absolute value, keeping a record in *P* of the permutations to the array *A*. That is, the elements of *P* are moved in the same manner as are the elements in *A* as *A* is being sorted. The routine SVIBP uses the algorithm discussed in SVRGN (page 1141). On completion, $A_j \le A_i$ for $j < i$.

Example

This example sorts the 10-element array IA by absolute value.

```
C                                  Declare variables
      PARAMETER  (N=10)
      INTEGER    IA(N), IB(N), IPERM(N)
C                                  Set values for  IA
C     IA    = ( 10  9  8  7  6  5  -4  3  -2  1 )
C
C     IPERM = ( 1  2  3  4  5  6  7  8  9  10 )
C
      DATA IA/10, 9, 8, 7, 6, 5, -4, 3, -2, 1/
      DATA IPERM/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
C                                  Sort IA by absolute value into IB
      CALL SVIBP (N, IA, IB, IPERM)
C                                  Print results
      CALL UMACH (2,NOUT)
      WRITE (NOUT, 99998) (IB(J),J=1,N)
      WRITE (NOUT, 99999) (IPERM(J),J=1,N)
C
99998 FORMAT ('  The output vector is:', /, 10(1X,I5))
99999 FORMAT ('  The permutation vector is:', /, 10(1X,I5))
      END
```

Output

```
The Output vector is:
1    -2     3    -4     5     6     7     8     9    10

The permutation vector is:
10     9     8     7     6     5     4     3     2     1
```

SRCH/DSRCH (Single/Double precision)

Search a sorted vector for a given scalar and return its index.

Usage

CALL SRCH (N, VALUE, X, INCX, INDEX)

Arguments

N — Length of vector Y. (Input)

VALUE — Scalar to be searched for in Y. (Input)

X — Vector of length N * INCX. (Input)
Y is obtained from X for I = 1, 2, …, N by Y(I) = X(1 + (I – 1) * INCX). Y(1), Y(2), …, Y(N) must be in ascending order.

INCX — Displacement between elements of X. (Input)
INCX must be greater than zero.

INDEX — Index of Y pointing to VALUE. (Output)
If INDEX is positive, VALUE is found in Y. If INDEX is negative, VALUE is not found in Y.

INDEX	**Location of VALUE**
1 thru N	VALUE = Y(INDEX)
–1	VALUE < Y(1) or N = 0
–N thru –2	Y(–INDEX – 1) < VALUE < Y(INDEX)
–(N + 1)	VALUE > Y(N)

Algorithm

Routine SRCH searches a real vector x (stored in X), whose n elements are sorted in ascending order for a real number c (stored in VALUE). If c is found in x, its index i (stored in INDEX) is returned so that $x_i = c$. Otherwise, a negative number i is returned for the index. Specifically,

if $1 \le i \le n$	then $x_i = c$
if $i = -1$	then $c < x_1$ or $n = 0$
if $-n \le i \le -2$	then $x_{-i-1} < c < x_{-i}$
if $i = -(n + 1)$	then $c > x_n$

The argument `INCX` is useful if a row of a matrix, for example, row number `I` of a matrix `X`, must be searched. The elements of row `I` are assumed to be in ascending order. In this case, set `INCX` equal to the leading dimension of `X` exactly as specified in the dimension statement in the calling program. With `X` declared

```
REAL X(LDX,N)
```

the invocation

```
CALL SRCH (N, VALUE, X(I,1), LDX, INDEX)
```

returns an index that will reference a column number of `X`.

Routine `SRCH` performs a binary search. The routine is an implementation of Algorithm *B* discussed by Knuth (1973, pages 407−411).

Example

This example searches a real vector sorted in ascending order for the value 653.0. The problem is discussed by Knuth (1973, pages 407−409).

```
      INTEGER    N
      PARAMETER  (N=16)
C
      INTEGER    INCX, INDEX, NOUT
      REAL       VALUE, X(N)
      EXTERNAL   SRCH, UMACH
C
      DATA X/61.0, 87.0, 154.0, 170.0, 275.0, 426.0, 503.0, 509.0,
     &     512.0, 612.0, 653.0, 677.0, 703.0, 765.0, 897.0, 908.0/
C
      INCX  = 1
      VALUE = 653.0
      CALL SRCH (N, VALUE, X, INCX, INDEX)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) 'INDEX = ', INDEX
      END
```

Output

```
INDEX =   11
```

ISRCH

Search a sorted integer vector for a given integer and return its index.

Usage

```
CALL ISRCH (N, IVALUE, IX, INCX, INDEX)
```

Arguments

N — Length of vector `IY`. (Input)

IVALUE — Scalar to be searched for in `IY`. (Input)

IX — Vector of length `N * INCX`. (Input)
`IY` is obtained from `IX` for `I` = 1, 2, …, `N` by `IY`(`I`) = `IX`(1 + (`I` – 1) `* INCX`). `IY`(1), `IY`(2), …, `IY`(`N`) must be in ascending order.

INCX — Displacement between elements of `IX`. (Input)
`INCX` must be greater than zero.

INDEX — Index of `IY` pointing to `IVALUE`. (Output)
If `INDEX` is positive, `IVALUE` is found in `IY`. If `INDEX` is negative, `IVALUE` is not found in `IY`.

`INDEX`	**Location of `VALUE`**
1 thru `N`	`IVALUE` = `IY`(`INDEX`)
–1	`IVALUE` < `IY`(1) or `N` = 0
–`N` thru –2	`IY`(–`INDEX` – 1) < `IVALUE` < `IY`(–`INDEX`)
–(`N` + 1)	`IVALUE` > `Y`(`N`)

Algorithm

Routine `ISRCH` searches an integer vector *x* (stored in `IX`), whose *n* elements are sorted in ascending order for an integer *c* (stored in `IVALUE`). If *c* is found in *x*, its index *i* (stored in `INDEX`) is returned so that $x_i = c$. Otherwise, a negative number *i* is returned for the index. Specifically,

if $1 \le i \le n$	then $x_i = c$
if $i = –1$	then $c < x_1$ or $n = 0$
if $–n \le i \le –2$	then $x_{–i–1} < c < x_{–i}$
if $i = –(n + 1)$	then $c > x_n$

The argument `INCX` is useful if a row of a matrix, for example, row number `I` of a matrix `IX`, must be searched. The elements of row `I` are assumed to be in ascending order. Here, set `INCX` equal to the leading dimension of `IX` exactly as specified in the dimension statement in the calling program. With `IX` declared

```
INTEGER IX(LDIX,N)
```

the invocation

`CALL ISRCH (N, IVALUE, IX(I,1), LDIX, INDEX)`

returns an index that will reference a column number of IX.

The routine ISRCH performs a binary search. The routine is an implementation of Algorithm *B* discussed by Knuth (1973, pages 407−411).

Example

This example searches an integer vector sorted in ascending order for the value 653. The problem is discussed by Knuth (1973, pages 407−409).

```
      INTEGER    N
      PARAMETER  (N=16)
C
      INTEGER    INCX, INDEX, NOUT
      INTEGER    IVALUE, IX(N)
      EXTERNAL   ISRCH, UMACH
C
      DATA IX/61, 87, 154, 170, 275, 426, 503, 509, 512, 612, 653, 677,
     &        703, 765, 897, 908/
C
      INCX  = 1
      IVALUE = 653
      CALL ISRCH (N, IVALUE, IX, INCX, INDEX)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) 'INDEX = ', INDEX
      END
```

Output

```
INDEX =   11
```

SSRCH

Search a character vector, sorted in ascending ASCII order, for a given string and return its index.

Usage

`CALL SSRCH (N, STRING, CHX, INCX, INDEX)`

Arguments

N — Length of vector CHY.   (Input)

STRING — Character string to be searched for in CHY.   (Input)

CHX — Vector of length N * INCX containing character strings.   (Input)
CHY is obtained from CHX for I = 1, 2, …, N by CHY(I) = CHX(1 + (I − 1) * INCX). CHY(1), CHY(2), …, CHY(N) must be in ascending ASCII order.

INCX — Displacement between elements of CHX.   (Input)
INCX must be greater than zero.

INDEX — Index of CHY pointing to STRING. (Output)
If INDEX is positive, STRING is found in CHY. If INDEX is negative, STRING is not found in CHY.

INDEX	**Location of STRING**
1 thru N	STRING = CHY(INDEX)
–1	STRING < CHY(1) or N = 0
–N thru –2	CHY(–INDEX – 1) < STRING < CHY(–INDEX)
–(N + 1)	STRING > CHY(N)

Algorithm

Routine SSRCH searches a vector of character strings x (stored in CHX), whose n elements are sorted in ascending ASCII order, for a character string c (stored in STRING). If c is found in x, its index i (stored in INDEX) is returned so that $x_i = c$. Otherwise, a negative number i is returned for the index. Specifically,

if $1 \le i \le n$	then $x_i = c$
if $i = –1$	then $c < x_1$ or $n = 0$
if $–n \le i \le – 2$	then $x_{–i–1} < c < x_{–i}$
if $i = –(n + 1)$	then $c > x_n$

Here, “<“ and “>” are in reference to the ASCII collating sequence. For comparisons made between character strings c and x_i with different lengths, the shorter string is considered as if it were extended on the right with blanks to the length of the longer string. (SSRCH uses FORTRAN intrinsic functions LLT and LGT.)

The argument INCX is useful if a row of a matrix, for example, row number I of a matrix CHX, must be searched. The elements of row I are assumed to be in ascending ASCII order. In this case, set INCX equal to the leading dimension of CHX exactly as specified in the dimension statement in the calling program. With CHX declared

```
CHARACTER * 7 CHX(LDCHX,N)
```

the invocation

```
CALL SSRCH (N, STRING, CHX(I,1), LDCHX, INDEX)
```

returns an index that will reference a column number of CHX.

Routine SSRCH performs a binary search. The routine is an implementation of Algorithm *B* discussed by Knuth (1973, pages 407–411).

Example

This example searches a CHARACTER * 2 vector containing 9 character strings, sorted in ascending ASCII order, for the value ′CC′.

```
      INTEGER    N
      PARAMETER  (N=9)
```

```
C
      INTEGER    INCX, INDEX, NOUT
      CHARACTER  CHX(N)*2, STRING*2
      EXTERNAL   SSRCH, UMACH
C
      DATA CHX/’AA’, ’BB’, ’CC’, ’DD’, ’EE’, ’FF’, ’GG’, ’HH’,
     &     ’II’/
C
      INCX   = 1
      STRING = ’CC’
      CALL SSRCH (N, STRING, CHX, INCX, INDEX)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’INDEX = ’, INDEX
      END
```

Output

```
INDEX =   3
```

ACHAR

Return a character given its ASCII value.

Usage

ACHAR(I)

Arguments

I — Integer ASCII value of the character desired. (Input)
I must be greater than or equal to zero and less than or equal to 127.

ACHAR — CHARACTER * 1 string containing the character in the I-th position of the ASCII collating sequence. (Output)

Algorithm

Routine ACHAR returns the character of the input ASCII value. The input value should be between 0 and 127. If the input value is out of range, the value returned in ACHAR is machine dependent.

Example

This example returns the character of the ASCII value 65.

```
      INTEGER    I, NOUT
      CHARACTER  ACHAR
      EXTERNAL   ACHAR, UMACH
C
      CALL UMACH (2, NOUT)
C                                 Get character for ASCII value
C                                 of 65 (’A’)
      I = 65
      WRITE (NOUT,99999) I, ACHAR(I)
C
```

```
99999 FORMAT (’ For the ASCII value of ’, I2, ’, the character is : ’,
     &       A1)
      END
```

Output

```
For the ASCII value of 65, the character is : A
```

IACHAR

Return the integer ASCII value of a character argument.

Usage

`IACHAR(CH)`

Arguments

CH — Character argument for which the integer ASCII value is desired. (Input)

IACHAR — Integer ASCII value for `CH`. (Output)
The character `CH` is in the `IACHAR`-th position of the ASCII collating sequence.

Algorithm

Routine `IACHAR` returns the ASCII value of the input character.

Example

This example gives the ASCII value of character A.

```
      INTEGER    IACHAR, NOUT
      CHARACTER  CH
      EXTERNAL   IACHAR, UMACH
C
      CALL UMACH (2, NOUT)
C                                  Get ASCII value for the character
C                                  ’A’.
      CH = ’A’
      WRITE (NOUT,99999) CH, IACHAR(CH)
C
99999 FORMAT (’ For the character  ’, A1, ’  the ASCII value is : ’,
     &       I3)
      END
```

Output

```
For the character  A  the ASCII value is :  65
```

ICASE

Return the ASCII value of a character converted to uppercase.

Usage

`ICASE(CH)`

Arguments

CH — Character to be converted. (Input)

ICASE — Integer ASCII value for `CH` without regard to the case of `CH`. (Output) Routine `ICASE` returns the same value as `IACHAR` (page 1156) for all but lowercase letters. For these, it returns the `IACHAR` value for the corresponding uppercase letter.

Algorithm

Routine `ICASE` converts a character to its integer ASCII value. The conversion is case insensitive; that is, it returns the ASCII value of the corresponding uppercase letter for a lowercase letter.

Example

This example shows the case insensitive conversion.

```
      INTEGER    ICASE, NOUT
      CHARACTER  CHR
      EXTERNAL   ICASE, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Get ASCII value for the character
C                                 ’a’.
      CHR = ’a’
      WRITE (NOUT,99999) CHR, ICASE(CHR)
C
99999 FORMAT (’ For the character  ’, A1, ’  the ICASE value is : ’,
     &       I3)
      END
```

Output

```
For the character  a  the ICASE value is :  65
```

IICSR

Compare two character strings using the ASCII collating sequence but without regard to case.

Usage

`IICSR(STR1, STR2)`

Arguments

STR1 — First character string.   (Input)

STR2 — Second character string.   (Input)

IICSR — Comparison indicator.   (Output)
Let USTR1 and USTR2 be the uppercase versions of STR1 and STR2, respectively. The following table indicates the relationship between USTR1 and USTR2 as determined by the ASCII collating sequence.

IICSR	**Meaning**
–1	USTR1 precedes USTR2
0	USTR1 equals USTR2
1	USTR1 follows USTR2

Comments

If the two strings, STR1 and STR2, are of unequal length, the shorter string is considered as if it were extended with blanks to the length of the longer string.

Algorithm

Routine IICSR compares two character strings. It returns –1 if the first string is less than the second string, 0 if they are equal, and 1 if the first string is greater than the second string. The comparison is case insensitive.

Example

This example shows different cases on comparing two strings.

```
      INTEGER    IICSR, NOUT
      CHARACTER  STR1*6, STR2*6
      EXTERNAL   IICSR, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Compare String1 and String2
C                                 String1 is ’bigger’ than String2
      STR1 = ’ABc 1’
      STR2 = ’ ’
      WRITE (NOUT,99999) STR1, STR2, IICSR(STR1,STR2)
C
C                                 String1 is ’equal’ to String2
      STR1 = ’AbC’
      STR2 = ’ABc’
      WRITE (NOUT,99999) STR1, STR2, IICSR(STR1,STR2)
C
C                                 String1 is ’smaller’ than String2
      STR1 = ’ABc’
      STR2 = ’aBC 1’
      WRITE (NOUT,99999) STR1, STR2, IICSR(STR1,STR2)
C
99999 FORMAT (’ For String1 = ’, A6, ’and String2 = ’, A6,
     &       ’ IICSR = ’, I2, /)
      END
```

Output

```
For String1 = ABc 1 and String2 =        IICSR =  1

For String1 = AbC   and String2 = ABc    IICSR =  0

For String1 = ABc   and String2 = aBC 1  IICSR = -1
```

IIDEX

Determine the position in a string at which a given character sequence begins without regard to case.

Usage

IIDEX(CHRSTR, KEY)

Arguments

CHRSTR — Character string to be searched. (Input)

KEY — Character string that contains the key sequence. (Input)

IIDEX — Position in CHRSTR where KEY begins. (Output)
If KEY occurs more than once in CHRSTR, the starting position of the first occurrence is returned. If KEY does not occur in CHRSTR, then IIDEX returns a zero.

Comments

If the length of KEY is greater than the length CHRSTR, IIDEX returns a zero.

Algorithm

Routine IIDEX searches for a key string in a given string and returns the index of the starting element at which the key character string begins. It returns 0 if there is no match. The comparison is case insensitive. For a case-sensitive version, use the FORTRAN 77 intrinsic function INDEX.

Example

This example locates a key string.

```
      INTEGER    IIDEX, NOUT
      CHARACTER  KEY*5, STRING*10
      EXTERNAL   IIDEX, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Locate KEY in STRING
      STRING = ’a1b2c3d4e5’
      KEY    = ’C3d4E’
      WRITE (NOUT,99999) STRING, KEY, IIDEX(STRING,KEY)
C
      KEY = ’F’
      WRITE (NOUT,99999) STRING, KEY, IIDEX(STRING,KEY)
```

```
C
99999 FORMAT (’ For STRING = ’, A10, ’ and KEY = ’, A5, ’ IIDEX = ’, I2,
     &       /)
      END
```

Output

```
For STRING = a1b2c3d4e5 and KEY = C3d4E IIDEX =  5

For STRING = a1b2c3d4e5 and KEY = F     IIDEX =  0
```

CVTSI

Convert a character string containing an integer number into the corresponding integer form.

Usage

CALL CVTSI (STRING, NUMBER)

Arguments

STRING — Character string containing an integer number. (Input)

NUMBER — The integer equivalent of STRING. (Output)

Algorithm

Routine CVTSI converts a character string containing an integer to an INTEGER variable. Leading and trailing blanks in the string are ignored. If the string contains something other than an integer, a terminal error is issued. If the string contains an integer larger than can be represented by an INTEGER variable as determined from routine IMACH (see page 1201 in the Reference Material), a terminal error is issued.

Example

The string “12345” is converted to an INTEGER variable.

```
      INTEGER    NOUT, NUMBER
      CHARACTER  STRING*10
      EXTERNAL   CVTSI, UMACH
C
      DATA STRING/’12345’/
C
      CALL CVTSI (STRING, NUMBER)
C
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’NUMBER = ’, NUMBER
      END
```

Output

```
NUMBER =   12345
```

CPSEC

Return CPU time used in seconds.

Usage

`CPSEC ()`

Arguments

CPSEC — CPU time used (in seconds) since first call to `CPSEC`. (Output)

Comments

1. The first call to `CPSEC` returns 0.0.
2. The accuracy of this routine depends on the hardware and the operating system. On some systems, identical runs can produce timings differing by more than 10 percent.

TIMDY

Get time of day.

Usage

`CALL TIMDY (IHOUR, MINUTE, ISEC)`

Arguments

IHOUR — Hour of the day. (Output)
`IHOUR` is between 0 and 23 inclusive.

MINUTE — Minute within the hour. (Output)
`MINUTE` is between 0 and 59 inclusive.

ISEC — Second within the minute. (Output)
`ISEC` is between 0 and 59 inclusive.

Algorithm

Routine `TIMDY` is used to retrieve the time of day.

Example

The following example uses `TIMDY` to return the current time. Obviously, the output is dependent upon the time at which the program is run.

```
      INTEGER    IHOUR, IMIN, ISEC, NOUT
      EXTERNAL   TIMDY, UMACH
C
      CALL TIMDY (IHOUR, IMIN, ISEC)
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’Hour:Minute:Second = ’, IHOUR, ’:’, IMIN,
     &              ’:’, ISEC
      IF (IHOUR .EQ. 0) THEN
         WRITE (NOUT,*) ’The time is ’, IMIN, ’ minute(s), ’, ISEC,
     &                 ’ second(s) past midnight.’
      ELSE IF (IHOUR .LT. 12) THEN
         WRITE (NOUT,*) ’The time is ’, IMIN, ’ minute(s), ’, ISEC,
     &                 ’ second(s) past ’, IHOUR, ’ am.’
      ELSE IF (IHOUR .EQ. 12) THEN
         WRITE (NOUT,*) ’The time is ’, IMIN, ’ minute(s), ’, ISEC,
     &                 ’ second(s) past noon.’
      ELSE
         WRITE (NOUT,*) ’The time is ’, IMIN, ’ minute(s), ’, ISEC,
     &                 ’ second(s) past ’, IHOUR-12, ’ pm.’
      END IF
      END
```

Output

```
Hour:Minute:Second =   16:  52:  29
The time is   52 minute(s),   29 second(s) past   4 pm.
```

TDATE

Get today’s date.

Usage

CALL TDATE (IDAY, MONTH, IYEAR)

Arguments

IDAY — Day of the month.   (Output)
IDAY is between 1 and 31 inclusive.

MONTH — Month of the year.   (Output)
MONTH is between 1 and 12 inclusive.

IYEAR — Year.   (Output)
For example, IYEAR = 1985.

Algorithm

Routine TDATE is used to retrieve today’s date. Obviously, the output is dependent upon the date the program is run.

Example

The following example uses TDATE to return today’s date.

```
      INTEGER    IDAY, IYEAR, MONTH, NOUT
      EXTERNAL   TDATE, UMACH
C
      CALL TDATE (IDAY, MONTH, IYEAR)
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’Day-Month-Year = ’, IDAY, ’-’, MONTH,
     &               '-', IYEAR
      END
```

Output

```
Day-Month-Year =   2-  4-  1991
```

NDAYS

Compute the number of days from January 1, 1900, to the given date.

Usage

```
NDAYS(IDAY, MONTH, IYEAR)
```

Arguments

IDAY — Day of the input date. (Input)

MONTH — Month of the input date. (Input)

IYEAR — Year of the input date. (Input)
1950 would correspond to the year 1950 A.D. and 50 would correspond to year 50 A.D.

NDAYS — Function value. (Output)
If NDAYS is negative, it indicates the number of days prior to January 1, 1900.

Comments

1. Informational error

Type	Code	
1	1	The Julian calendar, the first modern calendar, went into use in 45 B.C. No calendar prior to 45 B.C. was as universally used nor as accurate as the Julian. Therefore, it is assumed that the Julian calendar was in use prior to 45 B.C.

2. The number of days from one date to a second date can be computed by two references to NDAYS and then calculating the difference.

3. The beginning of the Gregorian calendar was the first day after October 4, 1582, which became October 15, 1582. Prior to that, the Julian calendar was in use. NDAYS makes the proper adjustment for the change in calendars.

Algorithm

Function NDAYS returns the number of days from January 1, 1900, to the given date. The function NDAYS returns negative values for days prior to January 1, 1900. A negative IYEAR can be used to specify B.C. Input dates in year 0 and for October 5, 1582, through October 14, 1582, inclusive, do not exist; consequently, in these cases, NDAYS issues a terminal error.

Example

The following example uses NDAYS to compute the number of days from January 15, 1986, to February 28, 1986:

```
      INTEGER    IDAY, IYEAR, MONTH, NDAY0, NDAY1, NDAYS, NOUT
      EXTERNAL   NDAYS, UMACH
C
      IDAY  = 15
      MONTH = 1
      IYEAR = 1986
      NDAY0 = NDAYS(IDAY,MONTH,IYEAR)
      IDAY  = 28
      MONTH = 2
      IYEAR = 1986
      NDAY1 = NDAYS(IDAY,MONTH,IYEAR)
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) 'Number of days = ', NDAY1 - NDAY0
      END
```

Output

```
Number of days =   44
```

NDYIN

Give the date corresponding to the number of days since January 1, 1900.

Usage

```
CALL NDYIN (NDAYS, IDAY, MONTH, IYEAR)
```

Arguments

NDAYS — Number of days since January 1, 1900. (Input)

IDAY — Day of the input date. (Output)

MONTH — Month of the input date. (Output)

IYEAR — Year of the input date. (Output)
1950 would correspond to the year 195 A.D. and −50 would correspond to year 50 B.C.

Comments

The beginning of the Gregorian calendar was the first day after October 4, 1582, which became October 15, 1582. Prior to that, the Julian calendar was in use. Routine NDYIN makes the proper adjustment for the change in calendars.

Algorithm

Routine NDYIN computes the date corresponding to the number of days since January 1, 1900. For an input value of NDAYS that is negative, the date computed is prior to January 1, 1900. The routine `NDYIN` is the inverse of `NDAYS` (page 1163).

Example

The following example uses `NDYIN` to compute the date for the 100th day of 1986. This is accomplished by first using `NDAYS` (page 1163) to get the “day number” for December 31, 1985.

```
      INTEGER    IDAY, IYEAR, MONTH, NDAYO, NDAYS, NOUT
      EXTERNAL   NDAYS, NDYIN, UMACH
C
      NDAY0 = NDAYS(31,12,1985)
      CALL NDYIN (NDAY0+100, IDAY, MONTH, IYEAR)
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) ’Day 100 of 1986 is (day-month-year) ’, IDAY,
     &              ’-’, MONTH, ’-’, IYEAR
      END
```

Output

```
Day 100 of 1986 is (day-month-year)   10-  4-  1986
```

IDYWK

Compute the day of the week for a given date.

Usage

`IDYWK(IDAY, MONTH, IYEAR)`

Arguments

IDAY — Day of the input date. (Input)

MONTH — Month of the input date. (Input)

IYEAR — Year of the input date. (Input)
1950 would correspond to the year 1950 A.D. and 50 would correspond to year 50 A.D.

IDYWK — Function value. (Output)
The value of `IDYWK` ranges from 1 to 7, where 1 corresponds to Sunday and 7 corresponds to Saturday.

Comments

1. Informational error
   Type Code
   1 1 The Julian calendar, the first modern calendar, went into use in 45 B.C. No calendar prior to 45 B.C. was as universally used nor as accurate as the Julian. Therefore, it is assumed that the Julian calendar was in use prior to 45 B.C.

2. The beginning of the Gregorian calendar was the first day after October 4, 1582, which became October 15, 1582. Prior to that, the Julian calendar was in use. Function IDYWK makes the proper adjustment for the change in calendars.

Algorithm

Function IDYWK returns an integer code that specifies the day of week for a given date. Sunday corresponds to 1, Monday corresponds to 2, and so forth.

A negative IYEAR can be used to specify B.C. Input dates in year 0 and for October 5, 1582, through October 14, 1582, inclusive, do not exist; consequently, in these cases, IDYWK issues a terminal error.

Example

The following example uses IDYWK to return the day of the week for February 24, 1963.

```
      INTEGER    IDAY, IDYWK, IYEAR, MONTH, NOUT
      EXTERNAL   IDYWK, UMACH
C
      IDAY  = 24
      MONTH = 2
      IYEAR = 1963
      CALL UMACH (2, NOUT)
      WRITE (NOUT,*) 'IDYWK (index for day of week) = ',
     &               IDYWK(IDAY,MONTH,IYEAR)
      END
```

Output

```
IDYWK (index for day of week) =   1
```

VERML

Obtain IMSL MATH/LIBRARY-related version, system and serial numbers.

Usage

`VERML(ISELCT)`

Arguments

ISELCT — Option for the information to retrieve. (Input)

ISELCT	**VERML**
1	IMSL MATH/LIBRARY version number
2	Operating system (and version number) for which the library was produced.
3	Fortran compiler (and version number) for which the library was produced.
4	IMSL MATH/LIBRARY serial number

VERML — `CHARACTER` string containing information. (Output)

Example

In this example, we print all of the information returned by `VERML` on a particular machine. The output is omitted because the results are system dependent.

```
      INTEGER    ISELCT, NOUT
      CHARACTER  STRING(4)*50, TEMP*32, VERML*32
      EXTERNAL   UMACH, VERML
C
      STRING(1) = '('' IMSL MATH/LIBRARY Version Number:  '',
     &              A)'
      STRING(2) = '('' Operating System ID Number:  '', A)'
      STRING(3) = '('' Fortran Compiler Version Number:  '', A)'
      STRING(4) = '('' IMSL MATH/LIBRARY Serial Number:  '', A)'
C                                  Print the versions and numbers.
      CALL UMACH (2, NOUT)
      DO 10  ISELCT=1, 4
         TEMP = VERML(ISELCT)
         WRITE (NOUT,STRING(ISELCT)) TEMP
   10 CONTINUE
      END
```

Output

```
IMSL MATH/LIBRARY Version Number:  IMSL MATH/LIBRARY Version 2.0
Operating System ID Number:  SunOS 4.1.1
Fortran Compiler Version Number:  f77 Sun FORTRAN 1.3.1
IMSL MATH/LIBRARY Serial Number:  123456
```

RNGET

Retrieve the current value of the seed used in the IMSL random number generators.

Usage

`CALL RNGET (ISEED)`

Arguments

ISEED — The seed of the random number generator. (Output)
`ISEED` is in the range (1, 2147483646).

Algorithm

Routine `RNGET` retrieves the current value of the “seed” used in the IMSL random number generators. A reason for doing this would be to restart a simulation, using `RNSET` to reset the seed.

Example

The following FORTRAN statements illustrate the use of `RNGET`:

```
      INTEGER ISEED
C                         Call RNSET to initialize the seed.
      CALL RNSET(123457)
C                         Do some simulations.
         ...
         ...
      CALL RNGET(ISEED)
C                         Save ISEED.  If the simulation is to be continued
C                         in a different program, ISEED should be output,
C                         possibly to a file.
         ...
         ...
C                         When the simulations begun above are to be
C                         restarted, restore ISEED to the value obtained
C                         above and use as input to RNSET.
      CALL RNSET(ISEED)
C                         Now continue the simulations.
         ...
         ...
```

RNSET

Initialize a random seed for use in the IMSL random number generators.

Usage

`CALL RNSET (ISEED)`

Arguments

ISEED — The seed of the random number generator.   (Input)
`ISEED` must be in the range (0, 2147483646). If `ISEED` is zero, a value is computed using the system clock; and, hence, the results of programs using the IMSL random number generators will be different at different times.

Algorithm

Routine `RNSET` is used to initialize the seed used in the IMSL random number generators. If the seed is not initialized prior to invocation of any of the routines for random number generation by calling `RNSET`, the seed is initialized via the system clock. The seed can be reinitialized to a clock-dependent value by calling `RNSET` with `ISEED` set to 0.

The effect of `RNSET` is to set some values in a FORTRAN `COMMON` block that is used by the random number generators.

A common use of `RNSET` is in conjunction with `RNGET` (page 1167) to restart a simulation.

Example

The following FORTRAN statements illustrate the use of `RNSET`:

```
      INTEGER ISEED
```

```
C                          Call RNSET to initialize the seed via the
C                          system clock.
        CALL RNSET(0)
C                          Do some simulations.
            ...
            ...
C                          Obtain the current value of the seed.
        CALL RNGET(ISEED)
C                          If the simulation is to be continued in a
C                          different program, ISEED should be output,
C                          possibly to a file.
             ...
             ...
C                          When the simulations begun above are to be
C                          restarted, restore ISEED to the value
C                          obtained above, and use as input to RNSET.
        CALL RNSET(ISEED)
C                          Now continue the simulations.
             ...
             ...
```

RNOPT

Select the uniform (0, 1) multiplicative congruential pseudorandom number generator.

Usage

CALL RNOPT (IOPT)

Arguments

IOPT — Indicator of the generator. (Input)
The random number generator is either a multiplicative congruential generator with modulus $2^{31} - 1$ or a GFSR generator. IOPT is used to choose the multiplier and whether or not shuffling is done, or else to choose the GFSR method.

IOPT	**Generator**
1	The multiplier 16807 is used.
2	The multiplier 16807 is used with shuffling.
3	The multiplier 397204094 is used.
4	The multiplier 397204094 is used with shuffling.
5	The multiplier 950706376 is used.
6	The multiplier 950706376 is used with shuffling.
7	GFSR, with the recursion $X_t = X_{t-1563} \oplus X_{t-96}$ is used.

Algorithm

The IMSL uniform pseudorandom number generators use a multiplicative congruential method, with or without shuffling or else a GFSR method. Routine RNOPT determines which method is used; and in the case of a multiplicative congruential method, it determines the value of the multiplier and whether or not to use shuffling. The description of RNUN (page 1171) may provide some guidance in the choice of the form of the generator. If no selection is made explicitly, the generators use the multiplier 16807 without shuffling. This form of the generator has been in use for some time (see Lewis, Goodman, and Miller, 1969). This is the generator formerly known as GGUBS in the IMSL Library. It is the "minimal standard generator" discussed by Park and Miller (1988).

Example

The FORTRAN statement

```
CALL RNOPT(1)
```

would select the simple multiplicative congruential generator with multiplier 16807. Since this is the same as the default, this statement would have no effect unless RNOPT had previously been called in the same program to select a different generator.

RNUNF/DRNUNF (Single/Double precision)

Generate a pseudorandom number from a uniform (0, 1) distribution.

Usage

```
RNUNF()
```

Arguments

RNUNF — Function value, a random uniform (0, 1) deviate. (Output)

Comments

1. Routine RNSET (page 1168) can be used to initialize the seed of the random number generator. The routine RNOPT (page 1169) can be used to select the form of the generator.
2. This function has a side effect: it changes the value of the seed, which is passed through a common block.

Algorithm

Routine RNUNF is the function form of RNUN (page 1171). The routine RNUNF generates pseudorandom numbers from a uniform (0, 1) distribution. The algorithm used is determined by RNOPT (page 1169). The values returned by RNUNF are positive and less than 1.0.

If several uniform deviates are needed, it may be more efficient to obtain them all at once by a call to RNUN rather than by several references to RNUNF.

Example

In this example, RNUNF is used to generate five pseudorandom uniform numbers. Since RNOPT (page 1169) is not called, the generator used is a simple multiplicative congruential one with a multiplier of 16807.

```
      INTEGER    I, ISEED, NOUT
      REAL       R(5), RNUNF
      EXTERNAL   RNSET, RNUNF, UMACH
C
      CALL UMACH (2, NOUT)
      ISEED = 123457
      CALL RNSET (ISEED)
      DO 10  I=1, 5
         R(I) = RNUNF()
   10 CONTINUE
      WRITE (NOUT,99999) R
99999 FORMAT (’      Uniform random deviates: ’, 5F8.4)
      END
```

Output

```
Uniform random deviates:   0.9662  0.2607  0.7663  0.5693  0.8448
```

RNUN/DRNUN (Single/Double precision)

Generate pseudorandom numbers from a uniform (0, 1) distribution.

Usage

```
CALL RNUN (NR, R)
```

Arguments

NR — Number of random numbers to generate.   (Input)

R — Vector of length NR containing the random uniform (0, 1) deviates. (Output)

Comments

The routine RNSET (page 1168) can be used to initialize the seed of the random number generator. The routine RNOPT (page 1169) can be used to select the form of the generator.

Algorithm

Routine RNUN generates pseudorandom numbers from a uniform (0,1) distribution using either a multiplicative congruential method or a generalized feedback shift register (GFSR) method. The form of the multiplicative congruential generator is

$$x_i \equiv cx_{i-1} \bmod \left(2^{31} - 1\right)$$

Each x_i is then scaled into the unit interval (0,1). The possible values for c in the IMSL generators are 16807, 397204094, and 950706376. The selection is made by the routine RNOPT (page 1169). The choice of 16807 will result in the fastest execution time. If no selection is made explicitly, the routines use the multiplier 16807.

The user can also select a shuffled version of the multiplicative congruential generators. In this scheme, a table is filled with the first 128 uniform (0,1) numbers resulting from the simple multiplicative congruential generator. Then, for each x_i from the simple generator, the low-order bits of x_i are used to select a random integer, j, from 1 to 128. The j-th entry in the table is then delivered as the random number; and x_i, after being scaled into the unit interval, is inserted into the j-th position in the table.

The GFSR method is based on the recursion $X_t = X_{t-1\,563} \oplus X_{t-96}$. This generator, which is different from earlier GFSR generators, was proposed by Fushimi (1990), who discusses the theory behind the generator and reports on several empirical tests of it. The values returned in R by RNUN are positive and less than 1.0. Values in R may be smaller than the smallest relative spacing, however. Hence, it may be the case that some value R(i) is such that 1.0 – R(i) = 1.0.

Deviates from the distribution with uniform density over the interval (A, B) can be obtained by scaling the output from RNUN. The following statements (in single precision) would yield random deviates from a uniform (A, B) distribution:

```
       CALL RNUN (NR, R)
       CALL SSCAL (NR, B-A, R, 1)
       CALL SADD (NR, A, R, 1)
```

Example

In this example, RNUN is used to generate five pseudorandom uniform numbers. Since RNOPT (page 1169) is not called, the generator used is a simple multiplicative congruential one with a multiplier of 16807.

```
      INTEGER    ISEED, NOUT, NR
      REAL       R(5)
      EXTERNAL   RNSET, RNUN, UMACH
C
      CALL UMACH (2, NOUT)
      NR    = 5
      ISEED = 123457
      CALL RNSET (ISEED)
      CALL RNUN (NR, R)
      WRITE (NOUT,99999) R
99999 FORMAT (’      Uniform random deviates: ’, 5F8.4)
      END
```

Output

```
Uniform random deviates:    .9662   .2607   .7663   .5693   .8448
```

IUMAG

This routine handles MATH/LIBRARY and STAT/LIBRARY type INTEGER options.

Usage

```
CALL IUMAG (PRODNM, ICHP, IACT, NUMOPT, IOPTS, IVALS)
```

Arguments

PRODNM — Product name. Use either “MATH” or “STAT.” (Input)

ICHP — Chapter number of the routine that uses the options. (Input)

IACT — 1 if user desires to “get” or read options, or 2 if user desires to “put” or write options. (Input)

NUMOPT — Size of IOPTS. (Input)

IOPTS — Integer array of size NUMOPT containing the option numbers to “get” or “put.” (Input)

IVALS — Integer array containing the option values. These values are arrays corresponding to the individual options in IOPTS in sequential order. The size of IVALS is the sum of the sizes of the individual options. (Input/Output)

Comments

1. Users can normally avoid reading about options when first using a routine that calls IUMAG.

2. Let I be any value between 1 and NUMOPT. A negative value of IOPTS(I) refers to option number −IOPTS(I) but with a different effect: For a “get” operation, the default values are returned in IVALS. For a “put” operation, the default values replace the current values. In the case of a “put,” entries of IVALS are not allocated by the user and are not used by IUMAG.

3. Both positive and negative values of IOPTS can be used.

4. INTEGER Options

   1 If the value is positive, print the next activity for any library routine that uses the Options Manager codes IUMAG, SUMAG, or DUMAG. Each printing step decrements the value if it is positive. Default value is 0.

   2 If the value is 2, perform error checking in IUMAG (page 1173), SUMAG (page 1175), and DUMAG (page 1178) such as the verifying of valid option numbers and the validity of input data. If the value is 1, do not perform error checking. Default value is 2.

3 This value is used for testing the installation of IUMAG by other IMSL software. Default value is 3.

Algorithm

The Options Manager routine IUMAG reads or writes INTEGER data for some MATH/LIBRARY and STAT/LIBRARY codes. See Atchison and Hanson (1991) for more complete details.

There are MATH/LIBRARY routines in Chapters 1, 2, and 5 that now use IUMAG to communicate optional data from the user.

Example

The number of iterations allowed for the constrained least squares solver LCLSQ that calls L2LSQ is changed from the default value of max(*nra*, *nca*) to the value 6. The default value is restored after the call to LCLSQ. This change has no effect on the solution. It is used only for illustration. The first two arguments required for the call to IUMAG are defined by the product name, “MATH,” and chapter number, 1, where LCLSQ is documented. The argument IACT denotes a write or “put” operation. There is one option to change so NUMOPT has the value 1. The arguments for the option number, 14, and the new value, 6, are defined by reading the documentation for LCLSQ.

```
C
C     Solve the following in the least squares sense:
C           3x1 + 2x2 +  x3 = 3.3
C           4x1 + 2x2 +  x3 = 2.3
C           2x1 + 2x2 +  x3 = 1.3
C            x1 +  x2 +  x3 = 1.0
C
C     Subject to:  x1 + x2 + x3 <= 1
C                  0 <= x1 <= .5
C                  0 <= x2 <= .5
C                  0 <= x3 <= .5
C
C ----------------------------------------------------------------------
C                                 Declaration of variables
C
      INTEGER    ICHP, IPUT, LDA, LDC, MCON, NCA, NEWMAX, NRA, NUMOPT
      PARAMETER  (ICHP=1, IPUT=2, MCON=1, NCA=3, NEWMAX=14, NRA=4,
     &           NUMOPT=1, LDA=NRA, LDC=MCON)
C
      INTEGER    IOPT(1), IRTYPE(MCON), IVAL(1), NOUT
      REAL       A(LDA,NCA), B(NRA), BC(MCON), C(LDC,NCA), RES(NRA),
     &           RESNRM, SNRM2, XLB(NCA), XSOL(NCA), XUB(NCA)
      EXTERNAL   IUMAG, LCLSQ, SNRM2, UMACH
C                                 Data initialization
C
      DATA A/3.0E0, 4.0E0, 2.0E0, 1.0E0, 2.0E0, 2.0E0, 2.0E0, 1.0E0,
     &     1.0E0, 1.0E0, 1.0E0, 1.0E0/, B/3.3E0, 2.3E0, 1.3E0, 1.0E0/,
     &     C/3*1.0E0/, BC/1.0E0/, IRTYPE/1/, XLB/3*0.0E0/, XUB/3*.5E0/
C ----------------------------------------------------------------------
C
C                                 Reset the maximum number of
```

```
C                                 iterations to use in the solver.
C                                 The value 14 is the option number.
C                                 The value 6 is the new maximum.
      IOPT(1) = NEWMAX
      IVAL(1) = 6
      CALL IUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, IVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
C                                 Solve the bounded, constrained
C                                 least squares problem.
C
      CALL LCLSQ (NRA, NCA, MCON, A, LDA, B, C, LDC, BC, BC, IRTYPE,
     &            XLB, XUB, XSOL, RES)

C                                 Compute the 2-norm of the residuals.
      RESNRM = SNRM2(NRA,RES,1)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) XSOL, RES, RESNRM
C                                 -------------------------------------
C                                 ---------------------------------
C                                 Reset the maximum number of
C                                 iterations to its default value.
C                                 This is not required but is
C                                 recommended programming practice.
      IOPT(1) = -IOPT(1)
      CALL IUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, IVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
99999 FORMAT ('  The solution is ', 3F9.4, //, '  The residuals ',
     &        'evaluated at the solution are ', /, 18X, 4F9.4, //,
     &        '  The norm of the residual vector is ', F8.4)
C
      END
```

Output

```
The solution is    0.5000   0.3000   0.2000

The residuals evaluated at the solution are
-1.0000   0.5000   0.5000   0.0000

The norm of the residual vector is   1.2247
```

SUMAG

This routine handles MATH/LIBRARY and STAT/LIBRARY type SINGLE PRECISION options.

Usage

```
CALL SUMAG (PRODNM, ICHP, IACT, NUMOPT, IOPTS, SVALS)
```

Arguments

PRODNM — Product name. Use either "MATH" or "STAT."   (Input)

ICHP — Chapter number of the routine that uses the options.   (Input)

IACT — 1 if user desires to "get" or read options, or 2 if user desires to "put" or write options.   (Input)

NUMOPT — Size of IOPTS.   (Input)

IOPTS — Integer array of size NUMOPT containing the option numbers to "get" or "put."   (Input)

SVALS — Real array containing the option values. These values are arrays corresponding to the individual options in IOPTS in sequential order. The size of SVALS is the sum of the sizes of the individual options.   (Input/Output)

Comments

1. Users can normally avoid reading about options when first using a routine that calls SUMAG.

2. Let I be any value between 1 and NUMOPT. A negative value of IOPTS(I) refers to option number −IOPTS(I) but with a different effect: For a "get" operation, the default values are returned in SVALS. For a "put" operation, the default values replace the current values. In the case of a "put," entries of SVALS are not allocated by the user and are not used by SUMAG.

3. Both positive and negative values of IOPTS can be used.

4. Floating Point Options

   1 This value is used for testing the installation of SUMAG by other IMSL software. Default value is 3.0E0.

Algorithm

The Options Manager routine SUMAG reads or writes REAL data for some MATH/LIBRARY and STAT/LIBRARY codes. See Atchison and Hanson (1991) for more complete details. There are MATH/LIBRARY routines in Chapters 1 and 5 that now use SUMAG to communicate optional data from the user.

Example

The rank determination tolerance for the constrained least squares solver LCLSQ that calls L2LSQ is changed from the default value of SQRT(AMACH(4)) to the value 0.01. The default value is restored after the call to LCLSQ. This change has no effect on the solution. It is used only for illustration. The first two arguments required for the call to SUMAG are defined by the product name, "MATH," and chapter number, 1, where LCLSQ is documented. The argument IACT denotes a write or “put” operation. There is one option to change so NUMOPT has the value 1. The arguments for the option number, 2, and the new value, 0.01E+0, are defined by reading the documentation for LCLSQ.

```
C
C     Solve the following in the least squares sense:
C           3x1 + 2x2 +  x3 = 3.3
C           4x1 + 2x2 +  x3 = 2.3
C           2x1 + 2x2 +  x3 = 1.3
C            x1 +  x2 +  x3 = 1.0
C
C     Subject to:  x1 + x2 + x3 <= 1
C                  0 <= x1 <= .5
C                  0 <= x2 <= .5
C                  0 <= x3 <= .5
C
C ----------------------------------------------------------------------
C                                 Declaration of variables
C
      INTEGER    ICHP, IPUT, LDA, LDC, MCON, NCA, NEWTOL, NRA, NUMOPT
      PARAMETER  (ICHP=1, IPUT=2, MCON=1, NCA=3, NEWTOL=2, NRA=4,
     &           NUMOPT=1, LDA=NRA, LDC=MCON)
C
      INTEGER    IOPT(1), IRTYPE(MCON), NOUT
      REAL       A(LDA,NCA), B(NRA), BC(MCON), C(LDC,NCA), RES(NRA),
     &           RESNRM, SNRM2, SVAL(1), XLB(NCA), XSOL(NCA), XUB(NCA)
      EXTERNAL   LCLSQ, SNRM2, SUMAG, UMACH
C                                 Data initialization
C
      DATA A/3.0E0, 4.0E0, 2.0E0, 1.0E0, 2.0E0, 2.0E0, 2.0E0, 1.0E0,
     &     1.0E0, 1.0E0, 1.0E0, 1.0E0/, B/3.3E0, 2.3E0, 1.3E0, 1.0E0/,
     &     C/3*1.0E0/, BC/1.0E0/, IRTYPE/1/, XLB/3*0.0E0/, XUB/3*.5E0/
C ----------------------------------------------------------------------
C
C                                 Reset the rank determination
C                                 tolerance used in the solver.
C                                 The value 2 is the option number.
C                                 The value 0.01 is the new tolerance.
C
      IOPT(1) = NEWTOL
      SVAL(1) = 0.01E+0
      CALL SUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, SVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
C                                 Solve the bounded, constrained
C                                 least squares problem.
C
      CALL LCLSQ (NRA, NCA, MCON, A, LDA, B, C, LDC, BC, BC, IRTYPE,
     &            XLB, XUB, XSOL, RES)
C                                 Compute the 2-norm of the residuals.
      RESNRM = SNRM2(NRA,RES,1)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) XSOL, RES, RESNRM
C                                 -------------------------------------
C                                 ---------------------------------
C                                 Reset the rank determination
C                                 tolerance to its default value.
```

```
C                                 This is not required but is
C                                 recommended programming practice.
      IOPT(1) = -IOPT(1)
      CALL SUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, SVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
99999 FORMAT ('  The solution is ', 3F9.4, //, '  The residuals ',
     &       'evaluated at the solution are ', /, 18X, 4F9.4, //,
     &       '  The norm of the residual vector is ', F8.4)
C
      END
```

Output

```
The solution is    0.5000   0.3000   0.2000

The residuals evaluated at the solution are
-1.0000   0.5000   0.5000   0.0000

The norm of the residual vector is   1.2247
```

DUMAG

This routine handles MATH/LIBRARY and STAT/LIBRARY type DOUBLE PRECISION options.

Usage

CALL DUMAG (PRODNM, ICHP, IACT, NUMOPT, IOPTS, DVALS)

Arguments

PRODNM — Product name. Use either “MATH” or “STAT.”   (Input)

ICHP — Chapter number of the routine that uses the options.   (Input)

IACT — 1 if user desires to “get” or read options, or 2 if user desires to “put” or write options.   (Input)

NUMOPT — Size of IOPTS.   (Input)

IOPTS — Integer array of size NUMOPT containing the option numbers to “get” or “put.”   (Input)

DVALS — Double precision array containing the option values. These values are arrays corresponding to the individual options in IOPTS in sequential order. The size of DVALS is the sum of the sizes of the individual options.   (Input/ Output)

Comments

1. Users can normally avoid reading about options when first using a routine that calls DUMAG.

2. Let I be any value between 1 and NUMOPT. A negative value of IOPTS(I) refers to option number −IOPTS(I) but with a different effect: For a “get” operation, the default values are returned in DVALS. For a “put” operation, the default values replace the current values. In the case of a “put,” entries of DVALS are not allocated by the user and are not used by DUMAG.

3. Both positive and negative values of IOPTS can be used.

4. Floating Point Options

   1 This value is used for testing the installation of DUMAG by other IMSL software. Default value is 3.0D0.

Algorithm

The Options Manager routine DUMAG reads or writes DOUBLE PRECISION data for some MATH/LIBRARY and STAT/LIBRARY codes. See Atchison and Hanson (1991) for more complete details. There are MATH/LIBRARY routines in Chapters 1 and 5 that now use DUMAG to communicate optional data from the user.

Example

The rank determination tolerance for the constrained least squares solver DLCLSQ that calls DL2LSQ is changed from the default value of SQRT(DMACH(4)) to the value 0.01. The default value is restored after the call to DLCLSQ. This change has no effect on the solution. It is used only for illustration. The first two arguments required for the call to DUMAG are defined by the product name, “MATH,” and chapter number, 1, where DLCLSQ is documented. The argument IACT denotes a write or “put” operation. There is one option to change so NUMOPT has the value 1. The arguments for the option number, 2, and the new value, 0.01D+0, are defined by reading the documentation for DLCLSQ.

```
C
C     Solve the following in the least squares sense:
C           3x1 + 2x2 +  x3 = 3.3
C           4x1 + 2x2 +  x3 = 2.3
C           2x1 + 2x2 +  x3 = 1.3
C            x1 +  x2 +  x3 = 1.0
C
C     Subject to:  x1 + x2 + x3 <= 1
C                  0 <= x1 <= .5
C                  0 <= x2 <= .5
C                  0 <= x3 <= .5
C
C ----------------------------------------------------------------------
C                                  Declaration of variables
C
      INTEGER    ICHP, IPUT, LDA, LDC, MCON, NCA, NEWTOL, NRA, NUMOPT
      PARAMETER  (ICHP=1, IPUT=2, MCON=1, NCA=3, NEWTOL=2, NRA=4,
     &           NUMOPT=1, LDA=NRA, LDC=MCON)
C
      INTEGER    IOPT(1), IRTYPE(MCON), NOUT
      DOUBLE PRECISION A(LDA,NCA), B(NRA), BC(MCON), C(LDC,NCA),
     &           DNRM2, DVAL(1), RES(NRA), RESNRM, XLB(NCA),
     &           XSOL(NCA), XUB(NCA)
      EXTERNAL   DLCLSQ, DNRM2, DUMAG, UMACH
C                                 Data initialization
C
      DATA A/3.0D0, 4.0D0, 2.0D0, 1.0D0, 2.0D0, 2.0D0, 2.0D0, 1.0D0,
     &     1.0D0, 1.0D0, 1.0D0, 1.0D0/, B/3.3D0, 2.3D0, 1.3D0, 1.0D0/,
     &     C/3*1.0D0/, BC/1.0D0/, IRTYPE/1/, XLB/3*0.0D0/, XUB/3*.5D0/
C ----------------------------------------------------------------------
C
C                                 Reset the rank determination
C                                 tolerance used in the solver.
C                                 The value 2 is the option number.
C                                 The value 0.01 is the new tolerance.
C
      IOPT(1) = NEWTOL
      DVAL(1) = 0.01D+0
      CALL DUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, DVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
C                                 Solve the bounded, constrained
C                                 least squares problem.
C
      CALL DLCLSQ (NRA, NCA, MCON, A, LDA, B, C, LDC, BC, BC, IRTYPE,
     &             XLB, XUB, XSOL, RES)
C                                 Compute the 2-norm of the residuals.
      RESNRM = DNRM2(NRA,RES,1)
C                                 Print results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) XSOL, RES, RESNRM
C                                 -------------------------------------
C                                 ---------------------------------
C                                 Reset the rank determination
C                                 tolerance to its default value.
C                                 This is not required but is
C                                 recommended programming practice.
      IOPT(1) = -IOPT(1)
      CALL DUMAG ('math', ICHP, IPUT, NUMOPT, IOPT, DVAL)
C                                 -------------------------------------
C                                 ---------------------------------
C
99999 FORMAT ('  The solution is ', 3F9.4, //, '  The residuals ',
     &       'evaluated at the solution are ', /, 18X, 4F9.4, //,
     &       '  The norm of the residual vector is ', F8.4)
C
      END
```

Output

```
The solution is    0.5000   0.3000   0.2000

The residuals evaluated at the solution are
-1.0000   0.5000   0.5000   0.0000

The norm of the residual vector is   1.2247
```

PLOTP/DPLOTP (Single/Double precision)

Print a plot of up to 10 sets of points.

Usage

```
CALL PLOTP (NDATA, NFUN, X, A, LDA, INC, RANGE, SYMBOL,
            XTITLE, YTITLE, TITLE)
```

Arguments

NDATA — Number of independent variable data points. (Input)

NFUN — Number of sets of points. (Input)
`NFUN` must be less than or equal to 10.

X — Vector of length `NDATA` containing the values of the independent variable. (Input)

A — Matrix of dimension `NDATA` by `NFUN` containing the `NFUN` sets of dependent variable values. (Input)

LDA — Leading dimension of `A` exactly as specified in the dimension statement of the calling program. (Input)

INC — Increment between elements of the data to be used. (Input)
`PLOTP` plots `X`(1 + (`I` – 1) `* INC`) for `I` = 1, 2, …, `NDATA`.

RANGE — Vector of length four specifying minimum *x*, maximum *x*, minimum *y* and maximum *y*. (Input)
`PLOTP` will calculate the range of the axis if the minimum and maximum of that range are equal.

SYMBOL — `CHARACTER` string of length `NFUN`. (Input)
`SYMBOL(I` : `I`) is the symbol used to plot function `I`.

XTITLE — `CHARACTER` string used to label the *x*-axis. (Input)

YTITLE — `CHARACTER` string used to label the *y*-axis. (Input)

TITLE — `CHARACTER` string used to label the plot. (Input)

Comments

1. Informational errors

Type	Code	
3	7	`NFUN` is greater than 10. Only the first 10 functions are plotted.
3	8	`TITLE` is too long. `TITLE` is truncated from the right side.
3	9	`YTITLE` is too long. `YTITLE` is truncated from the right side.

3	10	XTITLE is too long. XTITLE is truncated from the right side. The maximum number of characters allowed depends on the page width and the page length. See Comment 5 below for more information.

2. YTITLE and TITLE are automatically centered.

3. For multiple plots, the character M is used if the same print position is shared by two or more data sets.

4. Output is written to the unit specified by UMACH (page 1201).

5. Default page width is 78 and default page length is 60. They may be changed by calling PGOPT (page 1137) in advance.

Algorithm

Routine PLOTP produces a line printer plot of up to ten sets of points superimposed upon the same plot. A character “M” is printed to indicate multiple points. The user may specify the *x* and *y*-axis plot ranges and plotting symbols. Plot width and length may be reset in advance by calling PGOPT (page 1137).

Example

This example plots the sine and cosine functions from – 3.5 to + 3.5 and sets page width and length to 78 and 40, respectively, by calling PGOPT (page 1137) in advance.

```
      INTEGER    I, INC, LDA, NDATA, NFUN
      REAL       A(200,2), DELX, PI, RANGE(4), X(200)
      CHARACTER  SYMBOL*2
      INTRINSIC  COS, SIN
      EXTERNAL   CONST, PGOPT, PLOTP
C
      DATA SYMBOL/’SC’/
      DATA RANGE/-3.5, 3.5, -1.2, 1.2/
C
      PI     = 3.14159
      NDATA  = 200
      NFUN   = 2
      LDA    = 200
      INC    = 1
      DELX   = 2.*PI/199.
      DO 10  I= 1, 200
         X(I)   = -PI + FLOAT(I-1) * DELX
         A(I,1) = SIN(X(I))
         A(I,2) = COS(X(I))
   10 CONTINUE
C                                 Set page width and length
      CALL PGOPT (-1, 78)
      CALL PGOPT (-2, 40)
      CALL PLOTP (NDATA, NFUN, X, A, LDA, INC, RANGE, SYMBOL,
     &            ’X AXIS’, ’Y AXIS’, ’ C = COS,   S = SIN’)
C
      END
```

Output

```
                         C = COS,   S = SIN

         1.2 ::::+:::::::::::::::+:::::::::::::::+:::::::::::::::+::::
             .                           I                           .
             .                           I                           .
             .                        CCCCCCC     SSSSSSSS           .
             .                       CC  I  CC   SS      SS          .
        0.8 +                      C    I    C SS        SS         +
             .                     C     I     MS          SS        .
             .                    C      I    SSC           SS       .
             .                   CC      I   SS CC           SS      .
             .                  CC       I   S   CC           S      .
        0.4 +                  C        I  S     C            S     +
             .                 C         I SS      C           SS    .
Y            .                CC         I S       CC           S    .
             .                C          IS         C            S   .
A            .               C           SS          C           SS  .
X        0.0 +--S-----------CC-----------S-----------CC-----------S--+
I            .  SS         CC           SS            CC             .
S            .   S         C            SI             C             .
             .    S       CC           S I             CC            .
             .    SS      C           SS I              C            .
        -0.4 +     S     C            S  I               C           +
             .      S   CC           S   I               CC          .
             .      SS CC           SS   I                CC         .
             .       SSC           SS    I                 C         .
             .        MS          SS     I                  C        .
        -0.8 +       C SS        SS      I                   C       +
             .     CC   SS      SS       I                    CC     .
             .  CCCC     SSSSSSSS        I                     CCCC  .
             .  C                        I                        C  .
             .                           I                           .
        -1.2 ::::+:::::::::::::::+:::::::::::::::+:::::::::::::::+::::
                -3              -1               1               3

                                      X AXIS
```

PRIME

Decompose an integer into its prime factors.

Usage

```
CALL PRIME (N, NPF, IPF, IEXP, IPW)
```

Arguments

N — Integer to be decomposed.   (Input)

NPF — Number of different prime factors of `ABS(N)`.   (Output)
If `N` is equal to −1, 0, or 1, `NPF` is set to 0.

IPF — Integer vector of length 13. (Output)
IPF(I) contains the prime factors of the absolute value of N, for I = 1, …, NPF. The remaining 13 – NPF locations are not used.

IEXP — Integer vector of length 13. (Output)
IEXP(I) is the exponent of IPF(I), for I = 1, …, NPF. The remaining 13 – NPF locations are not used.

IPW — Integer vector of length 13. (Output)
IPW(I) contains the quantity IPF(I)**IEXP(I), for I = 1, …, NPF. The remaining 13 – NPF locations are not used.

Comments

The output from PRIME should be interpreted in the following way: ABS(N) = IPF(1)**IEXP(1) * …. * IPF(NPF)**IEXP(NPF).

Algorithm

Routine PRIME decomposes an integer into its prime factors. The number to be factored, *N*, may not have more than 13 distinct factors. The smallest number with more than 13 factors is about 1.3×10^{16}. Most computers do not allow integers of this size.

The routine PRIME is based on a routine by Brenner (1973).

Example

This example factors the integer $144 = 2^4 3^2$.

```
      INTEGER    N
      PARAMETER  (N=144)
C
      INTEGER    IEXP(13), IPF(13), IPW(13), NOUT, NPF
      EXTERNAL   PRIME, UMACH
C                                 Get prime factors of 144
      CALL PRIME (N, NPF, IPF, IEXP, IPW)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print results
      WRITE (NOUT,99999) N, IPF(1), IPF(2), IEXP(1), IEXP(2), IPW(1),
     &                  IPW(2), NPF
C
99999 FORMAT (’  The prime factors for’, I5, ’ are: ’, /, 10X, 2I6, //
     &       , ’  IEXP =’, 2I6, /, ’  IPW  =’, 2I6, /, ’  NPF  =’, I6,
     &       /)
      END
```

Output

```
The prime factors for  144 are:
2     3

IEXP =     4     2
IPW  =    16     9
NPF  =     2
```

CONST/DCONST (Single/Double precision)

Return the value of various mathematical and physical constants.

Usage

`CONST(NAME)`

Arguments

NAME — Character string containing the name of the desired constant. (Input) See Comment 3 for a list of valid constants.

CONST — Value of the constant. (Output)

Comments

1. The case of the character string in `NAME` does not matter. The names “`PI`”, “`Pi`”, “`pI`”, and “`pi`” are equivalent.
2. The units of the physical constants are in `SI` units (meter kilogram-second).
3. The names allowed are as follows:

Name	Description	Value	Ref.
`AMU`	Atomic mass unit	1.6605402E − 27 kg	[1]
`ATM`	Standard atm pressure	1.01325E + 5N/m^2E	[2]
`AU`	Astronomical unit	1.496E + 11*m*	[ ]
Avogadro	Avogadro's number	6.0221367E + 231/mole	[1]
Boltzman	Boltzman's constant	1.380658E − 23J/K	[1]
`C`	Speed of light	2.997924580E + 8*m*/*s*E	[1]
Catalan	Catalan's constant	0.915965 … E	[3]
E	Base of natural logs	2.718…E	[3]
ElectronCharge	Electron change	1.60217733E −19`C`	[1]

Name	Description	Value	Ref.
ElectronMass	Electron mass	9.1093897E − 31 kg	[1]
ElectronVolt	Electron volt	1.60217733E − 19J	[1]
Euler	Euler’s constant gamma	0.577 … E	[3]
Faraday	Faraday constant	9.6485309E + 4C/mole	[1]
FineStructure	fine structure	7.29735308E − 3	[1]
Gamma	Euler’s constant	0.577 … E	[3]
Gas	Gas constant	8.314510J/mole/k	[1]
Gravity	Gravitational constant	6.67259E − 11N * m^2/kg^2	[1]
Hbar	Planck constant / 2 pi	1.05457266E − 34J * s	[1]
PerfectGasVolume	Std vol ideal gas	2.241383E − 2m^3/mole	[*]
Pi	Pi	3.141 … E	[3]
Planck	Planck’s constant h	6.6260755E − 34J * s	[1]
ProtonMass	Proton mass	1.6726231E − 27 kg	[1]
Rydberg	Rydberg’s constant	1.0973731534E + 7/m	[1]
SpeedLight	Speed of light	2.997924580E + 8m/s E	[1]
StandardGravity	Standard g	9.80665m/s^2E	[2]
StandardPressure	Standard atm pressure	1.01325E + 5N/m^2E	[2]
StefanBoltzmann	Stefan-Boltzman	5.67051E − 8W/K^4/m^2	[1]
WaterTriple	Triple point of water	2.7316E + 2K E	[2]

Algorithm

Routine CONST returns the value of various mathematical and physical quantities. For all of the physical values, the Systeme International d’Unites (SI) are used.

The reference for constants are indicated by the code in [ ] Comment above.

[1] Cohen and Taylor (1986)

[2] Liepman (1964)

[3] Precomputed mathematical constants

The constants marked with an E before the [ ] are exact (to machine precision).

To change the units of the values returned by CONST, see CUNIT, page 1187.

Example

In this example, Euler’s constant γ is obtained and printed. Euler’s constant is defined to be

$$\gamma = \lim_{n\to\infty}\left[\sum_{k=1}^{n-1}\frac{1}{k} - \ln n\right]$$

```
      INTEGER    NOUT
      REAL       CONST, GAMMA
      EXTERNAL   CONST, UMACH
C                                  Get output unit number
      CALL UMACH (2, NOUT)
C                                  Get gamma
      GAMMA = CONST('GAMMA')
C                                  Print gamma
      WRITE (NOUT,*) 'GAMMA = ', GAMMA
      END
```

Output

```
GAMMA =    0.577216
```

For another example, see `CUNIT`, page 1187.

CUNIT/DCUNIT (Single/Double precision)

Convert `X` in units `XUNITS` to `Y` in units `YUNITS`.

Usage

`CALL CUNIT (X, XUNITS, Y, YUNITS)`

Arguments

X — Value to be converted. (Input)

XUNITS — Character string containing the name of the units for `X`. (Input)
See Comments for a description of units allowed.

Y — Value in `YUNITS` corresponding to `X` in `XUNITS`. (Output)

YUNITS — Character string containing the name of the units for `Y`. (Input)
See Comments for a description of units allowed.

Comments

1. Strings `XUNITS` and `YUNITS` have the form $U_1 * U_2 * \ldots * U_m/V_1 \ldots V_n$, where U_i and V_i are the names of basic units or are the names of basic units raised to a power. Examples are, "`METER * KILOGRAM/SECOND`", "`M * KG/S`", "`METER`", or "`M/KG`^{2}".

2. The case of the character string in `XUNITS` and `YUNITS` does not matter. The names "`METER`", "`Meter`" and "`meter`" are equivalent.

3. If `XUNITS` is "`SI`", then `X` is assumed to be in the standard international units corresponding to `YUNITS`. Similarly, if `YUNITS` is "`SI`", then `Y` is

assumed to be in the standard international units corresponding to `XUNITS`.

4. The basic unit names allowed are as follows:

Units of time
day, hour = hr, min = minute, s = sec = second, year

Units of frequency
Hertz = Hz

Units of mass
AMU, g = gram, lb = pound, ounce = oz, slug

Units of distance
Angstrom, AU, feet = foot = ft, in = inch, m = meter = metre, micron, mile, mill, parsec, yard

Units of area
acre

Units of volume
l = liter = litre

Units of force
dyne, N = Newton, poundal

Units of energy
BTU(thermochemical), Erg, J = Joule

Units of work
W = watt

Units of pressure
ATM = atomosphere, bar, Pascal

Units of temperature
degC = Celsius, degF = Fahrenheit, degK = Kelvin

Units of viscosity
poise, stoke

Units of charge
Abcoulomb, C = Coulomb, statcoulomb

Units of current
A = ampere, abampere, statampere,

Units of voltage
Abvolt, V = volt

Units of magnetic induction
T = Tesla, Wb = Weber

Other units
1, farad, mole, Gauss, Henry, Maxwell, Ohm

The following metric prefixes may be used with the above units. Note that the one or two letter prefixes may only be used with one letter unit abbreviations.

A	atto	1.E – 18
F	femto	1.E – 15
P	pico	1.E – 12
N	nano	1.E – 9
U	micro	1.E – 6
M	milli	1.E – 3
C	centi	1.E – 2
D	deci	1.E – 1
DK	deca	1.E + 2
K	kilo	1.E + 3
	myria	1.E + 4 (no single letter prefix; M means milli
	mega	1.E + 6 (no single letter prefix; M means milli
G	giga	1.E + 9
T	tera	1.E + 12

5. Informational error

Type	Code	
3	8	A conversion of units of mass to units of force was required for consistency.

Algorithm

Routine CUNIT converts a value expressed in one set of units to a value expressed in another set of units.

The input and output units are checked for consistency unless the input unit is “SI”. SI means the Systeme International d’Unites. This is the meter–kilogram–second form of the metric system. If the input units are “SI”, then the input is assumed to be expressed in the SI units consistent with the output units.

Example

The routine CONST is used to obtain the speed on light, *c*, in SI units. CUNIT is then used to convert *c* to mile/second and to parsec/year. An example involving substitution of force for mass is required in conversion of Newtons/Meter^2 to Pound/Inch^2.

```
C      INTEGER    NOUT
      REAL       CMH, CMS, CONST, CPY
      EXTERNAL   CONST, CUNIT, UMACH
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Get speed of light in SI (m/s)
      CMS = CONST(’SpeedLight’)
      WRITE (NOUT,*) ’Speed of Light = ’, CMS, ’ meter/second’
C                                 Get speed of light in mile/second
      CALL CUNIT (CMS, ’SI’, CMH, ’Mile/Second’)
      WRITE (NOUT,*) ’Speed of Light = ’, CMH, ’ mile/second’
C                                 Get speed of light in parsec/year
      CALL CUNIT (CMS, ’SI’, CPY, ’Parsec/Year’)
      WRITE (NOUT,*) ’Speed of Light = ’, CPY, ’ Parsec/Year’
C                                 Convert Newton/Meter**2 to
C                                 Pound/Inch**2.
      CALL CUNIT(1.E0, ’Newton/Meter**2’, CPSI,
     &                 ’Pound/Inch**2’)
      WRITE(NOUT,*)’ Atmospheres, in Pound/Inch**2 = ’,CPSI
      END
```

Output

```
Speed of Light =     2.99792E+08 meter/second
Speed of Light =     186282. mile/second
Speed of Light =    0.306387 Parsec/Year

*** WARNING  ERROR 8 from CUNIT.  A conversion of units of mass to units of
***          force was required for consistency.
Atmospheres, in Pound/Inch**2 =     1.45038E-04
```

HYPOT/DHYPOT (Single/Double precision)

Compute SQRT(A**2 + B**2) without underflow or overflow.

Usage

HYPOT(A, B)

Arguments

A — First parameter. (Input)

B — Second parameter. (Input)

HYPOT — SQRT(A**2 + B**2). (Output)

Algorithm

Routine HYPOT is based on the routine PYTHAG, used in EISPACK 3. This is an update of the work documented in Garbow et al. (1972).

Example

Compute

$$c = \sqrt{a^2 + b^2}$$

where $a = 10^{20}$ and $b = 2 \times 10^{20}$ without overflow.

```
C                                 Declare variables
      INTEGER    NOUT
      REAL       A, B, C, HYPOT
      EXTERNAL   HYPOT, UMACH
C
      A = 1.0E+20
      B = 2.0E+20
      C = HYPOT(A,B)
C                                 Get output unit number
      CALL UMACH (2, NOUT)
C                                 Print the results
      WRITE (NOUT,'(A,1PE10.4)') ' C = ', C
      END
```

Output

```
C = 2.2361E+20
```

Reference Material

Contents

User Errors

IMSL routines attempt to detect user errors and handle them in a way that provides as much information to the user as possible. To do this, we recognize various levels of severity of errors, and we also consider the extent of the error in the context of the purpose of the routine; a trivial error in one situation may be serious in another. IMSL routines attempt to report as many errors as they can reasonably detect. Multiple errors present a difficult problem in error detection because input is interpreted in an uncertain context after the first error is detected.

What Determines Error Severity

In some cases, the user's input may be mathematically correct, but because of limitations of the computer arithmetic and of the algorithm used, it is not possible to compute an answer accurately. In this case, the assessed degree of accuracy determines the severity of the error. In cases where the routine computes several output quantities, if some are not computable but most are, an error condition exists. The severity depends on an assessment of the overall impact of the error.

Terminal errors

If the user's input is regarded as meaningless, such as `N` = −1 when "`N`" is the number of equations, the routine prints a message giving the value of the erroneous input argument(s) and the reason for the erroneous input. The routine will then cause the user's program to stop. An error in which the user's input is meaningless is the most severe error and is called a *terminal error*. Multiple terminal error messages may be printed from a single routine.

Informational errors

In many cases, the best way to respond to an error condition is simply to correct the input and rerun the program. In other cases, the user may want to take actions in the program itself based on errors that occur. An error that may be used as the basis for corrective action within the program is called an *informational error*. If an informational error occurs, a user-retrievable code is set. A routine can return at most one informational error for a single reference to the routine. The codes for the informational error codes are printed in the error messages.

Other errors

In addition to informational errors, IMSL routines issue error messages for which no user- retrievable code is set. Multiple error messages for this kind of error may be printed. These errors, which generally are not described in the documentation, include terminal errors as well as less serious errors. Corrective action within the calling program is not possible for these errors.

Kinds of Errors and Default Actions

Five levels of severity of errors are defined in the MATH/LIBRARY. Each level has an associated PRINT attribute and a STOP attribute. These attributes have default settings (YES or NO), but they may also be set by the user. The purpose of having multiple error severity levels is to provide independent control of actions to be taken for errors of different severity. Upon return from an IMSL routine, exactly one error state exists. (A code 0 “error” is no informational error.) Even if more than one informational error occurs, only one message is printed (if the PRINT attribute is YES). Multiple errors for which no corrective action within the calling program is reasonable or necessary result in the printing of multiple messages (if the PRINT attribute for their severity level is YES). Errors of any of the severity levels except level 5 may be informational errors.

Level 1: **Note**. A *note* is issued to indicate the possibility of a trivial error or simply to provide information about the computations. Default attributes: PRINT=NO, STOP=NO

Level 2: **Alert**. An *alert* indicates that the user should be advised about events occurring in the software. Default attributes: PRINT=NO, STOP=NO

Level 3: **Warning**. A *warning* indicates the existence of a condition that may require corrective action by the user or calling routine. A warning error may be issued because the results are accurate to only a few decimal places, because some of the output may be erroneous but most of the output is correct, or because some assumptions underlying the analysis technique are violated. Often no corrective action is necessary and the condition can be ignored. Default attributes: PRINT=YES, STOP=NO

Level 4: **Fatal**.A *fatal* error indicates the existence of a condition that may be serious. In most cases, the user or calling routine must take corrective action to recover. Default attributes: PRINT=YES, STOP=YES

Level 5: **Terminal**.A *terminal* error is serious. It usually is the result of an incorrect specification, such as specifying a negative number as the number of equations. These errors may also be caused by various programming errors impossible to diagnose correctly in FORTRAN. The resulting error message may be perplexing to the user. In such cases, the user is advised to compare carefully the actual arguments passed to the routine with the dummy argument descriptions given in the documentation. Special attention should be given to checking argument order and data types.

A terminal error is not an informational error because corrective action within the program is generally not reasonable. In normal usage, execution is terminated immediately when a terminal error occurs. Messages relating to more than one terminal error are printed if they occur. Default attributes: PRINT=YES, STOP=YES

The user can set PRINT and STOP attributes by calling `ERSET` as described in “Routines for Error Handling.”

Errors in Lower-Level Routines

It is possible that a user’s program may call an IMSL routine that in turn calls a nested sequence of lower-level IMSL routines. If an error occurs at a lower level in such a nest of routines and if the lower-level routine cannot pass the information up to the original user-called routine, then a traceback of the routines is produced. The only common situation in which this can occur is when an IMSL routine calls a user-supplied routine that in turn calls another IMSL routine.

Routines for Error Handling

There are three ways in which the user may interact with the IMSL error handling system: (1) to change the default actions, (2) to retrieve the integer code of an informational error so as to take corrective action, and (3) to determine the severity level of an error. The routines to use are `ERSET`, `IERCD`, and `N1RTY`, respectively.

ERSET

Change the default printing or stopping actions when errors of a particular error severity level occur.

Usage

CALL ERSET (IERSVR, IPACT, ISACT)

Arguments

IERSVR — Error severity level indicator. (Input)
If IERSVR = 0, actions are set for levels 1 to 5. If IERSVR is 1 to 5, actions are set for errors of the specified severity level.

IPACT — Printing action. (Input)

IPACT	**Action**
−1	Do not change current setting(s).
0	Do not print.
1	Print.
2	Restore the default setting(s).

ISACT — Stopping action. (Input)

ISACT	**Action**
−1	Do not change current setting(s).
0	Do not stop.
1	Stop.
2	Restore the default setting(s).

IERCD and N1RTY

The last two routines for interacting with the error handling system, IERCD and N1RTY, are INTEGER functions and are described in the following material.

IERCD retrieves the integer code for an informational error. Since it has no arguments, it may be used in the following way:

ICODE = IERCD( )

The function retrieves the code set by the most recently called IMSL routine.

N1RTY retrieves the error type set by the most recently called IMSL routine. It is used in the following way:

ITYPE = N1RTY(1)

ITYPE = 1, 2, 4, and 5 correspond to error severity levels 1, 2, 4, and 5, respectively. ITYPE = 3 and ITYPE = 6 are both warning errors, error severity level 3. While ITYPE = 3 errors are informational errors (IERCD( ) ≠ 0), ITYPE = 6 errors are not informational errors (IERCD( ) = 0).

For software developers requiring additional interaction with the IMSL error handling system, see Aird and Howell (1991).

Examples

Changes to default actions

Some possible changes to the default actions are illustrated below. The default actions remain in effect for the kinds of errors not included in the call to `ERSET`.

To turn off printing of warning error messages:

```
CALL ERSET (3, 0, –1)
```

To stop if warning errors occur:

```
CALL ERSET (3, –1, 1)
```

To print all error messages:

```
CALL ERSET (0, 1, –1)
```

To restore all default settings:

```
CALL ERSET (0, 2, 2)
```

Use of informational error to determine program action

In the program segment below, the Cholesky factorization of a matrix is to be performed. If it is determined that the matrix is not nonnegative definite (and often this is not immediately obvious), the program is to take a different branch.

```
                     .
                     .
                     .
      CALL LFTDS (N, A, LDA, FAC, LDFAC)
      IF (IERCD() .EQ. 2) THEN
C                     Handle matrix that is not nonnegative definite
                     .
                     .
                     .
      END IF
```

Examples of errors

The program below illustrates each of the different types of errors detected by the MATH/LIBRARY routines.

The error messages refer to the argument names that are used in the documentation for the routine, rather than the user’s name of the variable used for the argument. In the message generated by IMSL routine `LINRG` in this example, reference is made to `N`, whereas in the program a literal was used for this argument.

```
      INTEGER    N
      PARAMETER  (N=2)
C
      REAL       A(N,N), AINV(N,N), B(N), X(N)
      EXTERNAL   ERSET, LINRG, LSARG
```

```
C
      DATA A/2.0, -3.0, 2.0, -3.0/
      DATA B/1.0, 2.0/
C                                 Turn on printing and turn off
C                                 stopping for all error types.
      CALL ERSET (0, 1, 0)
C                                 Generate level 4 informational error.
      CALL LSARG (2, A, 2, B, 1, X)
C                                 Generate level 5 terminal error.
      CALL LINRG (-1, A, 2, AINV, 2)
      END
```

Output

```
*** FATAL    ERROR 2 from LSARG.  The input matrix is singular.  Some of
***          the diagonal elements of the upper triangular matrix U of the
***          LU factorization are close to zero.

*** TERMINAL ERROR 1 from LINRG.  The order of the matrix must be positive
***          while N = –1 is given.
```

Example of traceback

The next program illustrates a situation in which a traceback is produced. The program uses the IMSL quadrature routines QDAG and QDAGS to evaluate the double integral

$$\int_0^1 \int_0^1 (x+y)dx\,dy = \int_0^1 g(y)\,dy$$

where

$$g(y) = \int_0^1 (x+y)\,dx = \int_0^1 f(x)\,dx, \text{ with } f(x) = x+y$$

Since both QDAG and QDAGS need 2500 numeric storage units of workspace, and since the workspace allocator uses some space to keep track of the allocations, 6000 numeric storage units of space are explicitly allocated for workspace. Although the traceback shows an error code associated with a terminal error, this code has no meaning to the user; the printed message contains all relevant information. It is not assumed that the user would take corrective action based on knowledge of the code.

```
C                                 Specifications for local variables
      REAL       A, B, ERRABS, ERREST, ERRREL, G, RESULT
      EXTERNAL   G, QDAGS
C                                 Set quadrature parameters
      A      = 0.0
      B      = 1.0
      ERRABS = 0.0
      ERRREL = 0.001
C                                 Do the outer integral
      CALL QDAGS (G, A, B, ERRABS, ERRREL, RESULT, ERREST)
C
      WRITE (*,*) RESULT, ERREST
      END
C
```

```
      REAL FUNCTION G (ARGY)
      REAL       ARGY
C
      INTEGER    IRULE
      REAL       C, D, ERRABS, ERREST, ERRREL, F, Y
      COMMON     /COMY/ Y
      EXTERNAL   F, QDAG
C
      Y      = ARGY
      C      = 0.0
      D      = 1.0
      ERRABS = 0.0
      ERRREL = -0.001
      IRULE  = 1
C
      CALL QDAG (F, C, D, ERRABS, ERRREL, IRULE, G, ERREST)
      RETURN
      END
C
      REAL FUNCTION F (X)
      REAL       X
C
      REAL       Y
      COMMON     /COMY/ Y
C
      F = X + Y
      RETURN
      END
```

Output

```
*** TERMINAL ERROR 4 from Q2AG.  The relative error desired ERRREL =
***          -1.000000E-03.  It must be at least zero.
Here is a traceback of subprogram calls in reverse order:
Routine name                    Error type  Error code
------------                    ----------  ----------
Q2AG                               5           4    (Called internally)
QDAG                               0           0
Q2AGS                              0           0    (Called internally)
QDAGS                              0           0
USER                               0           0
```

Automatic Workspace Allocation

FORTRAN subroutines that work with arrays as input and output often require extra arrays for use as workspace while doing computations or moving around data. IMSL routines generally do not require the user explicitly to allocate such arrays for use as workspace. On most systems the workspace allocation is handled transparently. The only limitation is the actual amount of memory available on the system.

On some systems the workspace is allocated out of a stack that is passed as a FORTRAN array in a named common block `WORKSP`. A very similar use of a workspace stack is described by Fox et al. (1978, pages 116−121). (For compatiblity with older versions of the IMSL Libraries, space is allocated from the COMMON block, if possible.)

The arrays for workspace appear as arguments in lower-level routines. For example, the IMSL routine LSARG (in Chapter 1, “Linear Systems”), which solves systems of linear equations, needs arrays for workspace. LSARG allocates arrays from the common area, and passes them to the lower-level routine L2ARG which does the computations. In the “Comments” section of the documentation for LSARG, the amount of workspace is noted and the call to L2ARG is described. This scheme for using lower-level routines is followed throughout the IMSL Libraries. The names of these routines have a “2” in the second position (or in the third position in double precision routines having a “D” prefix). The user can provide workspace explicitly and call directly the “2-level” routine, which is documented along with the main routine. In a very few cases, the 2-level routine allows additional options that the main routine does not allow.

Prior to returning to the calling program, a routine that allocates workspace generally deallocates that space so that it becomes available for use in other routines.

Changing the Amount of Space Allocated

This section is relevant only to those systems on which the transparent workspace allocator is not available.

By default, the total amount of space allocated in the common area for storage of numeric data is 5000 numeric storage units. (A numeric storage unit is the amount of space required to store an integer or a real number. By comparison, a double precision unit is twice this amount. Therefore the total amount of space allocated in the common area for storage of numeric data is 2500 double precision units.) This space is allocated as needed for INTEGER, REAL, or other numeric data. For larger problems in which the default amount of workspace is insufficient, the user can change the allocation by supplying the FORTRAN statements to define the array in the named common block and by informing the IMSL workspace allocation system of the new size of the common array. To request 7000 units, the statements are

```
      COMMON /WORKSP/ RWKSP
      REAL RWKSP(7000)
      CALL IWKIN(7000)
```

If an IMSL routine attempts to allocate workspace in excess of the amount available in the common stack, the routine issues a fatal error message that indicates how much space is needed and prints statements like those above to guide the user in allocating the necessary amount. The program below uses IMSL routine PERMA (page 1138) to permute rows or columns of a matrix. This routine requires workspace equal to the number of columns, which in this example is too large. (Note that the work vector RWKSP must also provide extra space for bookkeeping.)

```
C                                 Specifications for local variables
```

```
      INTEGER    NRA, NCA, LDA, IPERMU(6000), IPATH
      REAL       A(2,6000)
C                                 Specifications for subroutines
      EXTERNAL   PERMA
C
      NRA = 2
      NCA = 6000
      LDA = 2
C                                 Initialize permutation index
      DO 10 I = 1, NCA
         IPERMU(I) = NCA + 1 - I
   10 CONTINUE
      IPATH = 2
      CALL PERMA (NRA, NCA, A, LDA, IPERMU, IPATH, A, LDA)
      END
```

Output

```
*** TERMINAL ERROR 10 from PERMA.  Insufficient workspace for current
***          allocation(s). Correct by calling IWKIN from main program with
***          the three following statements:  (REGARDLESS OF PRECISION)
***                COMMON /WORKSP/  RWKSP
***                REAL RWKSP(6018)
***                CALL IWKIN(6018)

*** TERMINAL ERROR 10 from PERMA.  Workspace allocation was based on NCA =
***          6000.
```

In most cases, the amount of workspace is dependent on the parameters of the problem so the amount needed is known exactly. In a few cases, however, the amount of workspace is dependent on the data (for example, if it is necessary to count all of the unique values in a vector), so the IMSL routine cannot tell in advance exactly how much workspace is needed. In such cases the error message printed is an estimate of the amount of space required.

Character Workspace

Since character arrays cannot be equivalenced with numeric arrays, a separate named common block WKSPCH is provided for character workspace. In most respects this stack is managed in the same way as the numeric stack. The default size of the character workspace is 2000 character units. (A character unit is the amount of space required to store one character.) The routine analogous to IWKIN used to change the default allocation is IWKCIN.

Machine-Dependent Constants

The function subprograms in this section return machine-dependent information and can be used to enhance portability of programs between different computers. The routines IMACH, AMACH and DMACH describe the computer’s arithmetic. The routine UMACH describes the input, ouput, and error output unit numbers.

```
INTEGER FUNCTION IMACH(I)
```

`IMACH` retrieves machine integer constants that define the arithmetic used by the computer.
`IMACH`(1) = Number of bits per integer storage unit.
`IMACH`(2) = Number of characters per integer storage unit:

Integers are represented in *M*-digit, base *A* form as

$$\sigma \sum_{k=0}^{M} x_k A^k$$

where σ is the sign and $0 \le x_k < A$, $k = 0, \ldots, M$.

Then,
`IMACH`(3) = *A*, the base.
`IMACH`(4) = *M*, the number of base-*A* digits.
`IMACH`(5) = $A^M - 1$, the largest integer.

The machine model assumes that floating-point numbers are represented in normalized *N*-digit, base *B* form as

$$\sigma B^E \sum_{k=1}^{N} x_k B^{-k}$$

where σ is the sign, $0 < x_1 < B$, $0 \le x_k < B$, k = 2, …, *N* and $E_{\mathrm{min}} \le E \le E_{\mathrm{max}}$. Then,

`IMACH`(6) = B, the base.

`IMACH`(7) = N_s, the number of base - B digits in single precision.

`IMACH`(8) = E_{min_s}, the smallest single precision exponent.
`IMACH`(9) = E_{max_s}, the largest single precision exponent.

`IMACH`(10) = N_d, the number of base - B digits in double precision.

`IMACH`(11) = E_{min_d}, the smallest double precision exponent.

`IMACH`(12) = E_{max_d}, the number of base - B digits in double precision

```
REAL FUNCTION AMACH(I)
```

The function subprogram `AMACH` retrieves real machine constants that define the computer’s real or single-precision arithmetic. Such floating-point numbers are represented in normalized N_s-digit, base *B* form as

$$\sigma B^E \sum_{k=1}^{N_s} x_k B^{-k}$$

where σ is the sign, $0 < x_1 < B$, $0 \le x_k < B$, $k = 2, \ldots, N_s$ and

$$E_{\mathrm{min}_s} \le E \le E_{\mathrm{max}_s}$$

Note that B = `IMACH`(6), N_s = `IMACH`(7),

E_{min_s} = `IMACH`(8), and E_{max_s} = `IMACH`(9).

The IEEE standard for binary arithmetic (see IEEE 1985) specifies *quiet* NaN (not a number) as the result of various invalid or ambiguous operations, such as 0/0. The intent is that `AMACH`(6) return a *quiet* NaN. If the machine does not support a quiet NaN, a special value near `AMACH`(2) is returned for `AMACH`(6). On computers that do not have a special representation for infinity, `AMACH`(7) returns the same value as `AMACH`(2).

`AMACH` is defined by the following table:

`AMACH`(1) = $B^{E_{\min_s}-1}$, the smallest normalized positive number.

`AMACH`(2) = $B^{E_{\max_s}}\left(1-B^{-N_s}\right)$, the largest number.

`AMACH`(3) = B^{-N_s}, the smallest relative spacing.

`AMACH`(4) = B^{1-N_s}, the largest relative spacing.

AMACH(5) = $\log_{10}(B)$.

AMACH(6) = NaN (quiet not a number).

AMACH(7) = positive machine infinity.

AMACH(8) = negative machine infinity.

```
DOUBLE PRECISION FUNCTION DMACH(I)
```

The function subprogram `DMACH` retrieves real machine constants that define the computer's double precision arithmetic. Such double-precision floating-point numbers are represented in normalized N_d-digit, base B form as

$$\sigma B^E \sum_{k=1}^{N_d} x_k B^{-k}$$

where σ is the sign, $0 < x_1 < B$, $0 \le x_k < B$, $k = 2, \ldots, N_d$ and

$$E_{\min_d} \le E \le E_{\max_d}$$

Note that B = `IMACH`(6), N_d = `IMACH`(10),

$E_{\min_d}$ = `IMACH`(11), and $E_{\max_d}$ = `IMACH`(12).

The IEEE standard for binary arithmetic (see IEEE 1985) specifies quiet NaN (not a number) as the result of various invalid or ambiguous operations, such as 0/0. The intent is that `DMACH`(6) return a *quiet* NaN. If the machine does not support a quiet NaN, a special value near `DMACH`(2) is returned for `DMACH`(6). On computers that do not have a special representation for infinity, `DMACH`(7) returns the same value as `DMACH`(2).

`DMACH` is defined by the following table:

DMACH(1) = $B^{E_{\min_d}-1}$ the smallest normalized positive number.

DMACH(2) = $B^{E_{\max_d}}\left(1-B^{-N_d}\right)$, the largest number.

DMACH(3) = B^{-N_d}, the smallest relative spacing.

DMACH(4) = B^{1-N_d}, the largest relative spacing.

DMACH(5) = $\log_{10}(B)$

DMACH(6) = NaN (quiet not a number).

DMACH(7) = positive machine infinity.

DMACH(8) = negative machine infinity.

`LOGICAL FUNCTION IFNAN(X), DIFNAN(DX)`

The logical function `IFNAN` checks if the `REAL` argument `X` is NaN (not a number). Similarly, `DIFNAN` checks if the `DOUBLE PRECISION` argument `DX` is NaN.

The functions `IFNAN` and `DIFNAN` are provided to facilitate the transfer of programs across computer systems. This is because the check for NaN can be tricky and not portable across computer systems that do not adhere to the IEEE standard. For example, on computers that support the IEEE standard for binary arithmetic (see IEEE 1985), NaN is specified as a bit format not equal to itself. Thus, the check is performed as

```
IFNAN = X .NE. X
```

On other computers that do not use IEEE floating-point format, the check can be performed in single precision as

```
IFNAN = X .EQ. AMACH(6)
```

The function `IFNAN` or `DIFNAN` is equivalent to the specification of the function `Isnan` listed in the Appendix, (IEEE 1985). The following example illustrates the use of `IFNAN`. If `X` is NaN, a message is printed instead of `X`. (Routine `UMACH`, which is described in the following section, is used to retrieve the output unit number for printing the message.)

```
      INTEGER      NOUT
      REAL         AMACH, X
      LOGICAL      IFNAN
      EXTERNAL     AMACH, IFNAN, UMACH
C
      CALL UMACH (2, NOUT)
C
      X = AMACH(6)
      IF (IFNAN(X)) THEN
         WRITE (NOUT,*) ’ X is NaN (not a number).’
      ELSE
         WRITE (NOUT,*) ’ X = ’, X
      END IF
C
      END
```

Output

```
X is NaN (not a number).
```

SUBROUTINE UMACH(N, NUNIT)

Routine UMACH sets or retrieves the input, output, or error output device unit numbers. UMACH is set automatically so that the default FORTRAN unit numbers for standard input, standard output, and standard error are used. These unit numbers can be changed by inserting a call to UMACH at the beginning of the main program that calls MATH/LIBRARY routines. If these unit numbers are changed from the standard values, the user should insert an appropriate OPEN statement in the calling program. The calling sequence for UMACH is

```
CALL UMACH (N, NUNIT)
```

where NUNIT is the input, output, or error output unit number that is either retrieved or set, depending on which value of N is selected.

The arguments are summarized by the following table:

N	Effect
1	Retrieves input unit number in NUNIT.
2	Retrieves output unit number in NUNIT.
3	Retrieves error output unit number in NUNIT.
–1	Sets the input unit number to NUNIT.
–2	Sets the output unit number to NUNIT.
–3	Sets the error output unit number to NUNIT.

If the value of N is negative, the input, output, or error output unit number is reset to NUNIT. If the value of N is positive, the input, output, or error output unit number is returned in NUNIT.

In the following example, a terminal error is issued from the MATH/LIBRARY AMACH function since the argument is invalid. With a call to UMACH, the error message will be written to a local file named “CHECKERR”.

```
      INTEGER     N, AMACH
      REAL        X
      EXTERNAL   AMACH, UMACH
C                                      Set Parameter
      N = 0
C
      CALL UMACH (-3, 9)
      OPEN (UNIT=9,FILE='CHECKERR')
      X = AMACH(N)
      END
```

The output from this example, written to “CHECKERR” is:

```
*** TERMINAL ERROR 5 from AMACH.  The argument must be between 1 and 8
***           inclusive. N = 0
```

Matrix Storage Modes

In this section, the word *matrix* will be used to refer to a mathematical object, and the word *array* will be used to refer to its representation as a FORTRAN data structure.

General Mode

A *general* matrix is an $N \times N$ matrix A. It is stored in a FORTRAN array that is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of `A`. It must be at least as large as `N`. IMSL general matrix subprograms only refer to values A_{ij} for $i = 1, \ldots, N$ and $j = 1, \ldots, N$. The data type of a general array can be one of `REAL`, `DOUBLE PRECISION`, or `COMPLEX`. If your FORTRAN compiler allows, the nonstandard data type `DOUBLE COMPLEX` can also be declared.

Rectangular Mode

A *rectangular* matrix is an $M \times N$ matrix A. It is stored in a FORTRAN array that is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of `A`. It must be at least as large as `M`. IMSL rectangular matrix subprograms only refer to values A_{ij} for $i = 1, \ldots, M$ and $j = 1, \ldots, N$. The data type of a rectangular array can be `REAL`, `DOUBLE PRECISION`, or `COMPLEX`. If your FORTRAN compiler allows, you can declare the nonstandard data type `DOUBLE COMPLEX`.

Symmetric Mode

A symmetric matrix is a square $N \times N$ matrix A, such that $A^T = A$. (A^T is the transpose of A.) It is stored in a FORTRAN array that is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of `A`. It must be at least as large as `N`. IMSL symmetric matrix subprograms only refer to the upper or to the lower half of `A` (i.e., to values A_{ij} for $i = 1, \ldots, N$ and $j = i, \ldots, N$, or A_{ij} for $j = 1, \ldots, N$ and $i = j, \ldots, N$). The data type of a symmetric array can be one of `REAL` or `DOUBLE PRECISION`. Use of the upper half of the array is denoted in the BLAS that compute with symmetric matrices, page 1047, using the `CHARACTER*1` flag `UPLO = 'U'`. Otherwise, `UPLO = 'L'` denotes that the lower half of the array is used.

Hermitian Mode

A *Hermitian* matrix is a square $N \times N$ matrix A, such that

$$\overline{A}^T = A$$

The matrix

$$\overline{A}$$

is the complex conjugate of A and

$$A^H \equiv \overline{A}^T$$

is the conjugate transpose of A. For Hermitian matrices, $A^H = A$. The matrix is stored in a FORTRAN array that is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of `A`. It must be at least as large as N. IMSL Hermitian matrix subprograms only refer to the upper or to the lower half of `A` (i.e., to values A_{ij} for $i = 1, \ldots, N$ and $j = i, \ldots, N$., or A_{ij} for $j = 1, \ldots, N$ and $i = j, \ldots, N$). Use of the upper half of the array is denoted in the BLAS that compute with Hermitian matrices, page 1047, using the `CHARACTER*1` flag `UPLO` = `'U'`. Otherwise, `UPLO` = `'L'` denotes that the lower half of the array is used. The data type of a Hermitian array can be `COMPLEX` or, if your FORTRAN compiler allows, the nonstandard data type `DOUBLE COMPLEX`.

Triangular Mode

A *triangular* matrix is a square $N \times N$ matrix A such that values $A_{ij} = 0$ for $i < j$ or $A_{ij} = 0$ for $i > j$. The first condition defines a *lower* triangular matrix while the second condition defines an *upper* triangular matrix. A lower triangular matrix A is stored in the lower triangular part of a FORTRAN array `A`. An upper triangular matrix is stored in the upper triangular part of a FORTRAN array. Triangular matrices are called *unit* triangular whenever $A_{jj} = 1$, $j = 1, \ldots, N$. For unit triangular matrices, only the strictly lower or upper parts of the array are referenced. This is denoted in the BLAS that compute with triangular matrices, page 1047, using the `CHARACTER*1` flag `DIAG = 'U'`. Otherwise, `DIAG = 'N'` denotes that the diagonal array terms should be used. For unit triangular matrices, the diagonal terms are each used with the mathematical value 1. The array diagonal term does not need to be 1.0 in this usage. Use of the upper half of the array is denoted in the BLAS that compute with triangular matrices, page 1047, using the `CHARACTER*1` flag `UPLO = 'U'`. Otherwise, `UPLO = 'L'` denotes that the lower half of the array is used. The data type of an array that contains a triangular matrix can be one of `REAL`, `DOUBLE PRECISION`, or `COMPLEX`. If your FORTRAN compiler allows, the nonstandard data type `DOUBLE COMPLEX` can also be declared.

Band Storage Mode

A *band matrix* is an $M \times N$ matrix A with all of its nonzero elements “close” to the main diagonal. Specifically, values $A_{ij} = 0$ if $i - j >$ NLCA or $j - i >$ NUCA. The integers NLCA and NUCA are the *lower* and *upper* band widths. The integer $m =$ NLCA + NUCA + 1 is the total band width. The diagonals, other than the main diagonal, are called *codiagonals*. While any $M \times N$ matrix is a band matrix, the band matrix mode is most useful only when the number of nonzero codiagonals is much less than m.

In the band storage mode, the NLCA lower codiagonals and NUCA upper codiagonals are stored in the rows of a FORTRAN array of dimension $m \times N$. The elements are stored in the same column of the array as they are in the matrix. The values A_{ij} inside the band width are stored in array positions ($i - j$ + NUCA + 1, j). This array is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter LDA is called the *leading dimension* of A. It must be at least as large as m. The data type of a band matrix array can be one of REAL, DOUBLE PRECISION, COMPLEX or, if your FORTRAN compiler allows, the nonstandard data type DOUBLE COMPLEX . Use of the CHARACTER*1 flag TRANS='N' in the BLAS, page 1047, specifies that the matrix A is used. The flag value

TRANS = 'T' uses A^T

while

TRANS = 'C' uses $\overline{A}^T$

For example, consider a real 5 × 5 band matrix with 1 lower and 2 upper codiagonals, stored in the FORTRAN array declared by the following statements:

```
PARAMETER (N=5, NLCA=1, NUCA=2)
REAL A(NLCA+NUCA+1, N)
```

The matrix A has the form

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ A_{21} & A_{22} & A_{23} & A_{24} & 0 \\ 0 & A_{32} & A_{33} & A_{34} & A_{35} \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{54} & A_{55} \end{bmatrix}$$

As a FORTRAN array, it is

$$A = \begin{bmatrix} \times & \times & A_{13} & A_{24} & A_{35} \\ \times & A_{12} & A_{23} & A_{34} & A_{45} \\ A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \\ A_{21} & A_{32} & A_{43} & A_{54} & \times \end{bmatrix}$$

The entries marked with an `x` in the above array are not referenced by the IMSL band subprograms.

Band Symmetric Storage Mode

A *band symmetric* matrix is a band matrix that is also symmetric. The band symmetric storage mode is similar to the band mode except only the lower or upper codiagonals are stored.

In the band symmetric storage mode, the `NCODA` upper codiagonals are stored in the rows of a FORTRAN array of dimension (`NCODA` + `1`) × N. The elements are stored in the same column of the array as they are in the matrix. Specifically, values A_{ij}, $j \le i$ inside the band are stored in array positions ($i - j$ + `NCODA` + 1, j). This is the storage mode designated by using the `CHARACTER*`1 flag `UPLO = 'U'` in Level 2 BLAS that compute with band symmetric matrices, page 1047. Alternatively, A_{ij}, $j \le i$, inside the band, are stored in array positions ($i - j + 1$, j). This is the storage mode designated by using the `CHARACTER*`1 flag `UPLO = 'L'` in these Level 2 BLAS, page 1047. The array is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of A. It must be at least as large as `NCODA` + 1. The data type of a band symmetric array can be `REAL` or `DOUBLE PRECISION`.

For example, consider a real 5 × 5 band matrix with 2 codiagonals. Its FORTRAN declaration is

```
PARAMETER (N=5, NCODA=2)
REAL A(NCODA+1, N)
```

The matrix A has the form

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ A_{12} & A_{22} & A_{23} & A_{24} & 0 \\ A_{13} & A_{23} & A_{33} & A_{34} & A_{35} \\ 0 & A_{24} & A_{34} & A_{44} & A_{45} \\ 0 & 0 & A_{35} & A_{45} & A_{55} \end{bmatrix}$$

Since A is symmetric, the values $A_{ij} = A_{ji}$. In the FORTRAN array, it is

$$A = \begin{bmatrix} \times & \times & A_{13} & A_{24} & A_{35} \\ \times & A_{12} & A_{23} & A_{34} & A_{45} \\ A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \end{bmatrix}$$

The entries marked with an × in the above array are not referenced by the IMSL band symmetric subprograms.

An alternate storage mode for band symmetric matrices is designated using the `CHARACTER*`1 flag `UPLO = 'L'` in Level 2 BLAS that compute with band symmetric matrices, page 1047. In that case, the example matrix is represented as

$$A = \begin{bmatrix} A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \\ A_{12} & A_{23} & A_{34} & A_{45} & \times \\ A_{13} & A_{24} & A_{35} & \times & \times \end{bmatrix}$$

Band Hermitian Storage Mode

A *band Hermitian* matrix is a band matrix that is also Hermitian. The band Hermitian mode is a complex analogue of the band symmetric mode.

In the band Hermitian storage mode, the `NCODA` upper codiagonals are stored in the rows of a FORTRAN array of dimension (`NCODA` + 1) × *N*. The elements are stored in the same column of the array as they are in the matrix. In the Level 2 BLAS, page 1047, this is denoted by using the `CHARACTER*`1 flag `UPLO ='U'`. The array is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of *A*. It must be at least as large as (`NCODA` + 1) . The data type of a band Hermitian array can be `COMPLEX` or, if your FORTRAN compiler allows, the nonstandard data type `DOUBLE COMPLEX`.

For example, consider a complex 5 × 5 band matrix with 2 codiagonals. Its FORTRAN declaration is

```
PARAMETER (N=5, NCODA = 2)
COMPLEX A(NCODA + 1, N)
```

The matrix *A* has the form

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ \overline{A}_{12} & A_{22} & A_{23} & A_{24} & 0 \\ \overline{A}_{13} & \overline{A}_{23} & A_{33} & A_{34} & A_{35} \\ 0 & \overline{A}_{24} & \overline{A}_{34} & A_{44} & A_{45} \\ 0 & 0 & \overline{A}_{35} & \overline{A}_{45} & A_{55} \end{bmatrix}$$

where the value

$$\overline{A}_{ij}$$

is the complex conjugate of A_{ij}. This matrix represented as a FORTRAN array is

$$A = \begin{bmatrix} \times & \times & A_{13} & A_{24} & A_{35} \\ \times & A_{12} & A_{23} & A_{34} & A_{45} \\ A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \end{bmatrix}$$

The entries marked with an × in the above array are not referenced by the IMSL band Hermitian subprograms.

An alternate storage mode for band Hermitian matrices is designated using the `CHARACTER*1` flag `UPLO = 'L'` in Level 2 BLAS that compute with band Hermitian matrices, page 1047. In that case, the example matrix is represented as

$$A = \begin{bmatrix} A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \\ \overline{A}_{12} & \overline{A}_{23} & \overline{A}_{34} & \overline{A}_{45} & \times \\ \overline{A}_{13} & \overline{A}_{24} & \overline{A}_{35} & \times & \times \end{bmatrix}$$

Band Triangular Storage Mode

A *band triangular* matrix is a band matrix that is also triangular. In the band triangular storage mode, the `NCODA` codiagonals are stored in the rows of a FORTRAN array of dimension (`NCODA` + 1) × N. The elements are stored in the same column of the array as they are in the matrix. For usage in the Level 2 BLAS, page 1047, the `CHARACTER*1` flag `DIAG` has the same meaning as used in section “Triangular Storage Mode”. The flag `UPLO` has the meaning analogous with its usage in the section “Banded Symmetric Storage Mode”. This array is declared by the following statement:

```
DIMENSION A(LDA,N)
```

The parameter `LDA` is called the *leading dimension* of A. It must be at least as large as (`NCODA` + 1).

For example, consider a 5 ×5 band upper triangular matrix with 2 codiagonals. Its FORTRAN declaration is

```
PARAMETER (N = 5, NCODA = 2)
COMPLEX A(NCODA + 1, N)
```

The matrix A has the form

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ 0 & A_{22} & A_{23} & A_{24} & 0 \\ 0 & 0 & A_{33} & A_{34} & A_{35} \\ 0 & 0 & 0 & A_{44} & A_{45} \\ 0 & 0 & 0 & 0 & A_{55} \end{bmatrix}$$

This matrix represented as a FORTRAN array is

$$A = \begin{bmatrix} \times & \times & A_{13} & A_{24} & A_{35} \\ \times & A_{12} & A_{23} & A_{34} & A_{45} \\ A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \end{bmatrix}$$

This corresponds to the `CHARACTER*1` flags `DIAG = 'N'` and `UPLO = 'U'`. The matrix A^T is represented as the FORTRAN array

$$A = \begin{bmatrix} A_{11} & A_{22} & A_{33} & A_{44} & A_{55} \\ A_{12} & A_{23} & A_{34} & A_{45} & \times \\ A_{13} & A_{24} & A_{35} & \times & \times \end{bmatrix}$$

This corresponds to the `CHARACTER*1` flags `DIAG = 'N'` and `UPLO = 'L'`. In both examples, the entries indicated with an × are not referenced by IMSL subprograms.

Codiagonal Band Symmetric Storage Mode

This is an alternate storage mode for band symmetric matrices. It is not used by any of the BLAS, page 1047. Storing data in a form transposed from the **Band Symmetric Storage Mode** maintains unit spacing between consecutive referenced array elements. This data structure is used to get good performance in the Cholesky decomposition algorithm that solves positive definite symmetric systems of linear equations $Ax = b$. The data type can be `REAL` or `DOUBLE PRECISION`. In the codiagonal band symmetric storage mode, the `NCODA` upper codiagonals and right-hand-side are stored in columns of this FORTRAN array. This array is declared by the following statement:

```
DIMENSION A(LDA, NCODA + 2)
```

The parameter `LDA` is the *leading positive dimension* of *A*. It must be at least as large as `N` + `NCODA`.

Consider a real symmetric 5 × 5 matrix with 2 codiagonals

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ A_{12} & A_{22} & A_{23} & A_{24} & 0 \\ A_{13} & A_{23} & A_{33} & A_{34} & A_{35} \\ 0 & A_{24} & A_{34} & A_{44} & A_{45} \\ 0 & 0 & A_{35} & A_{45} & A_{55} \end{bmatrix}$$

and a right-hand-side vector

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}$$

A FORTRAN declaration for the array to hold this matrix and right-hand-side vector is

```
PARAMETER (N = 5, NCODA = 2, LDA = N + NCODA)
REAL A(LDA, NCODA + 2)
```

The matrix and right-hand-side entries are placed in the FORTRAN array *A* as follows:

$$A = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ A_{11} & \times & \times & b_1 \\ A_{22} & A_{12} & \times & b_2 \\ A_{33} & A_{23} & A_{13} & b_3 \\ A_{44} & A_{34} & A_{24} & b_4 \\ A_{55} & A_{45} & A_{35} & b_5 \end{bmatrix}$$

Entries marked with an × do not need to be defined. Certain of the IMSL band symmetric subprograms will initialize and use these values during the solution process. When a solution is computed, the b_i, $i = 1, \ldots, 5$, are replaced by x_i, $i = 1, \ldots, 5$.

The nonzero A_{ij}, $j \geq i$, are stored in array locations $A(j +$ `NCODA`, $(j - i) + 1)$. The right-hand-side entries b_j are stored in locations $A(j +$ `NCODA`, `NCODA` $+ 2)$. The solution entries x_j are returned in $A(j +$ `NCODA`, `NCODA` $+ 2)$.

Codiagonal Band Hermitian Storage Mode

This is an alternate storage mode for band Hermitian matrices. It is not used by any of the BLAS, page 1047. In the codiagonal band Hermitian storage mode, the real and imaginary parts of the 2 * NCODA + 1 upper codiagonals and right-hand-side are stored in columns of a FORTRAN array. Note that there is no explicit use of the COMPLEX or the nonstandard data type DOUBLE COMPLEX data type in this storage mode.

For *Hermitian* complex matrices,

$$A = U + \sqrt{-1}V$$

where U and V are real matrices. They satisfy the conditions $U = U^T$ and $V = -V^T$. The right-hand-side

$$b = c + \sqrt{-1}\,d$$

where c and d are real vectors. The solution vector is denoted as

$$x = u + \sqrt{-1}v$$

where u and v are real. The storage is declared with the following statement

```
DIMENSION A(LDA, 2*NCODA + 3)
```

The parameter LDA is the *leading positive dimension* of A. It must be at least as large as N + NCODA.

The diagonal terms U_{jj} are stored in array locations A (j + NCODA, 1). The diagonal V_{jj} are zero and are not stored. The nonzero U_{ij}, $j > i$, are stored in locations A(j + NCODA, 2 * ($j - i$)).

The nonzero V_{ij} are stored in locations A(j + NCODA, 2*($j - i$) + 1). The right side vector b is stored with c_j and d_j in locations A(j + NCODA, 2*NCODA + 2) and A(j + NCODA, 2*NCODA + 3) respectively. The real and imaginary parts of the solution, u_j and v_j, respectively overwrite c_j and d_j.

Consider a complex hermitian 5 × 5 matrix with 2 codiagonals

$$A = \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & 0 \\ U_{12} & U_{22} & U_{23} & U_{24} & 0 \\ U_{13} & U_{23} & U_{33} & U_{34} & U_{35} \\ 0 & U_{24} & U_{34} & U_{44} & U_{45} \\ 0 & 0 & U_{35} & U_{45} & U_{55} \end{bmatrix} + \sqrt{-1} \begin{bmatrix} 0 & V_{12} & V_{13} & 0 & 0 \\ -V_{12} & 0 & V_{23} & V_{24} & 0 \\ -V_{13} & -V_{23} & 0 & V_{34} & V_{35} \\ 0 & -V_{24} & -V_{34} & 0 & V_{45} \\ 0 & 0 & -V_{35} & -V_{45} & 0 \end{bmatrix}$$

and a right-hand-side vector

$$b = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} + \sqrt{-1} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix}$$

A FORTRAN declaration for the array to hold this matrix and right-hand-side vector is

```
PARAMETER (N = 5, NCODA = 2, LDA = N + NCODA)
REAL A(LDA,2*NCODA + 3)
```

The matrix and right-hand-side entries are placed in the FORTRAN array *A* as follows:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ U_{11} & \times & \times & \times & \times & c_1 & d_1 \\ U_{22} & U_{12} & V_{12} & \times & \times & c_2 & d_2 \\ U_{33} & U_{23} & V_{23} & U_{13} & V_{13} & c_3 & d_3 \\ U_{44} & U_{34} & V_{34} & U_{24} & V_{24} & c_4 & d_4 \\ U_{55} & U_{45} & V_{45} & U_{35} & V_{35} & c_5 & d_5 \end{bmatrix}$$

Entries marked with an × do not need to be defined.

Sparse Matrix Storage Mode

The sparse linear algebraic equation solvers in Chapter 1 accept the input matrix in *sparse storage mode*. This structure consists of `INTEGER` values `N` and `NZ`, the matrix dimension and the total number of nonzero entries in the matrix. In addition, there are two `INTEGER` arrays `IROW(*)` and `JCOL(*)` that contain unique matrix row and column coordinates where values are given. There is also an array *A*(`*`) of values. All other entries of the matrix are zero. Each of the arrays `IROW(*)`, `JCOL(*)`, *A*(`*`) must be of size `NZ`. The correspondence between matrix and array entries is given by

$$A_{\mathtt{IROW}(i),\mathtt{JCOL}(i)} = A(i),\ i = 1, \ldots, \mathtt{NZ}$$

The data type for *A*(`*`) can be one of `REAL`, `DOUBLE PRECISION`, or `COMPLEX`. If your FORTRAN compiler allows, the nonstandard data type `DOUBLE COMPLEX` can also be declared.

For example, consider a real 5 × 5 sparse matrix with 11 nonzero entries. The matrix *A* has the form

$$A = \begin{bmatrix} A_{11} & 0 & A_{13} & A_{14} & 0 \\ A_{21} & A_{22} & 0 & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{43} & 0 & 0 \\ 0 & 0 & 0 & A_{54} & A_{55} \end{bmatrix}$$

Declarations of arrays and definitions of the values for this sparse matrix are

```
      PARAMETER (NZ = 11, N = 5)
      DIMENSION IROW(NZ), JCOL(NZ), A(NZ)
      DATA IROW /1,1,1,2,2,3,3,3,4,5,5/
      DATA JCOL /1,3,4,1,2,2,3,4,3,4,5/
      DATA A    /$A_{11}$,$A_{13}$,$A_{14}$,$A_{21}$,$A_{22}$,$A_{32}$,$A_{33}$,$A_{34}$,
     &           $A_{43}$,$A_{54}$,$A_{55}$/
```

Reserved Names

When writing programs accessing the MATH/LIBRARY, the user should choose FORTRAN names that do not conflict with names of IMSL subroutines, functions, or named common blocks, such as the workspace common block WORKSP (see page 1199). The user needs to be aware of two types of name conflicts that can arise. The first type of name conflict occurs when a name (technically a *symbolic name*) is not uniquely defined within a program unit (either a main program or a subprogram). For example, such a name conflict exists when the name RCURV is used to refer both to a type REAL variable and to the IMSL subroutine RCURV in a single program unit. Such errors are detected during compilation and are easy to correct. The second type of name conflict, which can be more serious, occurs when names of program units and named common blocks are not unique. For example, such a name conflict would be caused by the user defining a subroutine named WORKSP and also referencing an MATH/LIBRARY subroutine that uses the named common block WORKSP. Likewise, the user must not define a subprogram with the same name as a subprogram in the MATH/LIBRARY, that is referenced directly by the user's program or is referenced indirectly by other MATH/LIBRARY subprograms.

The MATH/LIBRARY consists of many routines, some that are described in the *User's Manual* and others that are not intended to be called by the user and, hence, that are not documented. If the choice of names were completely random over the set of valid FORTRAN names, and if a program uses only a small subset of the MATH/LIBRARY, the probability of name conflicts is very small. Since names are usually chosen to be mnemonic, however, the user may wish to take some precautions in choosing FORTRAN names.

Many IMSL names consist of a root name that may have a prefix to indicate the type of the routine. For example, the IMSL single precision subroutine for fitting a polynomial by least squares has the name RCURV, which is the root name, and the corresponding IMSL double precision routine has the name DRCURV. Associated with these two routines are R2URV and DR2URV. RCURV and DRCURV are listed in the Alphabetical Index of Routines, but R2URV and DR2URV are not. The user of RCURV must consider both names RCURV and R2URV to be reserved; likewise, the user of DRCURV must consider both names DRCURV and DR2URV to be reserved. The names of *all* routines and named common blocks that are used by the MATH/LIBRARY and that do not have a numeral in the second position of the root name are listed in the Alphabetical Index of Routines. Some of the routines in this Index (such as the "Level 2 BLAS") are not intended to be called by the user and so are not documented.

The careful user can avoid any conflicts with IMSL names if the following rules are observed:

- Do not choose a name that appears in the Alphabetical Summary of Routines in the *User's Manual*.
- Do not choose a name of three or more characters with a numeral in the second or third position.

These simplified rules include many combinations that are, in fact, allowable. However, if the user selects names that conform to these rules, no conflict will be encountered.

Deprecated and Renamed Routines

The routines in the following list are being deprecated in Version 2.0 of MATH/LIBRARY. A deprecated routine is one that is no longer used by anything in the library but is being included in the product for those users who may be currently referencing it in their application. However, any future versions of MATH/LIBRARY will not include these routines. If any of these routines are being called within an application, it is recommended that you change your code or retain the deprecated routine before replacing this library with the next version. Most of these routines were called by users only when they needed to set up their own workspace. Thus, the impact of these changes should be limited.

CZADD
CZINI
CZMUL
CZSTO
DE2AHF
DE2ASF
DE2BHF
DE2BSB
DE2BSF
DE2CCG
DE2CCH
DE2CHF
DE2CRG
DE2CRH
DE2CSB
DE2EHF
DE2ESB
DE2FHF
DE2FSB
DE2FSF
DE2LRH
DE2LSB
DE3CRG
DE3CRH
DE3LSF
DE4CRG
DE4ESF
DE5CRG
DE7CRG
DG2CCG
DG2CRG
DG2DF
DG2IND
DG2LCG
DG2LRG
DG3CCG
DG3DF
DG4CCG
DG5CCG
DG7CRG
E2AHF
E2BHF
E2BSB
E2BSF
E2CCG
E2CCH
E2CHF
E2CRG
E2CRH
E2CSB
E2EHF
E2ESB
E2FHF
E2FSB
E2FSF
E2LCG
E2LCH
E2LHF
E2LRG
E2LRH
E4CRG
E4ESF
E5CRG
E7CRG
G2CCG
G2CRG
G2LCG
G2LRG
G3CCG
G4CCG
G5CCG
G7CRG
SDADD
SDINI
SDMUL
SDSTO
SHOUAP
SHOUTR

DE2LCG
DE2LCH
DE2LHF
DE2LRG
DHOUAP
DHOUTR
DIVPBS
E2ASF
E2LSB
E3CRG
E3CRH
E3LSF

The following routines have been renamed due to naming conflicts with other software manufacturers.

CTIME – replaced with CPSEC
DTIME – replaced with TIMDY
PAGE – replaced with PGOPT

Appendix A: GAMS Index

Description

This index lists routines in MATH/LIBRARY by a tree-structured classification scheme known as GAMS Version 2.0 (Boisvert, Howe, Kahaner, and Springmann (1990). Only the GAMS classes that contain MATH/LIBRARY routines are included in the index. The page number for the documentation and the purpose of the routine appear alongside the routine name.

The first level of the full classification scheme contains the following major subject areas:

A.	Arithmetic, Error Analysis
B.	Number Theory
C.	Elementary and Special Functions
D.	Linear Algebra
E.	Interpolation
F.	Solution of Nonlinear Equations
G.	Optimization
H.	Differentiation and Integration
I.	Differential and Integral Equations
J.	Integral Transforms
K.	Approximation
L.	Statistics, Probability
M.	Simulation, Stochastic Modeling
N.	Data Handling
O.	Symbolic Computation
P.	Computational Geometry
Q.	Graphics
R.	Service Routines
S.	Software Development Tools
Z.	Other

There are seven levels in the classification scheme. Classes in the first level are identified by a capital letter as is given above. Classes in the remaining levels are identified by alternating letter-and-number combinations. A single letter (a-z) is used with the odd-numbered levels. A number (1–26) is used within the even-numbered levels.

IMSL MATH/LIBRARY

A .... ARITHMETIC, ERROR ANALYSIS

A3 .... Real

A3c ...... Extended precision

`DQADD` Add a double-precision scalar to the accumulator in extended precision.
`DQINI` Initialize an extended-precision accumulator with a double-precision scalar.
`DQMUL` Multiply double-precision scalars in extended precision.
`DQSTO` Store a double-precision approximation to an extended-precision scalar.

A4 .... Complex

A4c ...... Extended precision

`ZQADD` Add a double complex scalar to the accumulator in extended precision.
`ZQINI` Initialize an extended-precision complex accumulator to a double complex scalar.
`ZQMUL` Multiply double complex scalars using extended precision.
`ZQSTO` Store a double complex approximation to an extended-precision complex scalar.

A6 .... Change of representation

A6c ...... Decomposition, construction

`PRIME` Decompose an integer into its prime factors.

B .... NUMBER THEORY

`PRIME` Decompose an integer into its prime factors.

C .... ELEMENTARY AND SPECIAL FUNCTIONS

C2 .... Powers, roots, reciprocals

`HYPOT` Compute $\sqrt{a^2 + b^2}$ without underflow or overflow.

C19 ...... Other special functions

`CONST` Return the value of various mathematical and physical constants.
`CUNIT` Convert `X` in units `XUNITS` to `Y` in units `YUNITS`.

D .... LINEAR ALGEBRA

D1 .... Elementary vector and matrix operations

D1a ...... Elementary vector operations

D1a1 .... Set to constant

`CSET` Set the components of a vector to a scalar, all complex.
`ISET` Set the components of a vector to a scalar, all integer.

SSET Set the components of a vector to a scalar, all single precision.

D1a2 .... Minimum and maximum components

ICAMAX Find the smallest index of the component of a complex vector having maximum magnitude.

ICAMIN Find the smallest index of the component of a complex vector having minimum magnitude.

IIMAX Find the smallest index of the maximum component of a integer vector.

IIMIN Find the smallest index of the minimum of an integer vector.

ISAMAX Find the smallest index of the component of a single-precision vector having maximum absolute value.

ISAMIN Find the smallest index of the component of a single-precision vector having minimum absolute value.

ISMAX Find the smallest index of the component of a single-precision vector having maximum value.

ISMIN Find the smallest index of the component of a single-precision vector having minimum value.

D1a3 .... Norm

D1a3a .. L_1 (sum of magnitudes)

DISL1 Compute the 1-norm distance between two points.

SASUM Sum the absolute values of the components of a single-precision vector.

SCASUM Sum the absolute values of the real part together with the absolute values of the imaginary part of the components of a complex vector.

D1a3b .. L_2 (Euclidean norm)

DISL2 Compute the Euclidean (2-norm) distance between two points.

SCNRM2 Compute the Euclidean norm of a complex vector.

SNRM2 Compute the Euclidean length or L_2 norm of a single-precision vector.

D1a3c .. L_∞ (maximum magnitude)

DISLI Compute the infinity norm distance between two points.

ICAMAX Find the smallest index of the component of a complex vector having maximum magnitude.

ISAMAX Find the smallest index of the component of a single-precision vector having maximum absolute value.

D1a4 .... Dot product (inner product)

CDOTC Compute the complex conjugate dot product, $\bar{x}^T y$.

CDOTU Compute the complex dot product $x^T y$.

CZCDOT Compute the sum of a complex scalar plus a complex conjugate dot product, $a + \bar{x}^T y$, using a double-precision accumulator.

CZDOTA Compute the sum of a complex scalar, a complex dot product and the double-complex accumulator, which is set to the result ACC ← ACC + $a + x^T y$.

CZDOTC Compute the complex conjugate dot product, $\bar{x}^T y$, using a double-precision accumulator.

CZDOTI Compute the sum of a complex scalar plus a complex dot product using a double-complex accumulator, which is set to the result ACC ← $a + x^T y$.

CZDOTU Compute the complex dot product $x^T y$ using a double-precision accumulator.

CZUDOT Compute the sum of a complex scalar plus a complex dot product, $a + x^T y$, using a double-precision accumulator.

DSDOT Compute the single-precision dot product $x^T y$ using a double precision accumulator.

SDDOTA Compute the sum of a single-precision scalar, a single-precision dot product and the double-precision accumulator, which is set to the result ACC ← ACC + a + $x^T y$.

SDDOTI Compute the sum of a single-precision scalar plus a singleprecision dot product using a double-precision accumulator, which is set to the result ACC ← $a + x^T y$.

SDOT Compute the single-precision dot product $x^T y$.

SDSDOT Compute the sum of a single-precision scalar and a single precision dot product, $a + x^T y$, using a double-precision accumulator.

D1a5 .... Copy or exchange (swap)

CCOPY Copy a vector x to a vector y, both complex.

CSWAP Interchange vectors x and y, both complex.

ICOPY Copy a vector x to a vector y, both integer.

ISWAP Interchange vectors x and y, both integer.

SCOPY Copy a vector x to a vector y, both single precision.

SSWAP Interchange vectors x and y, both single precision.

D1a6 .... Multiplication by scalar

CSCAL Multiply a vector by a scalar, $y \leftarrow ay$, both complex.

CSSCAL Multiply a complex vector by a single-precision scalar, $y \leftarrow ay$.

CSVCAL Multiply a complex vector by a single-precision scalar and store the result in another complex vector, $y \leftarrow ax$.

CVCAL Multiply a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all complex.

`SSCAL` Multiply a vector by a scalar, $y \leftarrow ay$, both single precision.

`SVCAL` Multiply a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all single precision.

D1a7 .... Triad ($ax + y$ for vectors x, y and scalar a)

`CAXPY` Compute the scalar times a vector plus a vector, $y \leftarrow ax + y$, all complex.

`SAXPY` Compute the scalar times a vector plus a vector, $y \leftarrow ax + y$, all single precision.

D1a8 .... Elementary rotation (Givens transformation) (*search also class D1b10*)

`CSROT` Apply a complex Givens plane rotation.

`CSROTM` Apply a complex modified Givens plane rotation.

`SROT` Apply a Givens plane rotation in single precision.

`SROTM` Apply a modified Givens plane rotation in single precision.

D1a10 .. Convolutions

`RCONV` Compute the convolution of two real vectors.

`VCONC` Compute the convolution of two complex vectors.

`VCONR` Compute the convolution of two real vectors.

D1a11 .. Other vector operations

`CADD` Add a scalar to each component of a vector, $x \leftarrow x + a$, all complex.

`CSUB` Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all complex.

`DISL1` Compute the 1-norm distance between two points.

`DISL2` Compute the Euclidean (2-norm) distance between two points.

`DISLI` Compute the infinity norm distance between two points.

`IADD` Add a scalar to each component of a vector, $x \leftarrow x + a$, all integer.

`ISUB` Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all integer.

`ISUM` Sum the values of an integer vector.

`SADD` Add a scalar to each component of a vector, $x \leftarrow x + a$, all single precision.

`SHPROD` Compute the Hadamard product of two single-precision vectors.

`SPRDCT` Multiply the components of a single-precision vector.

`SSUB` Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all single precision.

`SSUM` Sum the values of a single-precision vector.

`SXYZ` Compute a single-precision *xyz* product.

D1b ...... Elementary matrix operations

`CGERC` Compute the rank-one update of a complex general matrix:
$A \leftarrow A + \alpha x \bar{y}^T$.

`CGERU` Compute the rank-one update of a complex general matrix:
$A \leftarrow A + \alpha x y^T$.

`CHER` Compute the rank-one update of an Hermitian matrix:
$A \leftarrow A + \alpha x \bar{x}^T$ with x complex and α real.

`CHER2` Compute a rank-two update of an Hermitian matrix:
$A \leftarrow A + \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T$.

`CHER2K` Compute one of the Hermitian rank $2k$ operations:
$C \leftarrow \alpha A \bar{B}^T + \bar{\alpha} B \bar{A}^T + \beta C$ or $C \leftarrow \alpha \bar{A}^T B + \bar{\alpha} \bar{B}^T A + \beta C$
,
where C is an n by n Hermitian matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

`CHERK` Compute one of the Hermitian rank k operations:
$C \leftarrow \alpha A \bar{A}^T + \beta C$ or $C \leftarrow \alpha \bar{A}^T A + \beta C$,
where C is an n by n Hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

`CSYR2K` Compute one of the symmetric rank $2k$ operations:
$C \leftarrow \alpha A B^T + \alpha B A^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
,
where C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

`CSYRK` Compute one of the symmetric rank k operations:
$C \leftarrow \alpha A A^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$,
where C is an n by n symmetric matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

`CTBSV` Solve one of the complex triangular systems:
$x \leftarrow A^{-1} x,\ x \leftarrow \left(A^{-1}\right)^T x,$ or $x \leftarrow \left(\bar{A}^T\right)^{-1} x$,
where A is a triangular matrix in band storage mode.

`CTRSM` Solve one of the complex matrix equations:
$B \leftarrow \alpha A^{-1} B,\ B \leftarrow \alpha B A^{-1},\ B \leftarrow \alpha \left(A^{-1}\right)^T B,\ B \leftarrow \alpha B \left(A^{-1}\right)^T,$

$B \leftarrow \alpha \left(\bar{A}^T\right)^{-1} B,$ or $B \leftarrow \alpha B \left(\bar{A}^T\right)^{-1}$
where A is a triangular matrix.

`CTRSV` Solve one of the complex triangular systems:
$x \leftarrow A^{-1} x,\ x \leftarrow \left(A^{-1}\right)^T x,$ or $x \leftarrow \left(\bar{A}^T\right)^{-1} x$,
where A is a triangular matrix.

`HRRRR` Compute the Hadamard product of two real rectangular matrices.

`SGER` Compute the rank-one update of a real general matrix: $A \leftarrow A + \alpha xy^T$.

`SSYR` Compute the rank-one update of a real symmetric matrix: $A \leftarrow A + \alpha xx^T$.

`SSYR2` Compute the rank-two update of a real symmetric matrix: $A \leftarrow A + \alpha xy^T + \alpha yx^T$.

`SSYR2K` Compute one of the symmetric rank 2*k* operations: $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
,
where *C* is an *n* by *n* symmetric matrix and *A* and *B* are *n* by *k* matrices in the first case and *k* by *n* matrices in the second case.

`SSYRK` Compute one of the symmetric rank *k* operations: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$, where *C* is an *n* by *n* symmetric matrix and *A* is an *n* by *k* matrix in the first case and a *k* by *n* matrix in the second case.

`STBSV` Solve one of the triangular systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$, where *A* is a triangular matrix in band storage mode.

`STRSM` Solve one of the matrix equations: $B \leftarrow \alpha A^{-1}B$, $B \leftarrow \alpha BA^{-1}$, $B \leftarrow \alpha\left(A^{-1}\right)^T B$, or $B \leftarrow \alpha B\left(A^{-1}\right)^T$ where *B* is an *m* by *n* matrix and *A* is a triangular matrix.

`STRSV` Solve one of the triangular linear systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$, where *A* is a triangular matrix.

D1b2 .... Norm

`NR1CB` Compute the 1-norm of a complex band matrix in band storage mode.

`NR1RB` Compute the 1-norm of a real band matrix in band storage mode.

`NR1RR` Compute the 1-norm of a real matrix.

`NR2RR` Compute the Frobenius norm of a real rectangular matrix.

`NRIRR` Compute the infinity norm of a real matrix.

D1b3 .... Transpose

`TRNRR` Transpose a rectangular matrix.

D1b4 Multiplication by vector

`BLINF` Compute the bilinear form $x^T Ay$.

CGBMV Compute one of the matrix-vector operations:
$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y,$
where *A* is a matrix stored in band storage mode.

CGEMV Compute one of the matrix-vector operations:
$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y,$

CHBMV Compute the matrix-vector operation
$y \leftarrow \alpha Ax + \beta y,$
where *A* is an Hermitian band matrix in band Hermitian storage.

CHEMV Compute the matrix-vector operation
$y \leftarrow \alpha Ax + \beta y,$
where *A* is an Hermitian matrix.

CTBMV Compute one of the matrix-vector operations:
$x \leftarrow Ax, x \leftarrow A^T x,$ or $x \leftarrow \overline{A}^T x,$
where *A* is a triangular matrix in band storage mode.

CTRMV Compute one of the matrix-vector operations:
$x \leftarrow Ax, x \leftarrow A^T x,$ or $x \leftarrow \overline{A}^T x,$
where *A* is a triangular matrix.

MUCBV Multiply a complex band matrix in band storage mode by a complex vector.

MUCRV Multiply a complex rectangular matrix by a complex vector.

MURBV Multiply a real band matrix in band storage mode by a real vector.

MURRV Multiply a real rectangular matrix by a vector.

SGBMV Compute one of the matrix-vector operations:
$y \leftarrow \alpha Ax + \beta y,$ or $y \leftarrow \alpha A^T x + \beta y,$
where *A* is a matrix stored in band storage mode.

SGEMV Compute one of the matrix-vector operations:
$y \leftarrow \alpha Ax + \beta y,$ or $y \leftarrow \alpha A^T x + \beta y,$

SSBMV Compute the matrix-vector operation
$y \leftarrow \alpha Ax + \beta y,$
where *A* is a symmetric matrix in band symmetric storage mode.

SSYMV Compute the matrix-vector operation
$y \leftarrow \alpha Ax + \beta y,$
where *A* is a symmetric matrix.

STBMV Compute one of the matrix-vector operations:
$x \leftarrow Ax$ or $x \leftarrow A^T x,$
where *A* is a triangular matrix in band storage mode.

STRMV Compute one of the matrix-vector operations:
$x \leftarrow Ax$ or $x \leftarrow A^T x,$
where *A* is a triangular matrix.

D1b5 .... Addition, subtraction

ACBCB Add two complex band matrices, both in band storage mode.

ARBRB Add two band matrices, both in band storage mode.

D1b6 .... Multiplication

CGEMM Compute one of the matrix-matrix operations:

$$C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^T B + \beta C,\ C \leftarrow \alpha AB^T$$

$$+\beta C,\ C \leftarrow \alpha A^T B^T + \beta C,\ C \leftarrow \alpha A\overline{B}^T + \beta C,$$

$$\text{or } C \leftarrow \alpha \overline{A}^T B + \beta C,\ C \leftarrow \alpha A^T \overline{B}^T + \beta C,$$

$$C \leftarrow \alpha \overline{A}^T B^T + \beta C,\ \text{or } C \leftarrow \alpha \overline{A}^T \overline{B}^T + \beta C$$

CHEMM Compute one of the matrix-matrix operations:

$C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$,

where A is an Hermitian matrix and B and C are m by n matrices.

CSYMM Compute one of the matrix-matrix operations:

$C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$,

where A is a symmetric matrix and B and C are m by n matrices.

CTRMM Compute one of the matrix-matrix operations:

$$B \leftarrow \alpha AB,\ B \leftarrow \alpha A^T B,\ B \leftarrow \alpha BA,\ B \leftarrow \alpha BA^T,$$

$$B \leftarrow \alpha \overline{A}^T B, \text{or } B \leftarrow \alpha B\overline{A}^T$$

where B is an m by n matrix and A is a triangular matrix.

MCRCR Multiply two complex rectangular matrices, AB.

MRRRR Multiply two real rectangular matrices, AB.

MXTXF Compute the transpose product of a matrix, A^TA.

MXTYF Multiply the transpose of matrix A by matrix B, A^TB.

MXYTF Multiply a matrix A by the transpose of a matrix B, AB^T.

SGEMM Compute one of the matrix-matrix operations:

$$C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^T B + \beta C,\ C \leftarrow \alpha AB^T$$

$$+\beta C,\ \text{or } C \leftarrow \alpha A^T B^T + \beta C$$

.

SSYMM Compute one of the matrix-matrix operations:

$C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$,

where A is a symmetric matrix and B and C are m by n matrices.

STRMM Compute one of the matrix-matrix operations:

$B \leftarrow \alpha AB,\ B \leftarrow \alpha A^T B$ or $B \leftarrow \alpha BA,\ B \leftarrow \alpha BA^T$,

where B is an m by n matrix and A is a triangular matrix.

D1b7 .... Matrix polynomial

POLRG 1207 Evaluate a real general matrix polynomial.

D1b8 .... Copy

`CCBCB` Copy a complex band matrix stored in complex band storage mode.
`CCGCG` Copy a complex general matrix.
`CRBRB` Copy a real band matrix stored in band storage mode.
`CRGRG` Copy a real general matrix.

D1b9 .... Storage mode conversion

`CCBCG` Convert a complex matrix in band storage mode to a complex matrix in full storage mode.
`CCGCB` Convert a complex general matrix to a matrix in complex band storage mode.
`CHBCB` Copy a complex Hermitian band matrix stored in band Hermitian storage mode to a complex band matrix stored in band storage mode.
`CHFCG` Extend a complex Hermitian matrix defined in its upper triangle to its lower triangle.
`CRBCB` Convert a real matrix in band storage mode to a complex matrix in band storage mode.
`CRBRG` Convert a real matrix in band storage mode to a real general matrix.
`CRGCG` Copy a real general matrix to a complex general matrix.
`CRGRB` Convert a real general matrix to a matrix in band storage mode.
`CRRCR` Copy a real rectangular matrix to a complex rectangular matrix.
`CSBRB` Copy a real symmetric band matrix stored in band symmetric storage mode to a real band matrix stored in band storage mode.
`CSFRG` Extend a real symmetric matrix defined in its upper triangle to its lower triangle.

D1b10 .. Elementary rotation (Givens transformation) (*search also class D1a8*)

`SROTG` Construct a Givens plane rotation in single precision.
`SROTMG` Construct a modified Givens plane rotation in single precision.

D2 .... Solution of systems of linear equations (including inversion, *LU* and related decompositions)

D2a ...... Real nonsymmetric matrices

`LSLTO` Solve a real Toeplitz linear system.

D2a1 .... General

`LFCRG` Compute the *LU* factorization of a real general matrix and estimate its L_1 condition number.
`LFIRG` Use iterative refinement to improve the solution of a real general system of linear equations.
`LFSRG` Solve a real general system of linear equations given the *LU* factorization of the coefficient matrix.
`LFTRG` Compute the *LU* factorization of a real general matrix.

`LINRG` Compute the inverse of a real general matrix.

`LSARG` Solve a real general system of linear equations with iterative refinement.

`LSLRG` Solve a real general system of linear equations without iterative refinement.

D2a2 .... Banded

`LFCRB` Compute the *LU* factorization of a real matrix in band storage mode and estimate its L_1 condition number.

`LFIRB` Use iterative refinement to improve the solution of a real system of linear equations in band storage mode.

`LFSRB` Solve a real system of linear equations given the *LU* factorization of the coefficient matrix in band storage mode.

`LFTRB` Compute the *LU* factorization of a real matrix in band storage mode.

`LSARB` Solve a real system of linear equations in band storage mode with iterative refinement.

`LSLRB` Solve a real system of linear equations in band storage mode without iterative refinement.

`STBSV` Solve one of the triangular systems:

$$x \leftarrow A^{-1}x \text{ or } x \leftarrow \left(A^{-1}\right)^T x,$$

where *A* is a triangular matrix in band storage mode.

D2a2a .. Tridiagonal

`LSLCR` Compute the *LDU* factorization of a real tridiagonal matrix *A* using a cyclic reduction algorithm.

`LSLTR` Solve a real tridiagonal system of linear equations.

D2a3 .... Triangular

`LFCRT` Estimate the condition number of a real triangular matrix.

`LINRT` Compute the inverse of a real triangular matrix.

`LSLRT` Solve a real triangular system of linear equations.

`STRSM` Solve one of the matrix equations:

$$B \leftarrow \alpha A^{-1}B,\ B \leftarrow \alpha BA^{-1},\ B \leftarrow \alpha\left(A^{-1}\right)^T B,$$

$$\text{or } B \leftarrow \alpha B\left(A^{-1}\right)^T$$

where *B* is an *m* by *n* matrix and *A* is a triangular matrix.

`STRSV` Solve one of the triangular linear systems:

$$x \leftarrow A^{-1}x \text{ or } x \leftarrow \left(A^{-1}\right)^T x$$

where *A* is a triangular matrix.

D2a4 .... Sparse

`LFSXG` Solve a sparse system of linear equations given the *LU* factorization of the coefficient matrix.

LFTXG Compute the LU factorization of a real general sparse matrix.

LSLXG Solve a sparse system of linear algebraic equations by Gaussian elimination.

GMRES Use restarted GMRES with reverse communication to generate an approximate solution of $Ax = b$.

D2b ...... Real symmetric matrices

D2b1 .... General

D2b1a. . Indefinite

LCHRG Compute the Cholesky decomposition of a symmetric positive semidefinite matrix with optional column pivoting.

LFCSF Compute the $U\,DU^T$ factorization of a real symmetric matrix and estimate its L_1 condition number.

LFISF Use iterative refinement to improve the solution of a real symmetric system of linear equations.

LFSSF Solve a real symmetric system of linear equations given the $U\,DU^T$ factorization of the coefficient matrix.

LFTSF Compute the $U\,DU^T$ factorization of a real symmetric matrix.

LSASF Solve a real symmetric system of linear equations with iterative refinement.

LSLSF Solve a real symmetric system of linear equations without iterative refinement.

D2b1b. . Positive definite

LCHRG Compute the Cholesky decomposition of a symmetric positive semidefinite matrix with optional column pivoting.

LFCDS Compute the $R^T\,R$ Cholesky factorization of a real symmetric positive definite matrix and estimate its L_1 condition number.

LFIDS Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations.

LFSDS Solve a real symmetric positive definite system of linear equations given the $R^T\,R$ Choleksy factorization of the coefficient matrix.

LFTDS Compute the $R^T\,R$ Cholesky factorization of a real symmetric positive definite matrix.

LINDS Compute the inverse of a real symmetric positive definite matrix.

LSADS Solve a real symmetric positive definite system of linear equations with iterative refinement.

LSLDS Solve a real symmetric positive definite system of linear equations without iterative refinement.

D2b2 .... Positive definite banded

LFCQS Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode and estimate its L_1 condition number.

LFDQS Compute the determinant of a real symmetric positive definite matrix given the $R^T R$ Cholesky factorization of the band symmetric storage mode.

LFIQS Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations in band symmetric storage mode.

LFSQS Solve a real symmetric positive definite system of linear equations given the factorization of the coefficient matrix in band symmetric storage mode.

LFTQS Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode.

LSAQS Solve a real symmetric positive definite system of linear equations in band symmetric storage mode with iterative refinement.

LSLPB Compute the $R^T DR$ Cholesky factorization of a real symmetric positive definite matrix A in codiagonal band symmetric storage mode. Solve a system $Ax = b$.

LSLQS Solve a real symmetric positive definite system of linear equations in band symmetric storage mode without iterative refinement.

D2b4 .... Sparse

JCGRC Solve a real symmetric definite linear system using the Jacobi preconditioned conjugate gradient method with reverse communication.

LFSXD Solve a real sparse symmetric positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.

LNFXD Compute the numerical Cholesky factorization of a sparse symmetrical matrix A.

LSCXD Perform the symbolic Cholesky factorization for a sparse symmetric matrix using a minimum degree ordering or a userspecified ordering, and set up the data structure for the numerical Cholesky factorization.

LSLXD Solve a sparse system of symmetric positive definite linear algebraic equations by Gaussian elimination.

PCGRC Solve a real symmetric definite linear system using a preconditioned conjugate gradient method with reverse communication.

D2c. ..... Complex non-Hermitian matrices

LSLCC Solve a complex circulant linear system.

LSLTC Solve a complex Toeplitz linear system.

D2c1 .... General

LFCCG Compute the *LU* factorization of a complex general matrix and estimate its L_1 condition number.

LFICG Use iterative refinement to improve the solution of a complex general system of linear equations.

LFSCG Solve a complex general system of linear equations given the *LU* factorization of the coefficient matrix.

LFTCG Compute the *LU* factorization of a complex general matrix.

LINCG Compute the inverse of a complex general matrix.

LSACG Solve a complex general system of linear equations with iterative refinement.

LSLCG Solve a complex general system of linear equations without iterative refinement.

D2c2 .... Banded

CTBSV Solve one of the complex triangular systems:

$$x \leftarrow A^{-1}x\,,\; x \leftarrow \left(A^{-1}\right)^{T} x,\ \text{or}\ x \leftarrow \left(\overline{A}^{T}\right)^{-1} x\,,$$

where *A* is a triangular matrix in band storage mode.

LFCCB Compute the *LU* factorization of a complex matrix in band storage mode and estimate its L_1 condition number.

LFICB Use iterative refinement to improve the solution of a complex system of linear equations in band storage mode.

LFSCB Solve a complex system of linear equations given the *LU* factorization of the coefficient matrix in band storage mode.

LFTCB Compute the *LU* factorization of a complex matrix in band storage mode.

LSACB Solve a complex system of linear equations in band storage mode with iterative refinement.

LSLCB Solve a complex system of linear equations in band storage mode without iterative refinement.

D2c2a... Tridiagonal

LSLCQ Compute the *LDU* factorization of a complex tridiagonal matrix *A* using a cyclic reduction algorithm.

LSLTQ Solve a complex tridiagonal system of linear equations.

D2c3 .... Triangular

CTRSM Solve one of the complex matrix equations:

$$B \leftarrow \alpha A^{-1}B,\ B \leftarrow \alpha B A^{-1},\ B \leftarrow \alpha\left(A^{-1}\right)^{T} B,\ B \leftarrow \alpha B\left(A^{-1}\right)^{T},$$

$$B \leftarrow \alpha\left(\overline{A}^{T}\right)^{-1} B,\ \text{or}\ B \leftarrow \alpha B\left(\overline{A}^{T}\right)^{-1}$$

where *A* is a traiangular matrix.

CTRSV Solve one of the complex triangular systems:

$$x \leftarrow A^{-1}x\,,\; x \leftarrow \left(A^{-1}\right)^{T} x,\ \text{or}\ x \leftarrow \left(\overline{A}^{T}\right)^{-1} x$$

where *A* is a triangular matrix.

`LFCCT` Estimate the condition number of a complex triangular matrix.

`LINCT` Compute the inverse of a complex triangular matrix.

`LSLCT` Solve a complex triangular system of linear equations.

D2c4 .... Sparse

`LFSZG` Solve a complex sparse system of linear equations given the *LU* factorization of the coefficient matrix.

`LFTZG` Compute the *LU* factorization of a complex general sparse matrix.

`LSLZG` Solve a complex sparse system of linear equations by Gaussian elimination.

D2d. ..... Complex Hermitian matrices

D2d1 .... General

D2d1a. . Indefinite

`LFCHF` Compute the $U DU^H$ factorization of a complex Hermitian matrix and estimate its L_1 condition number.

`LFDHF` Compute the determinant of a complex Hermitian matrix given the $U DU^H$ factorization of the matrix.

`LFIHF` Use iterative refinement to improve the solution of a complex Hermitian system of linear equations.

`LFSHF` Solve a complex Hermitian system of linear equations given the $U DU^H$ factorization of the coefficient matrix.

`LFTHF` Compute the $U DU^H$ factorization of a complex Hermitian matrix.

`LSAHF` Solve a complex Hermitian system of linear equations with iterative refinement.

`LSLHF` Solve a complex Hermitian system of linear equations without iterative refinement.

D2d1b. . Positive definite

`LFCDH` Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix and estimate its L_1 condition number.

`LFIDH` Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations.

`LFSDH` Solve a complex Hermitian positive definite system of linear equations given the $R^H R$ factorization of the coefficient matrix.

`LFTDH` Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix.

`LSADH` Solve a Hermitian positive definite system of linear equations with iterative refinement.

`LSLDH` Solve a complex Hermitian positive definite system of linear equations without iterative refinement.

D2d2 .... Positive definite banded

`LFCQH` Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode and estimate its L_1 condition number.

`LFIQH` Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations in band Hermitian storage mode.

`LFSQH` Solve a complex Hermitian positive definite system of linear equations given the factorization of the coefficient matrix in band Hermitian storage mode.

`LFTQH` Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode.

`LSAQH` Solve a complex Hermitian positive definite system of linear equations in band Hermitian storage mode with iterative refinement.

`LSLQB` Compute the $R^H DR$ Cholesky factorization of a complex hermitian positive-definite matrix *A* in codiagonal band hermitian storage mode. Solve a system $Ax = b$.

`LSLQH` Solve a complex Hermitian positive definite system of linearequations in band Hermitian storage mode without iterative refinement.

D2d4 .... Sparse

`LFSZD` Solve a complex sparse Hermitian positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.

`LNFZD` Compute the numerical Cholesky factorization of a sparse Hermitian matrix *A*.

`LSLZD` Solve a complex sparse Hermitian positive definite system of linear equations by Gaussian elimination.

D3 .... Determinants

D3a. ..... Real nonsymmetric matrices

D3a1 .... General

`LFDRG` Compute the determinant of a real general matrix given the *LU* factorization of the matrix.

D3a2 .... Banded

`LFDRB` Compute the determinant of a real matrix in band storage mode given the *LU* factorization of the matrix.

D3a3 .... Triangular

`LFDRT` Compute the determinant of a real triangular matrix.

D3b. ..... Real symmetric matrices

D3b1 .... General

D3b1a. . Indefinite

`LFDSF` Compute the determinant of a real symmetric matrix given the $U\,DU^T$ factorization of the matrix.

D3b1b. . Positive definite

`LFDDS` Compute the determinant of a real symmetric positive definite matrix given the $R^H\,R$ Cholesky factorization of the matrix.

D3c. ..... Complex non-Hermitian matrices

D3c1 .... General

`LFDCG` Compute the determinant of a complex general matrix given the *LU* factorization of the matrix.

D3c2 .... Banded

`LFDCB` Compute the determinant of a complex matrix given the *LU* factorization of the matrix in band storage mode.

D3c3 .... Triangular

`LFDCT` Compute the determinant of a complex triangular matrix.

D3d. ..... Complex Hermitian matrices

D3d1 .... General

D3d1b. . Positive definite

`LFDDH` Compute the determinant of a complex Hermitian positive definite matrix given the $R^H\,R$ Cholesky factorization of the matrix.

D3d2 .... Positive definite banded

`LFDQH` Compute the determinant of a complex Hermitian positive definite matrix given the $R^H\,R$ Cholesky factorization in band Hermitian storage mode.

D4 .... Eigenvalues, eigenvectors

D4a. ..... Ordinary eigenvalue problems ($Ax = \lambda x$)

D4a1 .... Real symmetric

`EVASF` Compute the largest or smallest eigenvalues of a real symmetric matrix.

`EVBSF` Compute selected eigenvalues of a real symmetric matrix.

`EVCSF` Compute all of the eigenvalues and eigenvectors of a real symmetric matrix.

EVESF Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix.

EVFSF Compute selected eigenvalues and eigenvectors of a real symmetric matrix.

EVLSF Compute all of the eigenvalues of a real symmetric matrix.

D4a2 .... Real nonsymmetric

EVCRG Compute all of the eigenvalues and eigenvectors of a real matrix.

EVLRG Compute all of the eigenvalues of a real matrix.

D4a3 .... Complex Hermitian

EVAHF Compute the largest or smallest eigenvalues of a complex Hermitian matrix.

EVBHF Compute the eigenvalues in a given range of a complex Hermitian matrix.

EVCHF Compute all of the eigenvalues and eigenvectors of a complex Hermitian matrix.

EVEHF Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix.

EVFHF Compute the eigenvalues in a given range and the corresponding eigenvectors of a complex Hermitian matrix.

EVLHF Compute all of the eigenvalues of a complex Hermitian matrix.

D4a4 .... Complex non-Hermitian

EVCCG Compute all of the eigenvalues and eigenvectors of a complex matrix.

EVLCG Compute all of the eigenvalues of a complex matrix.

D4a6 .... Banded

EVASB Compute the largest or smallest eigenvalues of a real symmetric matrix in band symmetric storage mode.

EVBSB Compute the eigenvalues in a given interval of a real symmetric matrix stored in band symmetric storage mode.

EVCSB Compute all of the eigenvalues and eigenvectors of a real symmetric matrix in band symmetric storage mode.

EVESB Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix in band symmetric storage mode.

EVFSB Compute the eigenvalues in a given interval and the corresponding eigenvectors of a real symmetric matrix stored in band symmetric storage mode.

EVLSB Compute all of the eigenvalues of a real symmetric matrix in band symmetric storage mode.

D4b. ..... Generalized eigenvalue problems (e.g., $Ax = \lambda Bx$)

D4b1 .... Real symmetric

`GVCSP`	Compute all of the eigenvalues and eigenvectors of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.
`GVLSP`	Compute all of the eigenvalues of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.

D4b2 .... Real general

`GVCRG`	Compute all of the eigenvalues and eigenvectors of a generalized real eigensystem $Az = \lambda Bz$.
`GVLRG`	Compute all of the eigenvalues of a generalized real eigensystem $Az = \lambda Bz$.

D4b4 .... Complex general

`GVCCG`	Compute all of the eigenvalues and eigenvectors of a generalized complex eigensystem $Az = \lambda Bz$.
`GVLCG`	Compute all of the eigenvalues of a generalized complex eigensystem $Az = \lambda Bz$.

D4c. ..... Associated operations

`EPICG`	Compute the performance index for a complex eigensystem.
`EPIHF`	Compute the performance index for a complex Hermitian eigensystem.
`EPIRG`	Compute the performance index for a real eigensystem.
`EPISB`	Compute the performance index for a real symmetric eigensystem in band symmetric storage mode.
`EPISF`	Compute the performance index for a real symmetric eigensystem.
`GPICG`	Compute the performance index for a generalized complex eigensystem $Az = \lambda Bz$.
`GPIRG`	Compute the performance index for a generalized real eigensystem $Az = \lambda Bz$.
`GPISP`	Compute the performance index for a generalized real symmetric eigensystem problem.

D4c2 .... Compute eigenvalues of matrix in compact form

D4c2b. . Hessenberg

`EVCCH`	Compute all of the eigenvalues and eigenvectors of a complex upper Hessenberg matrix.
`EVCRH`	Compute all of the eigenvalues and eigenvectors of a real upper Hessenberg matrix.
`EVLCH`	Compute all of the eigenvalues of a complex upper Hessenberg matrix.
`EVLRH`	Compute all of the eigenvalues of a real upper Hessenberg matrix.

D5 .... *QR* decomposition, Gram-Schmidt orthogonalization

`LQERR` Accumulate the orthogonal matrix Q from its factored form given the QR factorization of a rectangular matrix A.

`LQRRR` Compute the QR decomposition, $AP = QR$, using Householder transformations.

`LQRSL` Compute the coordinate transformation, projection, and complete the solution of the least-squares problem $Ax = b$.

`LSBRR` Solve a linear least-squares problem with iterative refinement.

`LSQRR` Solve a linear least-squares problem without iterative refinement.

D6 .... Singular value decomposition

`LSVCR` Compute the singular value decomposition of a complex matrix.

`LSVRR` Compute the singular value decomposition of a real matrix.

D7 .... Update matrix decompositions

D7b. ..... Cholesky

`LDNCH` Downdate the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is removed.

`LUPCH` Update the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is added.

D7c. ..... *QR*

`LUPQR` Compute an updated QR factorization after the rank-one matrix αxy^T is added.

D9 .... Singular, overdetermined or underdetermined systems of linear equations, generalized inverses

D9a. ..... Unconstrained

D9a1 .... Least squares (L_2) solution

`LQRRR` Compute the QR decomposition, $AP = QR$, using Householder transformations.

`LQRRV` Compute the least-squares solution using Householder transformations applied in blocked form.

`LQRSL` Compute the coordinate transformation, projection, and complete the solution of the least-squares problem $Ax = b$.

`LSBRR` Solve a linear least-squares problem with iterative refinement.

`LSQRR` Solve a linear least-squares problem without iterative refinement.

D9b. ..... Constrained

D9b1 .... Least squares (L_2) solution

`LCLSQ` Solve a linear least-squares problem with linear constraints.

D9c. ..... Generalized inverses

`LSGRR` Compute the generalized inverse of a real matrix.

E .... INTERPOLATION

E1 .... Univariate data (curve fitting)

E1a....... Polynomial splines (piecewise polynomials)

`BSINT` Compute the spline interpolant, returning the B-spline coefficients.

`CSAKM` Compute the Akima cubic spline interpolant.

`CSCON` Compute a cubic spline interpolant that is consistent with the concavity of the data.

`CSDEC` Compute the cubic spline interpolant with specified derivative endpoint conditions.

`CSHER` Compute the Hermite cubic spline interpolant.

`CSIEZ` Compute the cubic spline interpolant with the ‘not-a-knot’ condition and return values of the interpolant at specified points.

`CSINT` Compute the cubic spline interpolant with the ‘not-a-knot’ condition.

`CSPER` Compute the cubic spline interpolant with periodic boundary conditions.

`QDVAL` Evaluate a function defined on a set of points using quadratic interpolation.

`SPLEZ` Compute the values of a spline that either interpolates or fits user-supplied data.

E2 .... Multivariate data (surface fitting)

E2a....... Gridded

`BS2IN` Compute a two-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.

`BS3IN` Compute a three-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.

`QD2DR` Evaluate the derivative of a function defined on a rectangular grid using quadratic interpolation.

`QD2VL` Evaluate a function defined on a rectangular grid using quadratic interpolation.

`QD3DR` Evaluate the derivative of a function defined on a rectangular three-dimensional grid using quadratic interpolation.

`QD3VL` Evaluate a function defined on a rectangular three-dimensional grid using quadratic interpolation.

E2b ...... Scattered

`SURF` Compute a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.

E3......... Service routines for interpolation

E3a....... Evaluation of fitted functions, including quadrature

E3a1..... Function evaluation

`BS1GD` Evaluate the derivative of a spline on a grid, given its B-spline representation.

`BS2DR` Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation.

`BS2GD` Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

`BS2VL` Evaluate a two-dimensional tensor-product spline, given its tensor-product B-spline representation.

`BS3GD` Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

`BS3VL` Evaluate a three-dimensional tensor-product spline, given its tensor-product B-spline representation.

`BSVAL` Evaluate a spline, given its B-spline representation.

`CSVAL` Evaluate a cubic spline.

`PPVAL` Evaluate a piecewise polynomial.

`QDDER` Evaluate the derivative of a function defined on a set of points using quadratic interpolation.

E3a2..... Derivative evaluation

`BS1GD` Evaluate the derivative of a spline on a grid, given its B-spline representation.

`BS2DR` Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation.

`BS2GD` Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

`BS3DR` Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation.

`BS3GD` Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.

`BSDER` Evaluate the derivative of a spline, given its B-spline representation.

`CS1GD` Evaluate the derivative of a cubic spline on a grid.

`CSDER` Evaluate the derivative of a cubic spline.

`PP1GD` Evaluate the derivative of a piecewise polynomial on a grid.

PPDER Evaluate the derivative of a piecewise polynomial.

QDDER Evaluate the derivative of a function defined on a set of points using quadratic interpolation.

E3a3..... Quadrature

BS2IG Evaluate the integral of a tensor-product spline on a rectangular domain, given its tensor-product B-spline representation.

BS3IG Evaluate the integral of a tensor-product spline in three dimensions over a three-dimensional rectangle, given its tensorproduct B-spline representation.

BSITG Evaluate the integral of a spline, given its B-spline representation.

CSITG Evaluate the integral of a cubic spline.

E3b ...... Grid or knot generation

BSNAK Compute the ‘not-a-knot’ spline knot sequence.

BSOPK Compute the ‘optimal’ spline knot sequence.

E3c....... Manipulation of basis functions (e.g., evaluation, change of basis)

BSCPP Convert a spline in B-spline representation to piecewise polynomial representation.

F........... SOLUTION OF NONLINEAR EQUATIONS

F1......... Single equation

F1a....... Polynomial

F1a1..... Real coefficients

ZPLRC Find the zeros of a polynomial with real coefficients using Laguerre’s method.

ZPORC Find the zeros of a polynomial with real coefficients using the Jenkins-Traub three-stage algorithm.

F1a2..... Complex coefficients

ZPOCC Find the zeros of a polynomial with complex coefficients using the Jenkins-Traub three-stage algorithm.

F1b....... Nonpolynomial

ZANLY Find the zeros of a univariate complex function using Müller’s method.

ZBREN Find a zero of a real function that changes sign in a given interval.

ZREAL Find the real zeros of a real function using Müller’s method.

F2......... System of equations

NEQBF Solve a system of nonlinear equations using factored secant update with a finite-difference approximation to the Jacobian.

NEQBJ Solve a system of nonlinear equations using factored secant update with a user-supplied Jacobian.

NEQNF Solve a system of nonlinear equations using a modified Powell hybrid algorithm and a finite-difference approximation to the Jacobian.

NEQNJ Solve a system of nonlinear equations using a modified Powell hybrid algorithm with a user-supplied Jacobian.

G .... OPTIMIZATION (*search also classes K, L8*)

G1 .... Unconstrained

G1a. ..... Univariate

G1a1 .... Smooth function

G1a1a... User provides no derivatives

UVMIF Find the minimum point of a smooth function of a single variable using only function evaluations.

G1a1b .. User provides first derivatives

UVMID Find the minimum point of a smooth function of a single variable using both function evaluations and first derivative evaluations.

G1a2 .... General function (no smoothness assumed)

UVMGS Find the minimum point of a nonsmooth function of a single variable.

G1b ...... Multivariate

G1b1 .... Smooth function

G1b1a. . User provides no derivatives

UMCGF Minimize a function of N variables using a conjugate gradient algorithm and a finite-difference gradient.

UMINF Minimize a function of N variables using a quasi-New method and a finite-difference gradient.

UNLSF Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

G1b1b. . User provides first derivatives

UMCGG Minimize a function of N variables using a conjugate gradient algorithm and a user-supplied gradient.

UMIDH Minimize a function of N variables using a modified Newton method and a finite-difference Hessian.

UMING Minimize a function of N variables using a quasi-New method and a user-supplied gradient.

UNLSJ Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

G1b1c. . User provides first and second derivatives

UMIAH Minimize a function of N variables using a modified Newton method and a user-supplied Hessian.

G1b2 .... General function (no smoothness assumed)

UMPOL Minimize a function of N variables using a direct search polytope algorithm.

G2 .... Constrained

G2a. ..... Linear programming

G2a1 .... Dense matrix of constraints

DLPRS Solve a linear programming problem via the revised simplex algorithm.

G2a2 .... Sparse matrix of constraints

SLPRS Solve a sparse linear programming problem via the revised simplex algorithm.

G2e. ..... Quadratic programming

G2e1 .... Positive definite Hessian (i.e., convex problem)

QPROG Solve a quadratic programming problem subject to linear equality/inequality constraints.

G2h. ..... General nonlinear programming

G2h1 .... Simple bounds

G2h1a. . Smooth function

G2h1a1 User provides no derivatives

BCLSF Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

BCONF Minimize a function of N variables subject to bounds the variables using a quasi-Newton method and a finite-difference gradient.

G2h1a2 User provides first derivatives

BCLSJ Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

BCODH Minimize a function of N variables subject to bounds the variables using a modified Newton method and a finite-difference Hessian.

BCONG Minimize a function of N variables subject to bounds the variables using a quasi-Newton method and a user-supplied gradient.

G2h1a3 User provides first and second derivatives

BCOAH Minimize a function of N variables subject to bounds the variables using a modified Newton method and a user-supplied Hessian.

G2h1b .. General function (no smoothness assumed)

`BCPOL`	Minimize a function of `N` variables subject to bounds the variables using a direct search complex algorithm.

G2h2 .... Linear equality or inequality constraints

G2h2a. . Smooth function

G2h2a1 User provides no derivatives

`LCONF`	Minimize a general objective function subject to linear equality/inequality constraints.

G2h2a2 User provides first derivatives

`LCONG`	Minimize a general objective function subject to linear equality/inequality constraints.

G2h3 .... Nonlinear constraints

G2h3b .. Equality and inequality constraints

G2h3b1 Smooth function and constraints

G2h3b1a. User provides no derivatives

`NCONF`	Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a finite difference gradient.

G2h3b1b User provides first derivatives of function and constraints

`NCONG`	Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a user-supplied gradient.

G4 .... Service routines

G4c ...... Check user-supplied derivatives

`CHGRD`	Check a user-supplied gradient of a function.
`CHHES`	Check a user-supplied Hessian of an analytic function.
`CHJAC`	Check a user-supplied Jacobian of a system of equations with `M` functions in `N` unknowns.

G4d ...... Find feasible point

`GGUES`	Generate points in an `N`-dimensional space.

G4f .... Other

`CDGRD`	Approximate the gradient using central differences.
`FDGRD`	Approximate the gradient using forward differences.
`FDHES`	Approximate the Hessian using forward differences and function values.
`FDJAC`	Approximate the Jacobian of `M` functions in `N` unknowns using forward differences.
`GDHES`	Approximate the Hessian using forward differences and a user-supplied gradient.

H .... DIFFERENTIATION, INTEGRATION

H1 .... Numerical differentiation

DERIV Compute the first, second or third derivative of a user-supplied function.

H2 .... Quadrature (numerical evaluation of definite integrals)

H2a. ..... One-dimensional integrals

H2a1 .... Finite interval (general integrand)

H2a1a .. Integrand available via user-defined procedure

H2a1a1. Automatic (user need only specify required accuracy)
QDAG Integrate a function using a globally adaptive scheme based on Gauss-Kronrod rules.
QDAGS Integrate a function (which may have endpoint singularities).
QDNG Integrate a smooth function using a nonadaptive rule.

H2a2 .... Finite interval (specific or special type integrand including weight functions, oscillating and singular integrands, principal value integrals, splines, etc.)

H2a2a .. Integrand available via user-defined procedure

H2a2a1 Automatic (user need only specify required accuracy)
QDAGP Integrate a function with singularity points given.
QDAWC Integrate a function F(X)/(X – C) in the Cauchy principal value sense.
QDAWO Integrate a function containing a sine or a cosine.
QDAWS Integrate a function with algebraic-logarithmic singularities.

H2a2b .. Integrand available only on grid

H2a2b1 Automatic (user need only specify required accuracy)
BSITG Evaluate the integral of a spline, given its B-spline representation.

H2a3 .... Semi-infinite interval (including e^{-x} weight function)

H2a3a. . Integrand available via user-defined procedure

H2a3a1. Automatic (user need only specify required accuracy)
QDAGI Integrate a function over an infinite or semi-infinite interval.
QDAWF Compute a Fourier integral.

H2b. ..... Multidimensional integrals

H2b1 .... One or more hyper-rectangular regions (including iterated integrals)

H2b1a. . Integrand available via user-defined procedure

H2b1a1 Automatic (user need only specify required accuracy)
QAND Integrate a function on a hyper-rectangle.
TWODQ Compute a two-dimensional iterated integral.

H2b1b .. Integrand available only on grid

H2b1b2 Nonautomatic

`BS2IG`	Evaluate the integral of a tensor-product spline on a rectangular domain, given its tensor-product B-spline representation.
`BS3IG`	Evaluate the integral of a tensor-product spline in three dimensions over a three-dimensional rectangle, given its tensorproduct B-spline representation.

H2c. ..... Service routines (compute weight and nodes for quadrature formulas)

`FQRUL`	Compute a Fejér quadrature rule with various classical weight functions.
`GQRCF`	Compute a Gauss, Gauss-Radau or Gauss-Lobatto quadrature rule given the recurrence coefficients for the monic polynomials orthogonal with respect to the weight function.
`GQRUL`	Compute a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.
`RECCF`	Compute recurrence coefficients for various monic polynomials.
`RECQR`	Compute recurrence coefficients for monic polynomials given a quadrature rule.

I............ DIFFERENTIAL AND INTEGRAL EQUATIONS

I1.......... Ordinary differential equations (ODE’s)

I1a. .... Initial value problems

I1a1...... General, nonstiff or mildly stiff

I1a1a. ... One-step methods (e.g., Runge-Kutta)

`IVMRK`	Solve an initial-value problem $y' = f(t, y)$ for ordinary differential equations using Runge-Kutta pairs of various orders.
`IVPRK`	Solve an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.

I1a1b. ... Multistep methods (e.g., Adams predictor-corrector)

`IVPAG`	Solve an initial-value problem for ordinary differential equations using either Adams-Moulton’s or Gear’s BDF method.

I1a2...... Stiff and mixed algebraic-differential equations

`DASPG`	Solve a first order differential-algebraic system of equations, $g(t, y, y') = 0$, using Petzold−Gear BDF method.

I1b........ Multipoint boundary value problems

I1b2...... Nonlinear

`BVPFD` Solve a (parameterized) system of differential equations with boundary conditions at two points, using a variable order, variable step size finite-difference method with deferred corrections.

`BVPMS` Solve a (parameterized) system of differential equations with boundary conditions at two points, using a multiple-shooting method.

I1b3...... Eigenvalue (e.g., Sturm-Liouville)

`SLCNT` Calculate the indices of eigenvalues of a Sturm-Liouville problem with boundary conditions (at regular points) in a specified subinterval of the real line, [α, β].

`SLEIG` Determine eigenvalues, eigenfunctions and/or spectral density functions for Sturm-Liouville problems in the form with boundary conditions (at regular points).

I2.......... Partial differential equations

I2a........ Initial boundary value problems

I2a1...... Parabolic

I2a1a.... One spatial dimension

`MOLCH` Solve a system of partial differential equations of the form $u_t = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.

I2b........ Elliptic boundary value problems

I2b1...... Linear

I2b1a.... Second order

I2b1a1.. Poisson (Laplace) or Helmholtz equation

I2b1a1a Rectangular domain (or topologically rectangular in the coordinate system)

`FPS2H` Solve Poisson’s or Helmholtz’s equation on a two-dimensional rectangle using a fast Poisson solver based on the `HODIE` finite-difference scheme on a uni mesh.

`FPS3H` Solve Poisson’s or Helmholtz’s equation on a three-dimensional box using a fast Poisson solver based on the `HODIE` finite-difference scheme on a uniform mesh.

J .... INTEGRAL TRANSFORMS

J1 .... Trigonometric transforms including fast Fourier transforms

J1a........ One-dimensional

J1a1...... Real

`FFTRB` Compute the real periodic sequence from its Fourier coefficients.

`FFTRF` Compute the Fourier coefficients of a real periodic sequence.

`FFTRI` Compute parameters needed by `FFTRF` and `FFTRB`.

J1a2...... Complex

`FFTCB` Compute the complex periodic sequence from its Fourier coefficients.

`FFTCF` Compute the Fourier coefficients of a complex periodic sequence.

`FFTCI` Compute parameters needed by `FFTCF` and `FFTCB`.

J1a3...... Sine and cosine transforms

`FCOSI` Compute parameters needed by `FCOST`.

`FCOST` Compute the discrete Fourier cosine transformation of an even sequence.

`FSINI` Compute parameters needed by `FSINT`.

`FSINT` Compute the discrete Fourier sine transformation of an odd sequence.

`QCOSB` Compute a sequence from its cosine Fourier coefficients with only odd wave numbers.

`QCOSF` Compute the coefficients of the cosine Fourier transform with only odd wave numbers.

`QCOSI` Compute parameters needed by `QCOSF` and `QCOSB`.

`QSINB` Compute a sequence from its sine Fourier coefficients with only odd wave numbers.

`QSINF` Compute the coefficients of the sine Fourier transform with only odd wave numbers.

`QSINI` Compute parameters needed by `QSINF` and `QSINB`.

J1b .... Multidimensional

`FFT2B` Compute the inverse Fourier transform of a complex periodic two-dimensional array.

`FFT2D` Compute Fourier coefficients of a complex periodic two-dimensional array.

`FFT3B` Compute the inverse Fourier transform of a complex periodic three-dimensional array.

`FFT3F` Compute Fourier coefficients of a complex periodic threedimensional array.

J2 .... Convolutions

`CCONV` Compute the convolution of two complex vectors.

`RCONV` Compute the convolution of two real vectors.

J3 .... Laplace transforms

`INLAP` Compute the inverse Laplace transform of a complex function.

`SINLP` Compute the inverse Laplace transform of a complex function.

K .... APPROXIMATION (*search also class L8*)

K1 .... Least squares (L_2) approximation

K1a. ..... Linear least squares (*search also classes D5, D6, D9*)

K1a1 .... Unconstrained

K1a1a. . Univariate data (curve fitting)

K1a1a1 Polynomial splines (piecewise polynomials)

`BSLSQ` Compute the least-squares spline approximation, and return the B-spline coefficients.

`BSVLS` Compute the variable knot B-spline least squares approximation to given data.

`CONFT` Compute the least-squares constrained spline approximation, returning the B-spline coefficients.

K1a1a2 Polynomials

`RCURV` Fit a polynomial curve using least squares.

K1a1a3 Other functions (e.g., trigonometric, user-specified)

`FNLSQ` Compute a least-squares approximation with user-supplied basis functions.

K1a1b .. Multivariate data (surface fitting)

`BSLS2` Compute a two-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.

`BSLS3` Compute a three-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.

K1a2 .... Constrained

K1a2a .. Linear constraints

`LCLSQ` Solve a linear least-squares problem with linear constraints.

K1b...... Nonlinear least squares

K1b1 .... Unconstrained

K1b1a .. Smooth functions

K1b1a1 User provides no derivatives

`UNLSF` Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.

K1b1a2 User provides first derivatives

`UNLSJ` Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.

K1b2 .... Constrained

K1b2a .. Linear constraints

`BCLSF`	Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.
`BCLSJ`	Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.
`BCNLS`	Solve a nonlinear least-squares problem subject to bounds on the variables and general linear constraints.

K2 .... Minimax (L_∞) approximation

`RATCH`	Compute a rational weighted Chebyshev approximation to a continuous function on an interval.

K5 .... Smoothing

`CSSCV`	Compute a smooth cubic spline approximation to noisy data using cross-validation to estimate the smoothing parameter.
`CSSED`	Smooth one-dimensional data by error detection.
`CSSMH`	Compute a smooth cubic spline approximation to noisy data.

K6 .... Service routines for approximation

K6a. ..... Evaluation of fitted functions, including quadrature

K6a1 .... Function evaluation

`BSVAL`	Evaluate a spline, given its B-spline representation.
`CSVAL`	Evaluate a cubic spline.
`PPVAL`	Evaluate a piecewise polynomial.

K6a2 .... Derivative evaluation

`BSDER`	Evaluate the derivative of a spline, given its B-spline representation.
`CS1GD`	Evaluate the derivative of a cubic spline on a grid.
`CSDER`	Evaluate the derivative of a cubic spline.
`PP1GD`	Evaluate the derivative of a piecewise polynomial on a grid.
`PPDER`	Evaluate the derivative of a piecewise polynomial.

K6a3 .... Quadrature

`CSITG`	Evaluate the integral of a cubic spline.
`PPITG`	Evaluate the integral of a piecewise polynomial.

K6c. ..... Manipulation of basis functions (e.g., evaluation, change of basis)

`BSCPP`	Convert a spline in B-spline representation to piecewise polynomial representation.

L........... STATISTICS, PROBABILITY

L1......... Data summarization

L1c. ...... Multi-dimensional data

L1c1 ..... Raw data

L1c1b... Covariance, correlation

`CCORL` Compute the correlation of two complex vectors.

`RCORL` Compute the correlation of two real vectors.

L3 .... Elementary statistical graphics (*search also class Q*)

L3e....... Multi-dimensional data

L3e3..... Scatter diagrams

L3e3a... Superimposed *Y* vs. *X*

`PLOTP` Print a plot of up to 10 sets of points.

L6 .... Random number generation

L6a....... Univariate

L6a21... Uniform (continuous, discrete), uniform order statistics

`RNUN` Generate pseudorandom numbers from a uniform (0, 1) distribution.

`RNUNF` Generate a pseudorandom number from a uniform (0, 1) distribution.

L6c....... Service routines (e.g., seed)

`RNGET` Retrieve the current value of the seed used in the IMSL random number generators.

`RNOPT` Select the uniform (0, 1) multiplicative congruential pseudorandom number generator.

`RNSET` Initialize a random seed for use in the IMSL random number generators.

L8 .... Regression (*search also classes D5, D6, D9, G, K*)

L8a....... Simple linear (e.g., $y = \beta_0 + \beta_1 x + \varepsilon$) (*search also class L8h*)

L8a1..... Ordinary least squares

`FNLSQ` Compute a least-squares approximation with user-supplied basis functions.

L8a1a... Parameter estimation

L8a1a1. Unweighted data

`RLINE` Fit a line to a set of data points using least squares.

L8b. ..... Polynomial (e.g., $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$) (*search also class L8c*)

L8b1 .... Ordinary least squares

L8b1b .. Parameter estimation

L8b1b2. Using orthogonal polynomials

`RCURV` Fit a polynomial curve using least squares.

L8c....... Multiple linear (e.g., $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k + \varepsilon$)

L8c1..... Ordinary least squares

L8c1b... Parameter estimation (*search also class L8c1a*)

L8c1b1. Using raw data

LSBRR Solve a linear least-squares problem with iterative refinement.

LSQRR Solve a linear least-squares problem without iterative refinement.

N .... DATA HANDLING

N1 .... Input, output

PGOPT Set or retrieve page width and length for printing.

WRCRL Print a complex rectangular matrix with a given format and labels.

WRCRN Print a complex rectangular matrix with integer row and column labels.

WRIRL Print an integer rectangular matrix with a given format and labels.

WRIRN Print an integer rectangular matrix with integer row and column labels.

WROPT Set or retrieve an option for printing a matrix.

WRRRL Print a real rectangular matrix with a given format and labels.

WRRRN Print a real rectangular matrix with integer row and column labels.

N3 .... Character manipulation

ACHAR Return a character given its ASCII value.

CVTSI Convert a character string containing an integer number into the corresponding integer form.

IACHAR Return the integer ASCII value of a character argument.

ICASE Return the ASCII value of a character converted to uppercase.

IICSR Compare two character strings using the ASCII collating sequence but without regard to case.

IIDEX Determine the position in a string at which a given character sequence begins without regard to case.

N4 .... Storage management (e.g., stacks, heaps, trees)

IWKCIN Initialize bookkeeping locations describing the character workspace stack.

IWKIN Initialize bookkeeping locations describing the workspace stack.

N5 .... Searching

N5b ...... Insertion position

ISRCH Search a sorted integer vector for a given integer and return its index.

SRCH Search a sorted vector for a given scalar and return its index.

SSRCH Search a character vector, sorted in ascending ASCII order, for a given string and return its index.

N5c ...... On a key

IIDEX Determine the position in a string at which a given character sequence begins without regard to case.

ISRCH Search a sorted integer vector for a given integer and return its index.

SRCH Search a sorted vector for a given scalar and return its index.

SSRCH Search a character vector, sorted in ascending ASCII order, for a given string and return its index.

N6 .... Sorting

N6a ...... Internal

N6a1 .... Passive (i.e., construct pointer array, rank)

N6a1a .. Integer

SVIBP Sort an integer array by nondecreasing absolute value and return the permutation that rearranges the array.

SVIGP Sort an integer array by algebraically increasing value and return the permutation that rearranges the array.

N6a1b .. Real

SVRBP Sort a real array by nondecreasing absolute value and return the permutation that rearranges the array.

SVRGP Sort a real array by algebraically increasing value and return the permutation that rearranges the array.

N6a2 .... Active

N6a2a .. Integer

SVIBN Sort an integer array by nondecreasing absolute value.

SVIBP Sort an integer array by nondecreasing absolute value and return the permutation that rearranges the array.

SVIGN Sort an integer array by algebraically increasing value.

SVIGP Sort an integer array by algebraically increasing value and return the permutation that rearranges the array.

N6a2b .. Real

SVRBN Sort a real array by nondecreasing absolute value.

SVRBP Sort a real array by nondecreasing absolute value and return the permutation that rearranges the array.

SVRGN Sort a real array by algebraically increasing value.

SVRGP Sort a real array by algebraically increasing value and return the permutation that rearranges the array.

N8 .... Permuting

PERMA Permute the rows or columns of a matrix.

PERMU Rearrange the elements of an array as specified by a permutation.

Q .... GRAPHICS (*search also classes L3*)

PLOTP Print a plot of up to 10 sets of points.

R .... SERVICE ROUTINES

IDYWK Compute the day of the week for a given date.

IUMAG Set or retrieve MATH/LIBRARY integer options.

NDAYS Compute the number of days from January 1, 1900, to the given date.

NDYIN Give the date corresponding to the number of days since January 1, 1900.

SUMAG Set or retrieve MATH/LIBRARY single-precision options.

TDATE Get today’s date.

TIMDY Get time of day.

VERML Obtain IMSL MATH/LIBRARY-related version, system and license numbers.

R1 .... Machine-dependent constants

AMACH Retrieve single-precision machine constants.

IFNAN Check if a value is NaN (not a number).

IMACH Retrieve integer machine constants.

UMACH Set or retrieve input or output device unit numbers.

R3 .... Error handling

R3b ...... Set unit number for error messages

UMACH Set or retrieve input or output device unit numbers.

R3c....... Other utilities

ERSET Set error handler default print and stop actions.

IERCD Retrieve the code for an informational error.

N1RTY Retrieve an error type for the most recently called IMSL routine.

S........... SOFTWARE DEVELOPMENT TOOLS

S3......... Dynamic program analysis tools

CPSEC Return CPU time used in seconds.

Appendix B: Alphabetical Summary of Routines

IMSL MATH/LIBRARY

`ACBCB`	1096	Add two complex band matrices, both in band storage mode.
`ACHAR`	1155	Return a character given its ASCII value.
`AMACH`	1201	Retrieve single-precision machine constants.
`ARBRB`	1095	Add two band matrices, both in band storage mode.
`BCLSF`	952	Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.
`BCLSJ`	958	Solve a nonlinear least squares problem subject to bounds on the variables using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.
`BCNLS`	964	Solve a nonlinear least-squares problem subject to bounds on the variables and general linear constraints.
`BCOAH`	942	Minimize a function of `N` variables subject to bounds the variables using a modified Newton method and a user-supplied Hessian.
`BCODH`	936	Minimize a function of `N` variables subject to bounds the variables using a modified Newton method and a finite-difference Hessian.
`BCONF`	923	Minimize a function of `N` variables subject to bounds the variables using a quasi-Newton method and a finite-difference gradient.
`BCONG`	930	Minimize a function of `N` variables subject to bounds the variables using a quasi-Newton method and a user-supplied gradient.

`BCPOL`	948	Minimize a function of `N` variables subject to bounds the variables using a direct search complex algorithm.
`BLINF`	1086	Compute the bilinear form $x^T Ay$.
`BS1GD`	473	Evaluate the derivative of a spline on a grid, given its B-spline representation.
`BS2DR`	480	Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation.
`BS2GD`	483	Evaluate the derivative of a two-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.
`BS2IG`	487	Evaluate the integral of a tensor-product spline on a rectangular domain, given its tensor-product B-spline representation.
`BS2IN`	459	Compute a two-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.
`BS2VL`	479	Evaluate a two-dimensional tensor-product spline, given its tensor-product B-spline representation.
`BS3DR`	491	Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation.
`BS3GD`	495	Evaluate the derivative of a three-dimensional tensor-product spline, given its tensor-product B-spline representation on a grid.
`BS3IG`	500	Evaluate the integral of a tensor-product spline in three dimensions over a three-dimensional rectangle, given its tensorproduct B-spline representation.
`BS3IN`	464	Compute a three-dimensional tensor-product spline interpolant, returning the tensor-product B-spline coefficients.
`BS3VL`	490	Evaluate a three-dimensional tensor-product spline, given its tensor-product B-spline representation.
`BSCPP`	504	Convert a spline in B-spline representation to piecewise polynomial representation.
`BSDER`	471	Evaluate the derivative of a spline, given its B-spline representation.
`BSINT`	450	Compute the spline interpolant, returning the B-spline coefficients.
`BSITG`	476	Evaluate the integral of a spline, given its B-spline representation.

BSLS2	561	Compute a two-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.
BSLS3	566	Compute a three-dimensional tensor-product spline approximant using least squares, returning the tensor-product B-spline coefficients.
BSLSQ	543	Compute the least-squares spline approximation, and return the B-spline coefficients.
BSNAK	454	Compute the ‘not-a-knot’ spline knot sequence.
BSOPK	457	Compute the ‘optimal’ spline knot sequence.
BSVAL	469	Evaluate a spline, given its B-spline representation.
BSVLS	547	Compute the variable knot B-spline least squares approximation to given data.
BVPFD	678	Solve a (parameterized) system of differential equations with boundary conditions at two points, using a variable order, variable step size finite-difference method with deferred corrections.
BVPMS	689	Solve a (parameterized) system of differential equations with boundary conditions at two points, using a multiple-shooting method.
CADD	1038	Add a scalar to each component of a vector, $x \leftarrow x + a$, all complex.
CAXPY	1038	Compute the scalar times a vector plus a vector, $y \leftarrow ax + y$, all complex.
CCBCB	1060	Copy a complex band matrix stored in complex band storage mode.
CCBCG	1065	Convert a complex matrix in band storage mode to a complex matrix in full storage mode.
CCGCB	1064	Convert a complex general matrix to a matrix in complex band storage mode.
CCGCG	1058	Copy a complex general matrix.
CCONV	814	Compute the convolution of two complex vectors.
CCOPY	1037	Copy a vector x to a vector y, both complex.
CCORL	823	Compute the correlation of two complex vectors.
CDGRD	1007	Approximate the gradient using central differences.
CDOTC	1039	Compute the complex conjugate dot product, $\bar{x}^T y$.
CDOTU	1039	Compute the complex dot product $x^T y$.

CGBMV	1049	Compute one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y,$ where *A* is a matrix stored in band storage mode.
CGEMM	1053	Compute one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^T B + \beta C,\ C \leftarrow \alpha AB^T$ $+\beta C,\ C \leftarrow \alpha A^T B^T + \beta C,\ C \leftarrow \alpha A\overline{B}^T + \beta C,$ or $C \leftarrow \alpha \overline{A}^T B + \beta C,\ C \leftarrow \alpha A^T \overline{B}^T + \beta C,$ $C \leftarrow \alpha \overline{A}^T B^T + \beta C,$ or $C \leftarrow \alpha \overline{A}^T \overline{B}^T + \beta C$
CGEMV	1049	Compute one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y,$ or $y \leftarrow \alpha \overline{A}^T + \beta y,$
CGERC	1052	Compute the rank-one update of a complex general matrix: $A \leftarrow A + \alpha x\overline{y}^T.$
CGERU	1052	Compute the rank-one update of a complex general matrix: $A \leftarrow A + \alpha xy^T.$
CHBCB	1074	Copy a complex Hermitian band matrix stored in band Hermitian storage mode to a complex band matrix stored in band storage mode.
CHBMV	1050	Compute the matrix-vector operation $y \leftarrow \alpha Ax + \beta y,$ where *A* is an Hermitian band matrix in band Hermitian storage.
CHEMM	1054	Compute one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C,$ where *A* is an Hermitian matrix and *B* and *C* are *m* by *n* matrices.
CHEMV	1050	Compute the matrix-vector operation $y \leftarrow \alpha Ax + \beta y,$ where *A* is an Hermitian matrix.
CHER	1052	Compute the rank-one update of an Hermitian matrix: $A \leftarrow A + \alpha x\overline{x}^T$ with *x* complex and α real.
CHER2	1052	Compute a rank-two update of an Hermitian matrix: $A \leftarrow A + \alpha x\overline{y}^T + \overline{\alpha} y\overline{x}^T.$
CHER2K	1055	Compute one of the Hermitian rank 2*k* operations: $C \leftarrow \alpha A\overline{B}^T + \overline{\alpha} B\overline{A}^T + \beta C$ or $C \leftarrow \alpha \overline{A}^T B + \overline{\alpha}\overline{B}^T A + \beta C,$ where *C* is an *n* by *n* Hermitian matrix and *A* and *B* are *n* by *k* matrices in the first case and *k* by *n* matrices in the second case.

CHERK 1054 Compute one of the Hermitian rank k operations:
$C \leftarrow \alpha A\overline{A}^T + \beta C$ or $C \leftarrow \alpha \overline{A}^T A + \beta C$,
where C is an n by n Hermitian matrix and A is an n by k matrix in the first case and a k by n matrix in the second case.

CHFCG 1071 Extend a complex Hermitian matrix defined in its upper triangle to its lower triangle.

CHGRD 1018 Check a user-supplied gradient of a function.

CHHES 1021 Check a user-supplied Hessian of an analytic function.

CHJAC 1024 Check a user-supplied Jacobian of a system of equations with M functions in N unknowns.

CONFT 551 Compute the least-squares constrained spline approximation, returning the B-spline coefficients.

CONST 1185 Return the value of various mathematical and physical constants.

CPSEC 1161 Return CPU time used in seconds.

CRBCB 1069 Convert a real matrix in band storage mode to a complex matrix in band storage mode.

CRBRB 1059 Copy a real band matrix stored in band storage mode.

CRBRG 1063 Convert a real matrix in band storage mode to a real general matrix.

CRGCG 1067 Copy a real general matrix to a complex general matrix.

CRGRB 1062 Convert a real general matrix to a matrix in band storage mode.

CRGRG 1057 Copy a real general matrix.

CRRCR 1068 Copy a real rectangular matrix to a complex rectangular matrix.

CS1GD 443 Evaluate the derivative of a cubic spline on a grid.

CSAKM 432 Compute the Akima cubic spline interpolant.

CSBRB 1073 Copy a real symmetric band matrix stored in band symmetric storage mode to a real band matrix stored in band storage mode.

CSCAL 1037 Multiply a vector by a scalar, $y \leftarrow ay$, both complex.

CSCON 434 Compute a cubic spline interpolant that is consistent with the concavity of the data.

CSDEC 425 Compute the cubic spline interpolant with specified derivative endpoint conditions.

CSDER	441	Evaluate the derivative of a cubic spline.
CSET	1037	Set the components of a vector to a scalar, all complex.
CSFRG	1070	Extend a real symmetric matrix defined in its upper triangle to its lower triangle.
CSHER	429	Compute the Hermite cubic spline interpolant.
CSIEZ	420	Compute the cubic spline interpolant with the ‘not-a-knot’ condition and return values of the interpolant at specified points.
CSINT	423	Compute the cubic spline interpolant with the ‘not-a-knot’ condition.
CSITG	445	Evaluate the integral of a cubic spline.
CSPER	438	Compute the cubic spline interpolant with periodic boundary conditions.
CSROT	1044	Apply a complex Givens plane rotation.
CSROTM	1045	Apply a complex modified Givens plane rotation.
CSSCAL	1137	Multiply a complex vector by a single-precision scalar, $y \leftarrow ay$.
CSSCV	578	Compute a smooth cubic spline approximation to noisy data using cross-validation to estimate the smoothing parameter.
CSSED	572	Smooth one-dimensional data by error detection.
CSSMH	575	Compute a smooth cubic spline approximation to noisy data.
CSUB	1038	Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all complex.
CSVAL	440	Evaluate a cubic spline.
CSVCAL	1038	Multiply a complex vector by a single-precision scalar and store the result in another complex vector, $y \leftarrow ax$.
CSWAP	1038	Interchange vectors x and y, both complex.
CSYMM	1053	Compute one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is a symmetric matrix and B and C are m by n matrices.
CSYR2K	1055	Compute one of the symmetric rank 2k operations: $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$, where C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.

`CSYRK`	1054	Compute one of the symmetric rank *k* operations: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C,$ where *C* is an *n* by *n* symmetric matrix and *A* is an *n* by *k* matrix in the first case and a *k* by *n* matrix in the second case.
`CTBMV`	1051	Compute one of the matrix-vector operations: $x \leftarrow Ax, x \leftarrow A^T x,$ or $x \leftarrow \overline{A}^T x,$ where *A* is a triangular matrix in band storage mode.
`CTBSV`	1051	Solve one of the complex triangular systems: $x \leftarrow A^{-1}x, \ x \leftarrow \left(A^{-1}\right)^T x,$ or $x \leftarrow \left(\overline{A}^T\right)^{-1} x,$ where *A* is a triangular matrix in band storage mode.
`CTRMM`	1055	Compute one of the matrix-matrix operations: $B \leftarrow \alpha AB, B \leftarrow \alpha A^T B, B \leftarrow \alpha BA, B \leftarrow \alpha BA^T,$ $B \leftarrow \alpha \overline{A}^T B,$ or $B \leftarrow \alpha B\overline{A}^T$ where *B* is an *m* by *n* matrix and *A* is a triangular matrix.
`CTRMV`	1050	Compute one of the matrix-vector operations: $x \leftarrow Ax, x \leftarrow A^T x,$ or $x \leftarrow \overline{A}^T x,$ where *A* is a triangular matrix.
`CTRSM`	1055	Solve one of the complex matrix equations: $B \leftarrow \alpha A^{-1}B, B \leftarrow \alpha BA^{-1}, B \leftarrow \alpha\left(A^{-1}\right)^T B, B \leftarrow \alpha B\left(A^{-1}\right)^T,$ $B \leftarrow \alpha\left(\overline{A}^T\right)^{-1} B,$ or $B \leftarrow \alpha B\left(\overline{A}^T\right)^{-1}$ where *A* is a traiangular matrix.
`CTRSV`	1051	Solve one of the complex triangular systems: $x \leftarrow A^{-1}x, x \leftarrow \left(A^{-1}\right)^T x,$ or $x \leftarrow \left(\overline{A}^T\right)^{-1} x,$ where *A* is a triangular matrix.
`CUNIT`	1187	Convert `X` in units `XUNITS` to `Y` in units `YUNITS`.
`CVCAL`	1038	Multiply a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all complex.
`CVTSI`	1160	Convert a character string containing an integer number into the corresponding integer form.
`CZCDOT`	1039	Compute the sum of a complex scalar plus a complex conjugate dot product, $a + \overline{x}^T y$, using a double-precision accumulator.

`CZDOTA`	1040	Compute the sum of a complex scalar, a complex dot product and the double-complex accumulator, which is set to the result `ACC` ← `ACC` + $a + x^T y$.
`CZDOTC`	1039	Compute the complex conjugate dot product, $\bar{x}^T y$, using a double-precision accumulator.
`CZDOTI`	1040	Compute the sum of a complex scalar plus a complex dot product using a double-complex accumulator, which is set to the result `ACC` ← $a + x^T y$.
`CZDOTU`	1039	Compute the complex dot product $x^T y$ using a double-precision accumulator.
`CZUDOT`	1039	Compute the sum of a complex scalar plus a complex dot product, $a + x^T y$, using a double-precision accumulator.
`DASPG`	696	Solve a first order differential-algebraic system of equations, $g(t, y, y') = 0$, using Petzold–Gear BDF method.
`DERIV`	636	Compute the first, second or third derivative of a user-supplied function.
`DISL1`	1105	Compute the 1-norm distance between two points.
`DISL2`	1104	Compute the Euclidean (2-norm) distance between two points.
`DISLI`	1106	Compute the infinity norm distance between two points.
`DLPRS`	973	Solve a linear programming problem via the revised simplex algorithm.
`DQADD`	1112	Add a double-precision scalar to the accumulator in extended precision.
`DQINI`	1111	Initialize an extended-precision accumulator with a double-precision scalar.
`DQMUL`	1112	Multiply double-precision scalars in extended precision.
`DQSTO`	1111	Store a double-precision approximation to an extended-precision scalar.
`DSDOT`	1039	Compute the single-precision dot product $x^T y$ using a double precision accumulator.
`DUMAG`	1178	This routine handles MATH/LIBRARY and STAT/LIBRARY type `DOUBLE PRECISION` options.
`EPICG`	336	Compute the performance index for a complex eigensystem.
`EPIHF`	382	Compute the performance index for a complex Hermitian eigensystem.
`EPIRG`	330	Compute the performance index for a real eigensystem.

`EPISB`	366	Compute the performance index for a real symmetric eigensystem in band symmetric storage mode.
`EPISF`	350	Compute the performance index for a real symmetric eigensystem.
`ERSET`	1196	Set error handler default print and stop actions.
`EVAHF`	372	Compute the largest or smallest eigenvalues of a complex Hermitian matrix.
`EVASB`	356	Compute the largest or smallest eigenvalues of a real symmetric matrix in band symmetric storage mode.
`EVASF`	341	Compute the largest or smallest eigenvalues of a real symmetric matrix.
`EVBHF`	376	Compute the eigenvalues in a given range of a complex Hermitian matrix.
`EVBSB`	361	Compute the eigenvalues in a given interval of a real symmetric matrix stored in band symmetric storage mode.
`EVBSF`	345	Compute selected eigenvalues of a real symmetric matrix.
`EVCCG`	330	Compute all of the eigenvalues and eigenvectors of a complex matrix.
`EVCCH`	388	Compute all of the eigenvalues and eigenvectors of a complex upper Hessenberg matrix.
`EVCHF`	369	Compute all of the eigenvalues and eigenvectors of a complex Hermitian matrix.
`EVCRG`	327	Compute all of the eigenvalues and eigenvectors of a real matrix.
`EVCRH`	385	Compute all of the eigenvalues and eigenvectors of a real upper Hessenberg matrix.
`EVCSB`	353	Compute all of the eigenvalues and eigenvectors of a real symmetric matrix in band symmetric storage mode.
`EVCSF`	339	Compute all of the eigenvalues and eigenvectors of a real symmetric matrix.
`EVEHF`	374	Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a complex Hermitian matrix.
`EVESB`	358	Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix in band symmetric storage mode.
`EVESF`	343	Compute the largest or smallest eigenvalues and the corresponding eigenvectors of a real symmetric matrix.

EVFHF	379	Compute the eigenvalues in a given range and the corresponding eigenvectors of a complex Hermitian matrix.
EVFSB	363	Compute the eigenvalues in a given interval and the corresponding eigenvectors of a real symmetric matrix stored in band symmetric storage mode.
EVFSF	347	Compute selected eigenvalues and eigenvectors of a real symmetric matrix.
EVLCG	331	Compute all of the eigenvalues of a complex matrix.
EVLCH	387	Compute all of the eigenvalues of a complex upper Hessenberg matrix.
EVLHF	367	Compute all of the eigenvalues of a complex Hermitian matrix.
EVLRG	325	Compute all of the eigenvalues of a real matrix.
EVLRH	383	Compute all of the eigenvalues of a real upper Hessenberg matrix.
EVLSB	351	Compute all of the eigenvalues of a real symmetric matrix in band symmetric storage mode.
EVLSF	337	Compute all of the eigenvalues of a real symmetric matrix.
FCOSI	784	Compute parameters needed by FCOST.
FCOST	782	Compute the discrete Fourier cosine transformation of an even sequence.
FDGRD	1009	Approximate the gradient using forward differences.
FDHES	1011	Approximate the Hessian using forward differences and function values.
FDJAC	1015	Approximate the Jacobian of M functions in N unknowns using forward differences.
FFT2B	800	Compute the inverse Fourier transform of a complex periodic two-dimensional array.
FFT2D	797	Compute Fourier coefficients of a complex periodic two-dimensional array.
FFT3B	806	Compute the inverse Fourier transform of a complex periodic three-dimensional array.
FFT3F	803	Compute Fourier coefficients of a complex periodic threedimensional array.
FFTCB	774	Compute the complex periodic sequence from its Fourier coefficients.
FFTCF	772	Compute the Fourier coefficients of a complex periodic sequence.

FFTCI	777	Compute parameters needed by FFTCF and FFTCB.
FFTRB	768	Compute the real periodic sequence from its Fourier coefficients.
FFTRF	765	Compute the Fourier coefficients of a real periodic sequence.
FFTRI	770	Compute parameters needed by FFTRF and FFTRB.
FNLSQ	539	Compute a least-squares approximation with user-supplied basis functions.
FPS2H	734	Solve Poisson's or Helmholtz's equation on a two-dimensional rectangle using a fast Poisson solver based on the HODIE finite-difference scheme on a uni mesh.
FPS3H	739	Solve Poisson's or Helmholtz's equation on a three-dimensional box using a fast Poisson solver based on the HODIE finite-difference scheme on a uniform mesh.
FQRUL	632	Compute a Fejér quadrature rule with various classical weight functions.
FSINI	781	Compute parameters needed by FSINT.
FSINT	779	Compute the discrete Fourier sine transformation of an odd sequence.
GDHES	1013	Approximate the Hessian using forward differences and a user-supplied gradient.
GGUES	1027	Generate points in an N-dimensional space.
GMRES	262	Use restarted GMRES with reverse communication to generate an approximate solution of $Ax = b$.
GPICG	403	Compute the performance index for a generalized complex eigensystem $Az = \lambda Bz$.
GPIRG	396	Compute the performance index for a generalized real eigensystem $Az = \lambda Bz$.
GPISP	409	Compute the performance index for a generalized real symmetric eigensystem problem.
GQRCF	625	Compute a Gauss, Gauss-Radau or Gauss-Lobatto quadrature rule given the recurrence coefficients for the monic polynomials orthogonal with respect to the weight function.
GQRUL	621	Compute a Gauss, Gauss-Radau, or Gauss-Lobatto quadrature rule with various classical weight functions.
GVCCG	400	Compute all of the eigenvalues and eigenvectors of a generalized complex eigensystem $Az = \lambda Bz$.

GVCRG	393	Compute all of the eigenvalues and eigenvectors of a generalized real eigensystem $Az = \lambda Bz$.
GVCSP	407	Compute all of the eigenvalues and eigenvectors of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.
GVLCG	398	Compute all of the eigenvalues of a generalized complex eigensystem $Az = \lambda Bz$.
GVLRG	391	Compute all of the eigenvalues of a generalized real eigensystem $Az = \lambda Bz$.
GVLSP	405	Compute all of the eigenvalues of the generalized real symmetric eigenvalue problem $Az = \lambda Bz$, with B symmetric positive definite.
HRRRR	1084	Compute the Hadamard product of two real rectangular matrices.
HYPOT	1190	Compute $\sqrt{a^2 + b^2}$ without underflow or overflow.
IACHAR	1156	Return the integer ASCII value of a character argument.
IADD	1038	Add a scalar to each component of a vector, $x \leftarrow x + a$, all integer.
ICAMAX	1042	Find the smallest index of the component of a complex vector having maximum magnitude.
ICAMIN	1042	Find the smallest index of the component of a complex vector having minimum magnitude.
ICASE	1157	Return the ASCII value of a character converted to uppercase.
ICOPY	1037	Copy a vector x to a vector y, both integer.
IDYWK	1165	Compute the day of the week for a given date.
IERCD	1196	Retrieve the code for an informational error.
IFNAN	1204	Check if a value is NaN (not a number).
IICSR	1157	Compare two character strings using the ASCII collating sequence but without regard to case.
IIDEX	1159	Determine the position in a string at which a given character sequence begins without regard to case.
IIMAX	1042	Find the smallest index of the maximum component of a integer vector.
IIMIN	1042	Find the smallest index of the minimum of an integer vector.
IMACH	1201	Retrieve integer machine constants.

`INLAP`	827	Compute the inverse Laplace transform of a complex function.
`ISAMAX`	1042	Find the smallest index of the component of a single-precision vector having maximum absolute value.
`ISAMIN`	1042	Find the smallest index of the component of a single-precision vector having minimum absolute value.
`ISET`	1037	Set the components of a vector to a scalar, all integer.
`ISMAX`	1042	Find the smallest index of the component of a single-precision vector having maximum value.
`ISMIN`	1042	Find the smallest index of the component of a single-precision vector having minimum value.
`ISRCH`	1152	Search a sorted integer vector for a given integer and return its index.
`ISUB`	1038	Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all integer.
`ISUM`	1041	Sum the values of an integer vector.
`ISWAP`	1038	Interchange vectors x and y, both integer.
`IUMAG`	1173	Set or retrieve MATH/LIBRARY integer options.
`IVMRK`	652	Solve an initial-value problem $y' = f(t, y)$ for ordinary differential equations using Runge-Kutta pairs of various orders.
`IVPAG`	646	Solve an initial-value problem for ordinary differential equations using either Adams-Moulton’s or Gear’s BDF method.
`IVPRK`	645	Solve an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner fifth-order and sixth-order method.
`IWKCIN`	1201	Initialize bookkeeping locations describing the character workspace stack.
`IWKIN`	1200	Initialize bookkeeping locations describing the workspace stack.
`JCGRC`	259	Solve a real symmetric definite linear system using the Jacobi preconditioned conjugate gradient method with reverse communication.
`LCHRG`	299	Compute the Cholesky decomposition of a symmetric positive semidefinite matrix with optional column pivoting.
`LCLSQ`	282	Solve a linear least-squares problem with linear constraints.

`LCONF`	984	Minimize a general objective function subject to linear equality/inequality constraints.
`LCONG`	990	Minimize a general objective function subject to linear equality/inequality constraints.
`LDNCH`	304	Downdate the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is removed.
`LFCCB`	164	Compute the LU factorization of a complex matrix in band storage mode and estimate its L_1 condition number.
`LFCCG`	32	Compute the LU factorization of a complex general matrix and estimate its L_1 condition number.
`LFCCT`	52	Estimate the condition number of a complex triangular matrix.
`LFCDH`	92	Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix and estimate its L_1 condition number.
`LFCDS`	61	Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix and estimate its L_1condition number.
`LFCHF`	108	Compute the $U DU^H$ factorization of a complex Hermitian matrix and estimate its L_1 condition number.
`LFCQH`	184	Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode and estimate its L_1 condition number.
`LFCQS`	145	Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode and estimate its L_1 condition number.
`LFCRB`	127	Compute the LU factorization of a real matrix in band storage mode and estimate its L_1 condition number.
`LFCRG`	15	Compute the LU factorization of a real general matrix and estimate its L_1 condition number.
`LFCRT`	46	Estimate the condition number of a real triangular matrix.
`LFCSF`	77	Compute the $U DU^T$ factorization of a real symmetric matrix and estimate its L_1 condition number.
`LFDCB`	175	Compute the determinant of a complex matrix given the LU factorization of the matrix in band storage mode.

`LFDCG`	42	Compute the determinant of a complex general matrix given the *LU* factorization of the matrix.
`LFDCT`	54	Compute the determinant of a complex triangular matrix.
`LFDDH`	101	Compute the determinant of a complex Hermitian positive definite matrix given the $R^H R$ Cholesky factorization of the matrix.
`LFDDS`	69	Compute the determinant of a real symmetric positive definite matrix given the $R^H R$ Cholesky factorization of the matrix.
`LFDHF`	117	Compute the determinant of a complex Hermitian matrix given the $U DU^H$ factorization of the matrix.
`LFDQH`	193	Compute the determinant of a complex Hermitian positive definite matrix given the $R^H R$ Cholesky factorization in band Hermitian storage mode.
`LFDQS`	153	Compute the determinant of a real symmetric positive definite matrix given the $R^T R$ Cholesky factorization of the band symmetric storage mode.
`LFDRB`	136	Compute the determinant of a real matrix in band storage mode given the *LU* factorization of the matrix.
`LFDRG`	24	Compute the determinant of a real general matrix given the *LU* factorization of the matrix.
`LFDRT`	48	Compute the determinant of a real triangular matrix.
`LFDSF`	85	Compute the determinant of a real symmetric matrix given the $U DU^T$ factorization of the matrix.
`LFICB`	172	Use iterative refinement to improve the solution of a complex system of linear equations in band storage mode.
`LFICG`	39	Use iterative refinement to improve the solution of a complex general system of linear equations.
`LFIDH`	99	Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations.
`LFIDS`	67	Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations.
`LFIHF`	114	Use iterative refinement to improve the solution of a complex Hermitian system of linear equations.
`LFIQH`	191	Use iterative refinement to improve the solution of a complex Hermitian positive definite system of linear equations in band Hermitian storage mode.

LFIQS	151	Use iterative refinement to improve the solution of a real symmetric positive definite system of linear equations in band symmetric storage mode.
LFIRB	134	Use iterative refinement to improve the solution of a real system of linear equations in band storage mode.
LFIRG	22	Use iterative refinement to improve the solution of a real general system of linear equations.
LFISF	83	Use iterative refinement to improve the solution of a real symmetric system of linear equations.
LFSCB	170	Solve a complex system of linear equations given the LU factorization of the coefficient matrix in band storage mode.
LFSCG	37	Solve a complex general system of linear equations given the LU factorization of the coefficient matrix.
LFSDH	97	Solve a complex Hermitian positive definite system of linear equations given the $R^H R$ factorization of the coefficient matrix.
LFSDS	65	Solve a real symmetric positive definite system of linear equations given the $R^T R$ Choleksy factorization of the coefficient matrix.
LFSHF	112	Solve a complex Hermitian system of linear equations given the $U DU^H$ factorization of the coefficient matrix.
LFSQH	189	Solve a complex Hermitian positive definite system of linear equations given the factorization of the coefficient matrix in band Hermitian storage mode.
LFSQS	149	Solve a real symmetric positive definite system of linear equations given the factorization of the coefficient matrix in band symmetric storage mode.
LFSRB	132	Solve a real system of linear equations given the LU factorization of the coefficient matrix in band storage mode.
LFSRG	20	Solve a real general system of linear equations given the LU factorization of the coefficient matrix.
LFSSF	81	Solve a real symmetric system of linear equations given the $U DU^T$ factorization of the coefficient matrix.
LFSXD	232	Solve a real sparse symmetric positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.
LFSXG	204	Solve a sparse system of linear equations given the LU factorization of the coefficient matrix.

LFSZD	244	Solve a complex sparse Hermitian positive definite system of linear equations, given the Cholesky factorization of the coefficient matrix.
LFSZG	217	Solve a complex sparse system of linear equations given the *LU* factorization of the coefficient matrix.
LFTCB	167	Compute the *LU* factorization of a complex matrix in band storage mode.
LFTCG	35	Compute the *LU* factorization of a complex general matrix.
LFTDH	95	Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix.
LFTDS	64	Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix.
LFTHF	110	Compute the $U DU^H$ factorization of a complex Hermitian matrix.
LFTQH	187	Compute the $R^H R$ factorization of a complex Hermitian positive definite matrix in band Hermitian storage mode.
LFTQS	148	Compute the $R^T R$ Cholesky factorization of a real symmetric positive definite matrix in band symmetric storage mode.
LFTRB	130	Compute the *LU* factorization of a real matrix in band storage mode.
LFTRG	18	Compute the *LU* factorization of a real general matrix.
LFTSF	80	Compute the $U DU^T$ factorization of a real symmetric matrix.
LFTXG	199	Compute the *LU* factorization of a real general sparse matrix.
LFTZG	212	Compute the *LU* factorization of a complex general sparse matrix.
LINCG	43	Compute the inverse of a complex general matrix.
LINCT	55	Compute the inverse of a complex triangular matrix.
LINDS	71	Compute the inverse of a real symmetric positive definite matrix.
LINRG	26	Compute the inverse of a real general matrix.
LINRT	49	Compute the inverse of a real triangular matrix.
LNFXD	228	Compute the numerical Cholesky factorization of a sparse symmetrical matrix *A*.

`LNFZD`	240	Compute the numerical Cholesky factorization of a sparse Hermitian matrix *A*.
`LQERR`	289	Accumulate the orthogonal matrix *Q* from its factored form given the *QR* factorization of a rectangular matrix *A*.
`LQRRR`	286	Compute the *QR* decomposition, *AP* = *QR*, using Householder transformations.
`LQRRV`	275	Compute the least-squares solution using Householder transformations applied in blocked form.
`LQRSL`	292	Compute the coordinate transformation, projection, and complete the solution of the least-squares problem $Ax = b$.
`LSACB`	159	Solve a complex system of linear equations in band storage mode with iterative refinement.
`LSACG`	27	Solve a complex general system of linear equations with iterative refinement.
`LSADH`	87	Solve a Hermitian positive definite system of linear equations with iterative refinement.
`LSADS`	56	Solve a real symmetric positive definite system of linear equations with iterative refinement.
`LSAHF`	103	Solve a complex Hermitian system of linear equations with iterative refinement.
`LSAQH`	176	Solve a complex Hermitian positive definite system of linear equations in band Hermitian storage mode with iterative refinement.
`LSAQS`	138	Solve a real symmetric positive definite system of linear equations in band symmetric storage mode with iterative refinement.
`LSARB`	122	Solve a real system of linear equations in band storage mode with iterative refinement.
`LSARG`	10	Solve a real general system of linear equations with iterative refinement.
`LSASF`	72	Solve a real symmetric system of linear equations with iterative refinement.
`LSBRR`	279	Solve a linear least-squares problem with iterative refinement.
`LSCXD`	224	Perform the symbolic Cholesky factorization for a sparse symmetric matrix using a minimum degree ordering or a userspecified ordering, and set up the data structure for the numerical Cholesky factorization.
`LSGRR`	315	Compute the generalized inverse of a real matrix.

`LSLCB`	162	Solve a complex system of linear equations in band storage mode without iterative refinement.
`LSLCC`	251	Solve a complex circulant linear system.
`LSLCG`	30	Solve a complex general system of linear equations without iterative refinement.
`LSLCQ`	156	Compute the *LDU* factorization of a complex tridiagonal matrix A using a cyclic reduction algorithm.
`LSLCR`	120	Compute the *LDU* factorization of a real tridiagonal matrix A using a cyclic reduction algorithm.
`LSLCT`	50	Solve a complex triangular system of linear equations.
`LSLDH`	89	Solve a complex Hermitian positive definite system of linear equations without iterative refinement.
`LSLDS`	59	Solve a real symmetric positive definite system of linear equations without iterative refinement.
`LSLHF`	105	Solve a complex Hermitian system of linear equations without iterative refinement.
`LSLPB`	143	Compute the $R^T DR$ Cholesky factorization of a real symmetric positive definite matrix A in codiagonal band symmetric storage mode. Solve a system $Ax = b$.
`LSLQB`	182	Compute the $R^H DR$ Cholesky factorization of a complex hermitian positive-definite matrix A in codiagonal band hermitian storage mode. Solve a system $Ax = b$.
`LSLQH`	179	Solve a complex Hermitian positive definite system of linearequations in band Hermitian storage mode without iterative refinement.
`LSLQS`	140	Solve a real symmetric positive definite system of linear equations in band symmetric storage mode without iterative refinement.
`LSLRB`	124	Solve a real system of linear equations in band storage mode without iterative refinement.
`LSLRG`	12	Solve a real general system of linear equations without iterative refinement.
`LSLRT`	45	Solve a real triangular system of linear equations.
`LSLSF`	75	Solve a real symmetric system of linear equations without iterative refinement.
`LSLTC`	249	Solve a complex Toeplitz linear system.
`LSLTO`	248	Solve a real Toeplitz linear system.
`LSLTQ`	155	Solve a complex tridiagonal system of linear equations.

`LSLTR`	118	Solve a real tridiagonal system of linear equations.
`LSLXD`	220	Solve a sparse system of symmetric positive definite linear algebraic equations by Gaussian elimination.
`LSLXG`	195	Solve a sparse system of linear algebraic equations by Gaussian elimination.
`LSLZD`	236	Solve a complex sparse Hermitian positive definite system of linear equations by Gaussian elimination.
`LSLZG`	207	Solve a complex sparse system of linear equations by Gaussian elimination.
`LSQRR`	272	Solve a linear least-squares problem without iterative refinement.
`LSVCR`	311	Compute the singular value decomposition of a complex matrix.
`LSVRR`	307	Compute the singular value decomposition of a real matrix.
`LUPCH`	301	Update the R^TR Cholesky factorization of a real symmetric positive definite matrix after a rank-one matrix is added.
`LUPQR`	295	Compute an updated QR factorization after the rank-one matrix αxy^T is added.
`MCRCR`	1083	Multiply two complex rectangular matrices, AB.
`MOLCH`	717	Solve a system of partial differential equations of the form $u_t = f(x, t, u, u_x, u_{xx})$ using the method of lines. The solution is represented with cubic Hermite polynomials.
`MRRRR`	1081	Multiply two real rectangular matrices, AB.
`MUCBV`	1093	Multiply a complex band matrix in band storage mode by a complex vector.
`MUCRV`	1092	Multiply a complex rectangular matrix by a complex vector.
`MURBV`	1090	Multiply a real band matrix in band storage mode by a real vector.
`MURRV`	1089	Multiply a real rectangular matrix by a vector.
`MXTXF`	1077	Compute the transpose product of a matrix, A^TA.
`MXTYF`	1078	Multiply the transpose of matrix A by matrix B, A^TB.
`MXYTF`	1079	Multiply a matrx A by the transpose of a matrix B, AB^T.
`N1RTY`	1196	Retrieve an error type for the most recently called IMSL routine.

NCONF	996	Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a finite difference gradient.
NCONG	1003	Solve a general nonlinear programming problem using the successive quadratic programming algorithm and a user-supplied gradient.
NDAYS	1163	Compute the number of days from January 1, 1900, to the given date.
NDYIN	1164	Give the date corresponding to the number of days since January 1, 1900.
NEQBF	854	Solve a system of nonlinear equations using factored secant update with a finite-difference approximation to the Jacobian.
NEQBJ	860	Solve a system of nonlinear equations using factored secant update with a user-supplied Jacobian.
NEQNF	848	Solve a system of nonlinear equations using a modified Powell hybrid algorithm and a finite-difference approximation to the Jacobian.
NEQNJ	851	Solve a system of nonlinear equations using a modified Powell hybrid algorithm with a user-supplied Jacobian.
NR1CB	1103	Compute the 1-norm of a complex band matrix in band storage mode.
NR1RB	1102	Compute the 1-norm of a real band matrix in band storage mode.
NR1RR	1099	Compute the 1-norm of a real matrix.
NR2RR	1100	Compute the Frobenius norm of a real rectangular matrix.
NRIRR	1098	Compute the infinity norm of a real matrix.
PCGRC	253	Solve a real symmetric definite linear system using a preconditioned conjugate gradient method with reverse communication.
PERMA	1139	Permute the rows or columns of a matrix.
PERMU	1138	Rearrange the elements of an array as specified by a permutation.
PGOPT	1137	Set or retrieve page width and length for printing.
PLOTP	1181	Print a plot of up to 10 sets of points.
POLRG	1087	Evaluate a real general matrix polynomial.
PP1GD	510	Evaluate the derivative of a piecewise polynomial on a grid.
PPDER	507	Evaluate the derivative of a piecewise polynomial.

PPITG	512	Evaluate the integral of a piecewise polynomial.
PPVAL	505	Evaluate a piecewise polynomial.
PRIME	1183	Decompose an integer into its prime factors.
QAND	619	Integrate a function on a hyper-rectangle.
QCOSB	793	Compute a sequence from its cosine Fourier coefficients with only odd wave numbers.
QCOSF	791	Compute the coefficients of the cosine Fourier transform with only odd wave numbers.
QCOSI	795	Compute parameters needed by QCOSF and QCOSB.
QD2DR	520	Evaluate the derivative of a function defined on a rectangular grid using quadratic interpolation.
QD2VL	518	Evaluate a function defined on a rectangular grid using quadratic interpolation.
QD3DR	525	Evaluate the derivative of a function defined on a rectangular three-dimensional grid using quadratic interpolation.
QD3VL	523	Evaluate a function defined on a rectangular three-dimensional grid using quadratic interpolation.
QDAG	591	Integrate a function using a globally adaptive scheme based on Gauss-Kronrod rules.
QDAGI	598	Integrate a function over an infinite or semi-infinite interval.
QDAGP	594	Integrate a function with singularity points given.
QDAGS	589	Integrate a function (which may have endpoint singularities).
QDAWC	610	Integrate a function F(X)/(X – C) in the Cauchy principal value sense.
QDAWF	604	Compute a Fourier integral.
QDAWO	601	Integrate a function containing a sine or a cosine.
QDAWS	607	Integrate a function with algebraic-logarithmic singularities.
QDDER	516	Evaluate the derivative of a function defined on a set of points using quadratic interpolation.
QDNG	613	Integrate a smooth function using a nonadaptive rule.
QDVAL	514	Evaluate a function defined on a set of points using quadratic interpolation.

`QPROG`	982	Solve a quadratic programming problem subject to linear equality/inequality constraints.
`QSINB`	788	Compute a sequence from its sine Fourier coefficients with only odd wave numbers.
`QSINF`	786	Compute the coefficients of the sine Fourier transform with only odd wave numbers.
`QSINI`	790	Compute parameters needed by `QSINF` and `QSINB`.
`RATCH`	581	Compute a rational weighted Chebyshev approximation to a continuous function on an interval.
`RCONV`	810	Compute the convolution of two real vectors.
`RCORL`	818	Compute the correlation of two real vectors.
`RCURV`	535	Fit a polynomial curve using least squares.
`RECCF`	628	Compute recurrence coefficients for various monic polynomials.
`RECQR`	630	Compute recurrence coefficients for monic polynomials given a quadrature rule.
`RLINE`	532	Fit a line to a set of data points using least squares.
`RNGET`	1167	Retrieve the current value of the seed used in the IMSL random number generators.
`RNOPT`	1169	Select the uniform (0, 1) multiplicative congruential pseudorandom number generator.
`RNSET`	1168	Initialize a random seed for use in the IMSL random number generators.
`RNUN`	1171	Generate pseudorandom numbers from a uniform (0, 1) distribution.
`RNUNF`	1170	Generate a pseudorandom number from a uniform (0, 1) distribution.
`SADD`	1038	Add a scalar to each component of a vector, $x \leftarrow x + a$, all single precision.
`SASUM`	1041	Sum the absolute values of the components of a single-precision vector.
`SAXPY`	1038	Compute the scalar times a vector plus a vector, $y \leftarrow ax + y$, all single precision.
`SCASUM`	1041	Sum the absolute values of the real part together with the absolute values of the imaginary part of the components of a complex vector.
`SCNRM2`	1041	Compute the Euclidean norm of a complex vector.

SCOPY	1037	Copy a vector x to a vector y, both single precision.
SDDOTA	1040	Compute the sum of a single-precision scalar, a single-precision dot product and the double-precision accumulator, which is set to the result ACC ← ACC + $a + x^T y$.
SDDOTI	1040	Compute the sum of a single-precision scalar plus a singleprecision dot product using a double-precision accumulator, which is set to the result ACC ← $a + x^T y$.
SDOT	1039	Compute the single-precision dot product $x^T y$.
SDSDOT	1039	Compute the sum of a single-precision scalar and a single precision dot product, $a + x^T y$, using a double-precision accumulator.
SGBMV	1049	Compute one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,$ or $y \leftarrow \alpha A^T x + \beta y,$ where A is a matrix stored in band storage mode.
SGEMM	1053	Compute one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha A^T B + \beta C, C \leftarrow \alpha AB^T$. $+\beta C,$ or $C \leftarrow \alpha A^T B^T + \beta C$
SGEMV	1049	Compute one of the matrix-vector operations: $y \leftarrow \alpha Ax + \beta y,$ or $y \leftarrow \alpha A^T x + \beta y,$
SGER	1052	Compute the rank-one update of a real general matrix: $A \leftarrow A + \alpha xy^T.$
SHPROD	1040	Compute the Hadamard product of two single-precision vectors.
SINLP	830	Compute the inverse Laplace transform of a complex function.
SLCNT	757	Calculate the indices of eigenvalues of a Sturm-Liouville problem with boundary conditions (at regular points) in a specified subinterval of the real line, [α, β].
SLEIG	745	Determine eigenvalues, eigenfunctions and/or spectral density functions for Sturm-Liouville problems in the form with boundary conditions (at regular points).
SLPRS	976	Solve a sparse linear programming problem via the revised simplex algorithm.
SNRM2	1041	Compute the Euclidean length or L_2 norm of a single-precision vector.
SPLEZ	447	Compute the values of a spline that either interpolates or fits user-supplied data.

`SPRDCT`	1041	Multiply the components of a single-precision vector.
`SRCH`	1150	Search a sorted vector for a given scalar and return its index.
`SROT`	1043	Apply a Givens plane rotation in single precision.
`SROTG`	1045	Construct a Givens plane rotation in single precision.
`SROTM`	1044	Apply a modified Givens plane rotation in single precision.
`SROTMG`	1050	Construct a modified Givens plane rotation in single precision.
`SSBMV`	1037	Compute the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is a symmetric matrix in band symmetric storage mode.
`SSCAL`	1037	Multiply a vector by a scalar, $y \leftarrow ay$, both single precision.
`SSET`	1037	Set the components of a vector to a scalar, all single precision.
`SSRCH`	1153	Search a character vector, sorted in ascending ASCII order, for a given string and return its index.
`SSUB`	1038	Subtract each component of a vector from a scalar, $x \leftarrow a - x$, all single precision.
`SSUM`	1041	Sum the values of a single-precision vector.
`SSWAP`	1038	Interchange vectors x and y, both single precision.
`SSYMM`	1053	Compute one of the matrix-matrix operations: $C \leftarrow \alpha AB + \beta C$ or $C \leftarrow \alpha BA + \beta C$, where A is a symmetric matrix and B and C are m by n matrices.
`SSYMV`	1050	Compute the matrix-vector operation $y \leftarrow \alpha Ax + \beta y$, where A is a symmetric matrix.
`SSYR`	1053	Compute the rank-one update of a real symmetric matrix: $A \leftarrow A + \alpha xx^T$.
`SSYR2`	1053	Compute the rank-two update of a real symmetric matrix: $A \leftarrow A + \alpha xy^T + \alpha yx^T$.
`SSYR2K`	1054	Compute one of the symmetric rank 2k operations: $C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$ or $C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$, where C is an n by n symmetric matrix and A and B are n by k matrices in the first case and k by n matrices in the second case.
`SSYRK`	1054	Compute one of the symmetric rank k operations: $C \leftarrow \alpha AA^T + \beta C$ or $C \leftarrow \alpha A^T A + \beta C$,

		where *C* is an *n* by *n* symmetric matrix and *A* is an *n* by *k* matrix in the first case and a *k* by *n* matrix in the second case.
`STBMV`	1051	Compute one of the matrix-vector operations: $x \leftarrow Ax$ or $x \leftarrow A^T x$, where *A* is a triangular matrix in band storage mode.
`STBSV`	1051	Solve one of the triangular systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$, where *A* is a triangular matrix in band storage mode.
`STRMM`	1055	Compute one of the matrix-matrix operations: $B \leftarrow \alpha AB$, $B \leftarrow \alpha A^T B$ or $B \leftarrow \alpha BA$, $B \leftarrow \alpha BA^T$, where *B* is an *m* by *n* matrix and *A* is a triangular matrix.
`STRMV`	1050	Compute one of the matrix-vector operations: $x \leftarrow Ax$ or $x \leftarrow A^T x$, where *A* is a triangular matrix.
`STRSM`	1055	Solve one of the matrix equations: $B \leftarrow \alpha A^{-1}B$, $B \leftarrow \alpha BA^{-1}$, $B \leftarrow \alpha\left(A^{-1}\right)^T B$, or $B \leftarrow \alpha B\left(A^{-1}\right)^T$ where *B* is an *m* by *n* matrix and *A* is a triangular matrix.
`STRSV`	1051	Solve one of the triangular linear systems: $x \leftarrow A^{-1}x$ or $x \leftarrow \left(A^{-1}\right)^T x$ where *A* is a triangular matrix.
`SUMAG`	1175	Set or retrieve MATH/LIBRARY single-precision options.
`SURF`	529	Compute a smooth bivariate interpolant to scattered data that is locally a quintic polynomial in two variables.
`SVCAL`	1038	Multiply a vector by a scalar and store the result in another vector, $y \leftarrow ax$, all single precision.
`SVIBN`	1148	Sort an integer array by nondecreasing absolute value.
`SVIBP`	1149	Sort an integer array by nondecreasing absolute value and return the permutation that rearranges the array.
`SVIGN`	1143	Sort an integer array by algebraically increasing value.
`SVIGP`	1144	Sort an integer array by algebraically increasing value and return the permutation that rearranges the array.

SVRBN	1145	Sort a real array by nondecreasing absolute value.
SVRBP	1146	Sort a real array by nondecreasing absolute value and return the permutation that rearranges the array.
SVRGN	1141	Sort a real array by algebraically increasing value.
SVRGP	1142	Sort a real array by algebraically increasing value and return the permutation that rearranges the array.
SXYZ	1140	Compute a single-precision *xyz* product.
TDATE	1162	Get today’s date.
TIMDY	1161	Get time of day.
TRNRR	1075	Transpose a rectangular matrix.
TWODQ	615	Compute a two-dimensional iterated integral.
UMACH	1201	Set or retrieve input or output device unit numbers.
UMCGF	902	Minimize a function of N variables using a conjugate gradient algorithm and a finite-difference gradient.
UMCGG	905	Minimize a function of N variables using a conjugate gradient algorithm and a user-supplied gradient.
UMIAH	896	Minimize a function of N variables using a modified Newton method and a user-supplied Hessian.
UMIDH	891	Minimize a function of N variables using a modified Newton method and a finite-difference Hessian.
UMINF	881	Minimize a function of N variables using a quasi-New method and a finite-difference gradient.
UMING	886	Minimize a function of N variables using a quasi-New method and a user-supplied gradient.
UMPOL	909	Minimize a function of N variables using a direct search polytope algorithm.
UNLSF	912	Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a finite-difference Jacobian.
UNLSJ	918	Solve a nonlinear least squares problem using a modified Levenberg-Marquardt algorithm and a user-supplied Jacobian.
UVMGS	878	Find the minimum point of a nonsmooth function of a single variable.
UVMID	875	Find the minimum point of a smooth function of a single variable using both function evaluations and first derivative evaluations.

UVMIF	872	Find the minimum point of a smooth function of a single variable using only function evaluations.
VCONC	1109	Compute the convolution of two complex vectors.
VCONR	1107	Compute the convolution of two real vectors.
VERML	1166	Obtain IMSL MATH/LIBRARY-related version, system and license numbers.
WRCRL	1127	Print a complex rectangular matrix with a given format and labels.
WRCRN	1125	Print a complex rectangular matrix with integer row and column labels.
WRIRL	1123	Print an integer rectangular matrix with a given format and labels.
WRIRN	1121	Print an integer rectangular matrix with integer row and column labels.
WROPT	1130	Set or retrieve an option for printing a matrix.
WRRRL	1118	Print a real rectangular matrix with a given format and labels.
WRRRN	1116	Print a real rectangular matrix with integer row and column labels.
ZANLY	841	Find the zeros of a univariate complex function using Müller's method.
ZBREN	843	Find a zero of a real function that changes sign in a given interval.
ZPLRC	836	Find the zeros of a polynomial with real coefficients using Laguerre's method.
ZPOCC	839	Find the zeros of a polynomial with complex coefficients using the Jenkins-Traub three-stage algorithm.
ZPORC	838	Find the zeros of a polynomial with real coefficients using the Jenkins-Traub three-stage algorithm.
ZQADD	1112	Add a double complex scalar to the accumulator in extended precision.
ZQINI	1112	Initialize an extended-precision complex accumulator to a double complex scalar.
ZQMUL	1112	Multiply double complex scalars using extended precision.
ZQSTO	1112	Store a double complex approximation to an extended-precision complex scalar.
ZREAL	846	Find the real zeros of a real function using Müller's method.

Appendix C: References

Aird and Howell

Aird, Thomas J., and Byron W. Howell (1991), IMSL Technical Report 9103, IMSL, Houston.

Aird and Rice

Aird, T.J., and J.R. Rice (1977), Systematic search in high dimensional sets, *SIAM Journal on Numerical Analysis*, **14**, 296−312.

Akima

Akima, H. (1970), A new method of interpolation and smooth curve fitting based on local procedures, *Journal of the ACM*, **17**, 589−602.

Akima, H. (1978), A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points, *ACM Transactions on Mathematical Software*, **4**, 148−159.

Arushanian et al.

Arushanian, O.B., M.K. Samarin, V.V. Voevodin, E.E. Tyrtyshikov, B.S. Garbow, J.M. Boyle, W.R. Cowell, and K.W. Dritz (1983), *The TOEPLITZ Package Users' Guide*, Argonne National Laboratory, Argonne, Illinois.

Ashcraft

Ashcraft, C. (1987), *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.

Ashcraft et al.

Ashcraft, C., R.Grimes, J. Lewis, B. Peyton, and H. Simon (1987), Progress in sparse matrix methods for large linear systems on vector supercomputers. *Intern. J. Supercomputer Applic*., **1**(**4**), 10−29.

Atkinson

Atkinson, Ken (1978), *An Introduction to Numerical Analysis*, John Wiley & Sons, New York.

Atchison and Hanson

Atchison, M.A., and R.J. Hanson (1991), *An Options Manager for the IMSL Fortran 77 Libraries*, Technical Report 9101, IMSL, Houston.

Bischof et al.

Bischof, C., J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, D. Sorensen (1988), LAPACK Working Note #5: Provisional Contents, Argonne National Laboratory Report ANL-88-38, Mathematics and Computer Science.

Bjorck

Bjorck, Ake (1967), Iterative refinement of linear least squares solutions I, BIT, **7**, 322−337.

Bjorck, Ake (1968), Iterative refinement of linear least squares solutions II, BIT, **8**, 8−30.

Boisvert (1984)

Boisvert, Ronald (1984), A fourth order accurate fast direct method for the Helmholtz equation, *Elliptic Problem Solvers II*, (edited by G. Birkhoff and A. Schoenstadt), Academic Press, Orlando, Florida, 35−44.

Boisvert, Howe, and Kahaner

Boisvert, Ronald F., Sally E. Howe, and David K. Kahaner (1985), GAMS: A framework for the management of scientific software,*ACM Transactions on Mathematical Software*, **11**, 313−355.

Boisvert, Howe, Kahaner, and Springmann

Boisvert, Ronald F., Sally E. Howe, David K. Kahaner, and Jeanne L. Springmann (1990), *Guide to Available Mathematical Software*, NISTIR 90-4237, National Institute of Standards and Technology, Gaithersburg, Maryland.

Brankin et al.

Brankin, R.W., I. Gladwell, and L.F. Shampine, RKSUITE: a Suite of Runge-Kutta Codes for the Initial Value Problem for ODEs, Softreport 91-1, Mathematics Department, Southern Methodist University, Dallas, Texas, 1991.

Brenan, Campbell, and Petzold

Brenan, K.E., S.L. Campbell, L.R. Petzold (1989), *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elseview Science Publ. Co.

Brenner

Brenner, N. (1973), Algorithm 467: Matrix transposition in place [F1], *Communication of ACM*, **16**, 692−694.

Brent

Brent, R.P. (1971), An algorithm with guaranteed convergence for finding a zero of a function, *The Computer Journa*l, **14**, 422−425.

Brent, Richard P. (1973), *Algorithms for Minimization without Derivatives*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Brigham

Brigham, E. Oran (1974), *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey.

Cheney

Cheney, E.W. (1966), *Introduction to Approximation Theory*, McGraw-Hill, New York.

Cline et al.

Cline, A.K., C.B. Moler, G.W. Stewart, and J.H. Wilkinson (1979), An estimate for the condition number of a matrix, *SIAM Journal of Numerical Analysis*, **16**, 368–375.

Cody, Fraser, and Hart

Cody, W.J., W. Fraser, and J.F. Hart (1968), Rational Chebyshev approximation using linear equations, *Numerische Mathematik*, **12**, 242−251.

Cohen and Taylor

Cohen, E. Richard, and Barry N. Taylor (1986), *The 1986 Adjustment of the Fundamental Physical Constants*, Codata Bulletin, Pergamon Press, New York.

Cooley and Tukey

Cooley, J.W., and J.W. Tukey (1965), An algorithm for the machine computation of complex Fourier series, *Mathematics of Computation*, **19**, 297–301.

Courant and Hilbert

Courant, R., and D. Hilbert (1962), *Methods of Mathematical Physics, Volume II*, John Wiley & Sons, New York, NY.

Craven and Wahba

Craven, Peter, and Grace Wahba (1979), Smoothing noisy data with spline functions, *Numerische Mathematik*, **31**, 377−403.

Crowe et al.

Crowe, Keith, Yuan-An Fan, Jing Li, Dale Neaderhouser, and Phil Smith (1990), *A direct sparse linear equation solver using linked list storage*, IMSL Technical Report 9006, IMSL, Houston.

Crump

Crump, Kenny S. (1976), Numerical inversion of Laplace transforms using a Fourier series approximation, *Journal of the Association for Computing Machinery*, **23**, 89−96.

Davis and Rabinowitz

Davis, Philip F., and Philip Rabinowitz (1984), *Methods of Numerical Integration*, Academic Press, Orlando, Florida.

de Boor

de Boor, Carl (1978), *A Practical Guide to Splines*, Springer-Verlag, New York.

de Hoog, Knight, and Stokes

de Hoog, F.R., J.H. Knight, and A.N. Stokes (1982), An improved method for numerical inversion of Laplace transforms. *SIAM Journal on Scientific and Statistical Computing*, **3**, 357−366.

Dennis and Schnabel

Dennis, J.E., Jr., and Robert B. Schnabel (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.

Dongarra et al.

Dongarra, J.J., and C.B. Moler, (1977) *EISPACK − A package for solving matrix eigenvalue problems*, Argonne National Laboratory, Argonne, Illinois.

Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart (1979), *LINPACK Users' Guide*, SIAM, Philadelphia.

Dongarra, J.J., J. DuCroz, S. Hammarling, R. J. Hanson (1988), An Extended Set of Fortran basic linear algebra subprograms, *ACM Transactions on Mathematical Software*, **14** , 1−17.

Dongarra, J.J., J. DuCroz, S. Hammarling, I. Duff (1990), A set of level 3 basic linear algebra subprograms, *ACM Transactions on Mathematical Software*, **16** , 1−17.

Draper and Smith

Draper, N.R., and H. Smith (1981), *Applied Regression Analysis*, second edition, John Wiley & Sons, New York.

Du Croz et al.

Du Croz, Jeremy, P. Mayes, G. and Radicati (1990), Factorization of band matrices using Level-3 BLAS, *Proceedings of CONPAR 90 VAPP IV, Lecture Notes in Computer Science,* Springer, Berlin, 222.

Duff and Reid

Duff, I.S., and J.K. Reid (1983), The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, **9**, 302−325.

Duff, I.S., and J.K. Reid (1984), The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, **5**, 633−641.

Duff et al.

Duff, I.S., A.M. Erisman, and J.K. Reid (1986), *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.

Enright and Pryce

Enright, W.H., and J.D. Pryce (1987), Two FORTRAN packages for assessing initial value methods, *ACM Transactions on Mathematical Software*, **13**, 1−22.

Forsythe

Forsythe, G.E. (1957), Generation and use of orthogonal polynomials for fitting data with a digital computer, *SIAM Journal on Applied Mathematics*, **5**, 74–88.

Fox, Hall, and Schryer

Fox, P.A., A.D. Hall, and N.L. Schryer (1978), The PORT mathematical subroutine library, *ACM Transactions on Mathematical Software*, **4**, 104−126.

Garbow

Garbow, B.S. (1978) CALGO Algorithm 535: The QZ algorithm to solve the generalized eigenvalue problem for complex matrices, *ACM Transactions on Mathematical Software*, **4**, 404–410.

Garbow et al.

Garbow, B.S., J.M. Boyle, J.J. Dongarra, and C.B. Moler (1972), *Matrix eigensystem Routines: EISPACK Guide Extension*, Springer-Verlag, New York.

Garbow, B.S., J.M. Boyle, J.J. Dongarra, and C.B. Moler (1977), *Matrix Eigensystem Routines–EISPACK Guide Extension*, Springer-Verlag, New York.

Garbow, B.S., G. Giunta, J.N. Lyness, and A. Murli (1988), Software for an implementation of Weeks' method for the inverse Laplace transform problem, *ACM Transactions of Mathematical Software*, **14**, 163−170.

Gautschi

Gautschi, Walter (1968), Construction of Gauss-Christoffel quadrature formulas, *Mathematics of Computation*, **22**, 251–270.

Gautschi and Milovanofic

Gautschi, Walter, and Gradimir V. Milovanofic (1985), Gaussian quadrature involving Einstein and Fermi functions with an application to summation of series, *Mathematics of Computation*, **44**, 177−190.

Gay

Gay, David M. (1981), Computing optimal locally constrained steps, *SIAM Journal on Scientific and Statistical Computing*, **2**, 186−197.

Gay, David M. (1983), Algorithm 611: Subroutine for unconstrained minimization using a model/trust-region approach, *ACM Transactions on Mathematical Software*, **9**, 503− 524.

Gear

Gear, C.W. (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey.

Gear and Petzold

Gear, C.W., and Linda R. Petzold (1984), ODE methods for the solutions of differential/algebraic equations, *SIAM Journal Numerical Analysis*, **21**, #4, 716.

George and Liu

George, A., and J.W.H. Liu (1981), *Computer Solution of Large Sparse Positive-definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Gill et al.

Gill, Philip E., and Walter Murray (1976), *Minimization subject to bounds on the variables*, NPL Report NAC 72, National Physical Laboratory, England.

Gill, Philip E., Walter Murray, and Margaret Wright (1981), *Practical Optimization*, Academic Press, New York.

Gill, P.E., W. Murray, M.A. Saunders, and M.H. Wright (1985), Model building and practical aspects of nonlinear programming, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Goldfarb and Idnani

Goldfarb, D., and A. Idnani (1983), A numerically stable dual method for solving strictly convex quadratic programs, *Mathematical Programming*, **27**, 1–33.

Golub

Golub, G.H. (1973), Some modified matrix eigenvalue problems, *SIAM Review*, **15**, 318–334.

Golub and Van Loan

Golub, Gene H., and Charles F. Van Loan (1983), *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland.

Golub, Gene H., and Charles F. Van Loan (1989), *Matrix Computations*, 2d ed., Johns Hopkins University Press, Baltimore, Maryland.

Golub and Welsch

Golub, G.H., and J.H. Welsch (1969), Calculation of Gaussian quadrature rules, *Mathematics of Computation*, **23**, 221–230.

Gregory and Karney

Gregory, Robert, and David Karney (1969), *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, John Wiley & Sons, New York.

Griffin and Redish

Griffin, R., and K.A. Redish (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 54.

Grosse

Grosse, Eric (1980), Tensor spline approximation, *Linear Algebra and its Applications*, **34**, 29−41.

Guerra and Tapia

Guerra, V., and R. A. Tapia (1974), *A local procedure for error detection and data smoothing*, MRC Technical Summary Report 1452, Mathematics Research Center, University of Wisconsin, Madison.

Hageman and Young

Hageman, Louis A., and David M.Young (1981), *Applied Iterative Methods*, Academic Press, New York.

Hanson

Hanson, Richard J. (1986), Least squares with bounds and linear constraints, *SIAM Journal Sci. Stat. Computing*, **7**, #3.

Hanson, Richard.J. (1990), *A cyclic reduction solver for the IMSL Mathematics Library*, IMSL Technical Report 9002, IMSL, Houston.

Hanson et al.

Hanson, Richard J., R. Lehoucq, J. Stolle, and A. Belmonte (1990), *Improved performance of certain matrix eigenvalue computations for the IMSL/MATH Library*, IMSL Technical Report 9007, IMSL, Houston.

Hartman

Hartman, Philip (1964) *Ordinary Differential Equations*, John Wiley and Sons, New York, NY.

Hausman

Hausman, Jr., R.F. (1971), *Function Optimization on a Line Segment by Golden Section*, Lawrence Radiation Laboratory, University of California, Livermore.

Hindmarsh

Hindmarsh, A.C. (1974), *GEAR: Ordinary differential equation system solver*, Lawrence Livermore Laboratory Report UCID–30001, Revision 3.

Hull et al.

Hull, T.E., W.H. Enright, and K.R. Jackson (1976), *User’s guide for DVERK – A subroutine for solving non-stiff ODEs*, Department of Computer Science Technical Report 100, University of Toronto.

IEEE

ANSI/IEEE Std 754-1985 (1985), *IEEE Standard for Binary Floating-Point Arithmetic*, The IEEE, Inc., New York.

IMSL (1991)

IMSL (1991), IMSL STAT/LIBRARY *User’s Manual, Version 2.0*, IMSL, Houston.

Irvine et al.

Irvine, Larry D., Samuel P. Marin, and Philip W. Smith (1986), Constrained interpolation and smoothing, *Constructive Approximation*, **2**, 129−151.

Jenkins

Jenkins, M.A. (1975), Algorithm 493: Zeros of a real polynomial, *ACM Transactions on Mathematical Software*, **1**, 178−189.

Jenkins and Traub

Jenkins, M.A., and J.F. Traub (1970), A three-stage algorithm for real polynomials using quadratic iteration, *SIAM Journal on Numerical Analysis*, **7**, 545−566.

Jenkins, M.A., and J.F. Traub (1970), A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration, *Numerische Mathematik*, **14**, 252−263.

Jenkins, M.A., and J.F. Traub (1972), Zeros of a complex polynomial, *Communications of the ACM*, **15**, 97−99.

Kennedy and Gentle

Kennedy, William J., Jr., and James E. Gentle (1980), *Statistical Computing*, Marcel Dekker, New York.

Kershaw

Kershaw, D. (1982), Solution of tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray-1, *Parallel Computations*, Academic Press, Inc., 85-99.

Knuth

Knuth, Donald E. (1973), *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Addison-Wesley Publishing Company, Reading, Mass.

Lawson et al.

Lawson, C.L., R.J. Hanson, D.R. Kincaid, and F.T. Krogh (1979), Basic linear algebra subprograms for Fortran usage, *ACM Transactions on Mathematical Software*, **5**, 308− 323.

Leavenworth

Leavenworth, B. (1960), Algorithm 25: Real zeros of an arbitrary function, *Communications of the ACM*, **3**, 602.

Levenberg

Levenberg, K. (1944), A method for the solution of certain problems in least squares, *Quarterly of Applied Mathematics*, **2**, 164−168.

Lewis et al.

Lewis, P.A. W., A.S. Goodman, and J.M. Miller (1969), A pseudo-random number generator for the System/360, *IBM Systems Journa*l, **8**, 136−146.

Liepman

Liepman, David S. (1964), Mathematical constants, in *Handbook of Mathematical Functions*, Dover Publications, New York.

Liu

Liu, J.W.H. (1986), On the storage requirement in the out-of-core multifrontal method for sparse factorization. *ACM Transactions on Mathematical Software*, **12**, 249–264.

Liu, J.W.H. (1987), *A collection of routines for an implementation of the multifrontal method*, Technical Report CS-87-10, Department of Computer Science, York University, North York, Ontario, Canada.

Liu, J.W.H. (1989), The multifrontal method and paging in sparse Cholesky factorization. *ACM Transactions on Mathematical Software*, **15**, 310−325.

Liu, J.W.H. (1990), The multifrontal method for sparse matrix solution: theory and practice, Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada.

Liu and Ashcraft

Liu, J., and C. Ashcraft (1987), *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington.

Lyness and Giunta

Lyness, J.N. and G. Giunta (1986), A modification of the Weeks Method for numerical inversion of the Laplace transform, *Mathmetics of Computation*, **47**, 313–322.

Madsen and Sincovec

Madsen, N.K., and R.F. Sincovec (1979), Algorithm 540: PDECOL, General collocation software for partial differential equations, *ACM Transactions on Mathematical Software*, **5**, #3, 326-351.

Marquardt

Marquardt, D. (1963), An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal on Applied Mathematics*, **11**, 431–441.

Martin and Wilkinson

Martin, R.S., and J.W. Wilkinson (1968), Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form, *Numerische Mathematik*, **11**, 99–119.

Micchelli et al.

Micchelli, C.A., T.J. Rivlin, and S. Winograd (1976), The optimal recovery of smooth functions, *Numerische Mathematik*, **26**, 279–285

Micchelli, C.A., Philip W. Smith, John Swetits, and Joseph D. Ward (1985), Constrained L_p approximation, *Constructive Approximation*, **1**, 93–102.

Moler and Stewart

Moler, C., and G.W. Stewart (1973), An algorithm for generalized matrix eigenvalue problems, *SIAM Journal on Numerical Analysis*, **10**, 241–256.

More et al.

More, Jorge, Burton Garbow, and Kenneth Hillstrom (1980), *User guide for MINPACK-1*, Argonne National Labs Report ANL-80-74, Argonne, Illinois.

Muller

Muller, D.E. (1956), A method for solving algebraic equations using an automatic computer, *Mathematical Tables and Aids to Computation*, **10**, 208–215.

Murtagh

Murtagh, Bruce A. (1981), *Advanced Linear Programming: Computation and Practice*, McGraw-Hill, New York.

Murty

Murty, Katta G. (1983), *Linear Programming*, John Wiley and Sons, New York.

Nelder and Mead

Nelder, J.A., and R. Mead (1965), A simplex method for function minimization, *Computer Journa*l **7**, 308−313.

Neter and Wasserman

Neter, John, and William Wasserman (1974), *Applied Linear Statistical Models*, Richard D. Irwin, Homewood, Ill.

Park and Miller

Park, Stephen K., and Keith W. Miller (1988), Random number generators: good ones are hard to find, *Communications of the ACM*, **31**, 1192−1201.

Parlett

Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice−Hall, Inc., Englewood Cliffs, New Jersey.

Pereyra

Pereyra, Victor (1978), PASVA3: An adaptive finite-difference FORTRAN program for first order nonlinear boundary value problems, in *Lecture Notes in Computer Science*, **76**, Springer-Verlag, Berlin, 67−88.

Petro

Petro, R. (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **13**, 624.

Petzold

Petzold, L.R. (1982), A description of DASSL: A differential/ algebraic system solver, *Proceedings of the IMACS World Congress*, Montreal, Canada.

Piessens et al.

Piessens, R., E. deDoncker-Kapenga, C.W. Uberhuber, and D.K. Kahaner (1983), *QUADPACK*, Springer-Verlag, New York.

Powell

Powell, M.J.D. (1977), Restart procedures for the conjugate gradient method, *Mathematical Programming*, **12**, 241–254.

Powell, M.J.D. (1978), A fast algorithm for nonlinearly constrained optimization calculations, in *Numerical Analysis Proceedings, Dundee 1977, Lecture Notes in Mathematics*, (edited by G.A. Watson), **630**, Springer-Verlag, Berlin, Germany, 144–157.

Powell, M.J.D. (1983), ZQPCVX a FORTRAN *subroutine for convex quadratic programming*, DAMTP Report NA17, Cambridge, England.

Powell, M.J.D. (1985), On the quadratic programming algorithm of Goldfarb and Idnani, *Mathematical Programming Study*, **25**, 46-61.

Powell, M.J.D. (1988), *A tolerant algorithm for linearly constrained optimization calculations*, DAMTP Report NA17, University of Cambridge, England.

Powell, M.J.D. (1989), TOLMIN: *A fortran package for linearly constrained optimization calculations*, DAMTP Report NA2, University of Cambridge, England.

Pruess and Fulton

Pruess, S. and C.T. Fulton (1993), Mathematical Software for Sturm-Liouville Problems, *ACM Transactions on Mathematical Software*, **17**, *3*, 360–376.

Reinsch

Reinsch, Christian H. (1967), Smoothing by spline functions, *Numerische Mathematik*, **10**, 177–183.

Rice

Rice, J.R. (1983), *Numerical Methods, Software, and Analysis*, McGraw-Hill, New York.

Saad and Schultz

Saad, Y., and M.H. Schultz (1986), GMRES: a generalized minimal residual residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput*., **7**, 856–869.

Schittkowski

Schittkowski, K. (1980), Nonlinear programming codes, *Lecture Notes in Economics and Mathematical Systems*, **183**, Springer-Verlag, Berlin, Germany.

Schittkowski, K. (1983), On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function, *Mathematik Operationsforschung und Statistik, Serie Optimization*, **14**, 197−216.

Schittkowski, K. (1986), NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems, (edited by Clyde L. Monma), Annals of Operations Research, **5**, 485−500.

Schittkowski, K. (1987), *More test examples for nonlinear programming codes*, SpringerVerlag, Berlin, 74.

Schnabel

Schnabel, Robert B. (1985), Finite Difference Derivatives − Theory and Practice, Report, National Bureau of Standards, Boulder, Colorado.

Schreiber and Van Loan

Schreiber, R., and C. Van Loan (1989), A Storage−Efficient *WY* Representation for Products of Householder Transformations, *SIAM J. Sci. Stat. Comp*., Vol. 10, No. 1, pp. 53-57, January (1989).

Scott et al.

Scott, M.R., L.F. Shampine, and G.M. Wing (1969), Invariant Embedding and the Calculation of Eigenvalues for Sturm-Liouville Systems, *Computing*, **4**, 10−23.

Sewell

Sewell, Granville (1982), *IMSL software for differential equations in one space variable*, IMSL Technical Report 8202, IMSL, Houston.

Shampine

Shampine, L.F. (1975), Discrete least-squares polynomial fits, *Communications of the ACM*, **18**, 179−180.

Shampine and Gear

Shampine, L.F. and C.W. Gear (1979), A user’s view of solving stiff ordinary differential equations, *SIAM Review*, **21**, 1−17.

Sincovec and Madsen

Sincovec, R.F., and N.K. Madsen (1975), Software for nonlinear partial differential equations, *ACM Transactions on Mathematical Software*, **1**, #3, 232-260.

Singleton

Singleton, R.C. (1969), Algorithm 347: An efficient algorithm for sorting with minimal storage, *Communications of the ACM*, **12**, 185−187.

Smith

Smith, B.T. (1967), *ZERPOL, A Zero Finding Algorithm for Polynomials Using Laguerre's Method*, Department of Computer Science, University of Toronto.

Smith et al.

Smith, B.T., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler (1976), *Matrix Eigensystem Routines − EISPACK Guide*, Springer-Verlag, New York.

Spang

Spang, III, H.A. (1962), A review of minimization techniques for non-linear functions, *SIAM Review*, **4**, 357−359.

Stewart

Stewart, G.W. (1973), *Introduction to Matrix Computations*, Academic Press, New York.

Stewart, G.W. (1976), The economical storage of plane rotations, *Numerische Mathematik*, **25**, 137−139.

Stoer

Stoer, J. (1985), Principles of sequential quadratic programming methods for solving nonlinear programs, in *Computational Mathematical Programming*, (edited by K. Schittkowski), NATO ASI Series, **15**, Springer-Verlag, Berlin, Germany.

Stroud and Secrest

Stroud, A.H., and D.H. Secrest (1963), *Gaussian Quadrature Formulae*, Prentice-Hall, Englewood Cliffs, New Jersey.

Titchmarsh

Titchmarsh, E. *Eigenfunction Expansions Associated with Second Order Differential Equations*, *Part I*, 2d Ed., Oxford University Press, London, 1962.

Trench

Trench, W.F. (1964), An algorithm for the inversion of finite Toeplitz matrices, J*ournal of the Society for Industrial and Applied Mathematics*, **12**, 515−522.

Walker

Walker, H.F. (1988), Implementation of the GMRES method using Householder transformations, *SIAM J. Sci. Stat. Comput*., **9**, 152–163.

Washizu

Washizu, K. (1968), *Variational Methods in Elasticity and Plasticity*, Pergamon Press, New York.

Watkins and Elsner

Watkins, D.S., and L. Elsner (1990), Convergence of algorithms of decomposition type for the eigenvalue problem, *Linear Algebra and Applications* (to appear).

Weeks

Weeks, W.T. (1966), Numerical inversion of Laplace transforms using Laguerre functions, *J. ACM*, **13**, 419–429.

Wilkinson

Wilkinson, J.H. (1965),*The Algebraic Eigenvalue Problem*, Oxford University

Index

Product Support

Contacting Visual Numerics Support

Users within support warranty may contact Visual Numerics regarding the use of the IMSL Libraries. Visual Numerics can consult on the following topics:

- Clarity of documentation
- Possible Visual Numerics-related programming problems
- Choice of IMSL Libraries functions or procedures for a particular problem
- Evolution of the IMSL Libraries

Not included in these consultation topics are mathematical/statistical consulting and debugging of your program.

Consultation

Contact Visual Numerics Product Support by faxing 713/781-9260 or by emailing:

- for PC support, `pcsupport@houston.vni.com`.
- for non-PC support, `support@houston.vni.com`.

Electronic addresses are not handled uniformly across the major networks, and some local conventions for specifying electronic addresses might cause further variations to occur; contact your local E-mail postmaster for further details.

The following describes the procedure for consultation with Visual Numerics.

1. Include your serial (or license) number
2. Include the product name and version number: IMSL Numerical Libraries Version 3.0
3. Include compiler and operating system version numbers
4. Include the name of the routine for which assistance is needed and a description of the problem.