NSWC TR 86-251

NSWC LIBRARY OF MATHEMATICS SUBROUTINES

BY ALFRED H. MORRIS, JR.
STRATEGIC SYSTEMS DEPARTMENT

APRIL 1987

NAVAL SURFACE WEAPONS CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NSWC TR 86-251	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Surface Weapons Center; 6b. OFFICE SYMBOL *(If applicable)* K33	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State, and ZIP Code)* Dahlgren, VA 22448-5000	7b. ADDRESS *(City, State, and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.

11. TITLE *(Include Security Classification)*
NSWC LIBRARY OF MATHEMATICS SUBROUTINES

12. PERSONAL AUTHOR(S)
Alfred H. Morris, Jr.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
	FROM _TO	1987 April	440

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The NSWC library is a library of general-purpose FORTRAN subroutines that provide a basic computational capability in a variety of mathematical activities. Emphasis has been placed on the transportability of the codes. Subroutines are available in the following areas: Elementary Operations, Geometry, Special Functions, Polynomials, Vectors, Matrices, Large Dense Systems of Linear Equations, Banded Matrices, Sparse Matrices, Eigenvalues and Eigenvectors, ℓ_1 Solution of Linear Equations, Least-Squares Solution of Linear Equations, Optimization, Transforms, Approximation of Functions, Curve Fitting, Surface Fitting, Manifold Fitting, Numerical Integration, Integral Equations, Ordinary Differential Equations, Partial Differential Equations, and Random Number Generation.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED SAME AS RPT. DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Alfred H. Morris, Jr.	22b. TELEPHONE NUMBER *(Include Area Code)* (703) 663-7164	22c. OFFICE SYMBOL K33

FOREWORD

In 1976 work began on the formation of a library of numerical mathematics subroutines for general use at NSWC. The subroutines were to be written in FORTRAN. This report describes the subroutines currently in the library. The report supersedes technical report NSWC TR 84-143. Development of the NSWC library is funded by the Computer and Information Systems Division, Strategic Systems Department, NSWC. The report was administratively reviewed by Robert T. Bevan, Head of the Systems Integration and Networking branch (K33).

Released by:

[illegible]

David B. Colby, Head
Strategic Systems Department

CONTENTS

INTRODUCTION

In 1976 formation of the NSWC library of numerical mathematics subroutines began. The objective was to form a high-quality library of general purpose subroutines that would provide a basic computational capability in a variety of mathematical activities. The routines were to be written in FORTRAN. Even though the routines were intended for use on the CDC 6000-7000 Series computers, emphasis was to be placed on their transportability. This report describes the subroutines currently in the library. Every attempt has been made to ensure the reliability of the codes for any FORTRAN having a single precision arithmetic of 7 or more digits.

All routines are subject to evaluation and possible modification before being accepted for the NSWC library. Primary considerations include the reliability and transportability of the routine, its efficiency and ease of use, and the generality of the routine. In regard to reliability, the major concerns are accuracy, the mathematical stability of the algorithm being employed, and the routine's robustness. The routine is tested, portions of its code are examined, and an assessment is made of the utility and overall performance of the routine. All routines in the library are periodically reviewed for possible improvement. When better routines are obtained the older routines are eliminated.

In regard to transportability, it is clear that machine dependent constants and precision dependent algorithms cannot be avoided. However, machine dependent code and I/O statements are not permitted. For a subroutine or function to be acceptable it is required that the coding adhere to the 1966 and 1977 ANSI FORTRAN standards, the only exception being the following:

> Statements such as REAL A(1), which specify as arrays arguments in the call line of a subroutine or function are acceptable. It is assumed that subscript bound checking for such arrays is not performed by the FORTRAN being used.

It is assumed that the single and double precision floating point arithmetics being used satisfy criteria such as:

(1) All small integers are represented exactly in the floating point arithmetic.
(2) $-x$ is representable as a floating point number if x is a floating point number.
(3) $1/x$ is representable as a floating point number (possibly 0) if x is a nonzero floating point number.

To date, no procedures have been formulated for avoiding the problems that can arise when such assumptions are violated.

The ease of use criterion is of considerable importance. The main purpose of the library is to provide a service to as broad an audience as possible. Thus it is important that duplicate abilities be kept to a minimum, and that the routines be as simple to use and as comprehensive in scope as is practical. Development of software that satisfies the ease of use criterion can be characterized as a packaging problem, the objective being to package mathematical theory and formulae into comprehensive, simple-to-use subroutines. To help meet this criterion, many specialized subroutines are incorporated into the library at a subordinate level, being referenced by simple-to-use driver routines. The driver routines are fully documented in this report, but the supportive routines are only referenced. The policy of referencing supportive code makes it possible to replace or modify the code without bothering the programmer. Also, it significantly simplifies the situation for many users, thereby promoting greater and better use of the software than could otherwise be expected.

The routines in the NSWC library are selected from a variety of sources. All routines are subject to testing before being accepted for the library. The testing serves many purposes, including determination of the accuracy and efficiency of the software, checking for defects in the code, and searching for regions of numerical instability. Because of the theoretical complexity of many of the mathematical activities being computerized, only infrequently will the testing be complete. Normally the testing must be highly selective, being used to locate and examine weaknesses in the algorithm and code. If the precision of a subroutine can be established, then this information is given with the description of the routine. All precision estimates are for the CDC 6000-7000 Series floating point arithmetics. ***The estimates do not include inherent error.***[1] Thus it may occur that the accuracy of a routine is better than the inherent error of the mathematical function that it is computing.

Distribution of Code. The NSWC library subroutines are available for general use, both at NSWC and elsewhere. The library contains no proprietary code.

[1] See "Automatic Error Analysis using Computer Algebraic Manipulation" by David R. Stoutemyer in *ACM Trans. Math Software 3* (1977), pp. 26-43.

SORTING LISTS

Let A be an integer or real array containing $n \geqslant 1$ elements $a_1, \ldots, a_n$. Then the following subroutines are available for reordering the elements of A in increasing order.

CALL ISHELL(A,n)
CALL SHELL(A,n)
CALL SHELL2(A,B,n)

If ISHELL is called then it is assumed that A is an integer array. Otherwise, if SHELL or SHELL2 is called then it is assumed that A is a real array. The elements of A are reordered so that $a_i \leqslant a_{i+1}$ for $i = 1, \ldots, n-1$. In SHELL2 it is assumed that B is also a real array containing n entries. The same permutations are performed on B as on A, thereby reordering the elements of B so as to correspond with the new ordering of A.

Algorithm. The Shell sorting algorithm with increments $(3^k - 1)/2$ is employed.

Programmer. A. H. Morris

Reference. Knuth, D. E., ***The Art of Computer Programming, Vol. 3, Sorting and Searching.*** Addison-Wesley, Reading, Mass., 1973, pp. 84-95.

CUBE ROOT

The following function is available for computing the real cube root of a real number.

CBRT(x)

CBRT(x) $=\sqrt[3]{x}$ for any real x.

Programmer. A. H. Morris

FOUR QUADRANT ARCTANGENT

The function ARTNQ is similar to the ATAN2 function, the differences being that its value lies in the interval $[0,2\pi)$ and its value at the origin is 0. DARTNQ is the double precision counterpart of ARTNQ.

ARTNQ(y,x)
DARTNQ(y,x)

ARTNQ is used if x and y are single precision real values, and DARTNQ is used if x and y are double precision real values. ARTNQ is a single precision function and DARTNQ is a double precision function.

If (x,y) is a point in the plane other than the origin (0,0), let L denote the straight line connecting the points (x,y) and (0,0). Then the function is assigned the value θ where θ is the angle between L and the positive x-axis measured in a counterclockwise direction. Here $0 \leqslant \theta < 2\pi$. Otherwise, if (x,y) is the origin (0,0), then the function is assigned the value 0.

Programmer. Richard Pasto

LENGTH OF A TWO-DIMENSIONAL VECTOR

The following functions are available for computing the length of a real vector (x,y).

CPABS(x,y)
DCPABS(x,y)

CPABS is used if x and y are single precision real values, and DCPABS is used if x and y are double precision values. CPABS is a single precision function and DCPABS is a double precision function. The value of the function is $\sqrt{x^2 + y^2}$.

Programmer. A. H. Morris

SQUARE ROOT OF A DOUBLE PRECISION COMPLEX NUMBER

The following subroutine is available for computing the square root of a double precision complex number.

CALL DCSQRT(Z,W)

Z and W are double precision arrays of dimension 2. It is assumed that Z(1) and Z(2) are the real and imaginary parts of a complex number z. When DCSQRT is called, if $z = 0$ then W(1) and W(2) are set to 0. Otherwise, if $z \neq 0$ then the square root $w = \sqrt{z}$ where $-\pi/2 < \arg(w) \leqslant \pi/2$ is computed and stored in W. W(1) and W(2) contain the real and imaginary parts of w respectively.

Note. Z and W may reference the same storage area.

Programming. DCSQRT calls the function DCPABS. DCSQRT was written by A. H. Morris.

CONVERSION OF POLAR TO CARTESIAN COORDINATES

The following subroutine is available for converting polar coordinates (r,θ) to cartesiar coordinates (x,y).

CALL POCA(r,θ,x,y)

Let (r,θ) be the polar coordinates of a point in the plane and let x,y be variables. When the routine is called, x and y are assigned the values $x = r \cos \theta$ and $y = r \sin \theta$.

CONVERSION OF CARTESIAN TO POLAR COORDINATES

The following subroutine is available for converting cartesian coordinates (x,y) to polar coordinates (r,θ).

CALL CAPO(x,y,r,θ)

Let (x,y) be the cartesian coordinates of a point in the plane and let r,θ be variables. If (x,y) is the origin then CAPO sets $r = \theta = 0$. Otherwise, if (x,y) is a point other than the origin, let L denote the straight line connecting the points (0,0) and (x,y). Then when CAPO is called, r is assigned the value $\sqrt{x^2 + y^2}$ and θ is defined to be the angle between L and the positive x axis. Here $-\pi < \theta \leqslant \pi$.

ROTATION OF AXES

Let (x_1,y_1) be the (cartesian) coordinates for a point in the plane. The following subroutine computes the new coordinates (x_2,y_2) for the point after the x,y axes have been rotated by an angle θ.

CALL ROTA(x_1,y_1,θ,x_2,y_2)

Here x_2 and y_2 are variables. When the routine is called, x_2 and y_2 are assigned the values:

$$x_2 = x_1 \cos \theta + y_1 \sin \theta$$
$$y_2 = -x_1 \sin \theta + y_1 \cos \theta$$

PLANAR GIVENS ROTATIONS

If a and b are real numbers where $a^2 + b^2 \neq 0$, then there is an orthogonal matrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ such that $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$. In this case $r^2 = a^2 + b^2$, $c = a/r$, and $s = b/r$. The matrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ represents what is called a ***Givens rotation.*** Given a and b, the matrix is uniquely defined up to the sign of r. For any real a, let $\text{sgn}(a) = 1$ if $a \geqslant 0$ and $\text{sgn}(a) = -1$ if $a < 0$. If we define $r = \sigma\sqrt{a^2 + b^2}$ where

$$\sigma = \begin{cases} \text{sgn}(a) & \text{if } |a| > |b| \\ \text{sgn}(b) & \text{if } |a| \leqslant |b| \end{cases}$$

then for $r \neq 0$ we note that $|c| > |s|$ implies $c > 0$, and that $|c| \leqslant |s|$ implies $s > 0$. For convenience, when $r = 0$ we set $c = 1$ and $s = 0$.

The value σ is not needed for the construction of a Givens rotation matrix, but its use permits the representation of c and s by a single value z. For each c and s, z is defined as follows:

$$z = \begin{cases} s & \text{if } |s| < c \text{ or } c = 0 \\ 1/c & \text{if } 0 < |c| \leqslant s \end{cases}$$

The mapping $(c,s) \to z$ is 1-1. If the user wishes to reconstruct c and s from z, then this can be done as follows:

If $z = 1$ then set $c = 0$ and $s = 1$.
If $|z| < 1$ then set $c = \sqrt{1-z^2}$ and $s = z$.
If $|z| > 1$ then set $c = 1/z$ and $s = \sqrt{1-c^2}$.

The subroutines SROTG and DROTG are available for computing c, s, r, and z. SROTG is used when a and b are single precision real numbers, and DROTG is used when a and b are double precision numbers.

```
CALL SROTG(AR,BZ,C,S)
CALL DROTG(AR,BZ,C,S)
```

When SROTG is used then AR, BZ, C, and S are single precision real variables. Otherwise, when DROTG is used then AR, BZ, C, and S are double precision variables. On input, AR = a and BZ = b. When the routine terminates AR = r, BZ = z, C = c, and S = s.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. SROTG and DROTG were coded by Charles Lawson (Jet Propulsion Laboratory).

THREE DIMENSIONAL ROTATIONS

If $A = (a_{ij})$ is a 3×3 orthogonal matrix, then A can be represented in the form $A = R_3R_2R_1E$ where

$$R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{pmatrix} \qquad R_2 = \begin{pmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{pmatrix}$$

$$R_3 = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \pm 1 \end{pmatrix}.$$

R_1 represents a rotation around the x-axis, R_2 a rotation around the y-axis, and R_3 a rotation around the z-axis. Since A is orthogonal the determinant $\det(A) = \pm 1$. If $\det(A) = 1$ then E is the identity matrix and A is the combined rotation $R_3R_2R_1$. If $\det(A) = -1$ then A is composed of the rotation $R_3R_2R_1$ and the reflection $E = \mathrm{diag}(1, 1, -1)$. The following subroutine is available for finding the angles $\theta_1, \theta_2, \theta_3$ where $-\pi < \theta_1 \leqslant \pi$, $|\theta_2| \leqslant \pi/2$, and $-\pi < \theta_3 \leqslant \pi$.

CALL ROT3(A,THETA)

THETA is an array of dimension 3 or larger. When ROT3 is called, the angles $\theta_1, \theta_2, \theta_3$ are computed and stored in THETA.

Algorithm. If $a_{11} = a_{21} = 0$ then let $\theta_3 = 0$. Otherwise, let $\theta_3 = \mathrm{ATAN2}(a_{21}, a_{11})$. Then

$$R_3^t A = \begin{pmatrix} r_1 & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = A'$$

where $r_1 = \sqrt{a_{11}^2 + a_{21}^2}$. Also, if $\theta_2 = \mathrm{ATAN2}(a_{31}, r_1)$ then

$$R_2^t A' = \begin{pmatrix} r_2 & a''_{12} & a''_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{pmatrix} = A''$$

where $r_2 = \sqrt{r_1^2 + a_{32}^2}$. Since $r_2 \geqslant 0$, by orthogonality it follows that $r_2 = 1$ and $a''_{12} = a''_{13} = 0$. Finally, if $\theta_1 = \text{ATAN2}(a'_{32}, a'_{22})$ then

$$R_1^t A'' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & r_3 & a''_{23} \\ 0 & 0 & a''_{33} \end{pmatrix}$$

where $r_3 = \sqrt{(a'_{22})^2 + (a'_{32})^2}$. Since $r_3 \geqslant 0$, by orthogonality we obtain $r_3 = 1$, $a''_{23} = 0$, and $a''_{33} = \pm 1$.

Programmer. A. H. Morris

ROTATION OF A POINT ON THE UNIT SPHERE TO THE NORTH POLE

Given the point (x,y,z) where $x^2 + y^2 + z^2 = 1$. Then there exist orthogonal matrices

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{pmatrix} \quad \text{and} \quad R_y = \begin{pmatrix} c_y & 0 & -s_y \\ 0 & 1 & 0 \\ s_y & 0 & c_y \end{pmatrix}$$

such that $R_y R_x \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. R_x represents a rotation about the x-axis and R_y a rotation about the y-axis. The following subroutine is available for obtaining the values c_x, s_x, c_y, s_y.

<u>CALL CONSTR(x,y,z,CX,SX,CY,SY)</u>

CX,SX,CY,SY are variables. When CONSTR is called, these variables are assigned the values c_x, s_x, c_y, s_y.

Programmer. Robert J. Renka (Oak Ridge National Laboratory)

HYPERBOLIC SINE AND COSINE FUNCTIONS

The following subroutine is available for computing the functions $\sinh(x) - x$, $\cosh(x) - 1$, and $\cosh(x) - 1 - x^2/2$ for real x.

CALL SNHCSH(S,C,x,ISW)

S and C are variables, and ISW is an input argument which specifies the functions to be computed. ISW takes the values:

ISW = −1	if only $\sinh(x) - x$ is desired.
ISW = 0	if $\sinh(x) - x$ and $\cosh(x) - 1$ are desired.
ISW = 1	if only $\cosh(x) - 1$ is desired.
ISW = 2	if only $\cosh(x) - 1 - x^2/2$ is desired.
ISW = 3	if $\sinh(x) - x$ and $\cosh(x) - 1 - x^2/2$ are desired.

S is assigned the value $\sinh(x) - x$ if this function is requested. When $\cosh(x) - 1$ or $\cosh(x) - 1 - x^2/2$ is computed then the value is stored in C.

Precision. For all x, $\sinh(x) - x$ has a relative error less than 2.3E-14, $\cosh(x) - 1$ has a relative error less than 2.2E-14, and $\cosh(x) - 1 - x^2/2$ has a relative error less than 3.8E-14.

Programming. Written by A. K. Cline and R. J. Renka (University of Texas at Austin). Modified by A. H. Morris.

EXPONENTIALS

The following function is available for computing $e^x - 1$.

REXP(x)

$\text{REXP}(x) = e^x - 1$ for all real x.

Algorithm. For $|x| \leqslant .15$ a rational minimax approximation is used. Otherwise, $e^x - 1$ is computed using the function EXP.

Precision. REXP is accurate to within 2 units of the 14th significant digit.

Programmer. A. H. Morris

LOGARITHMS

The following functions are available for computing $\ln(1 + a)$ and $\phi(x) = x - 1 - \ln(x)$.

ALNREL(a)

$\text{ALNREL}(a) = \ln(1 + a)$ for $a > -1$.

Algorithm. For $|a| \leqslant .375$ a rational minimax approximation is used. Otherwise, $\ln(1 + a)$ is computed using the function ALOG.

Precision. ALNREL(a) is accurate to within 2 units of the 14th significant digit when $\text{ALNREL}(a) \neq 0$.

Programmer. A. H. Morris

RLOG(x)

$\text{RLOG}(x) = \phi(x)$ for $x > 0$.

Algorithm. If $|x - 1| \leqslant .18$ then $\phi(x) = 2r^2\ [1/(1 - r) - rw]$ is applied where $r = (x - 1)/(x + 1)$ and w is a rational minimax approximation of $\sum_{n=0}^{\infty} r^{2n}/(2n + 3)$. If $.61 \leqslant x < .82$ or $1.18 < x \leqslant 1.57$ then x is reduced to the interval $[.82, 1.18]$ using $\phi(cz) = \phi(c) + \phi(z) + (c - 1)(z - 1)$. Otherwise, $\phi(x)$ is computed using the function ALOG.

Precision. RLOG is accurate to within 3 units of the 14th significant digit.

Programmer. A. H. Morris

THE CONVEX HULL FOR A FINITE PLANAR SET

If $(x_1,y_1),\dots,(x_m,y_m)$ are m distinct points in the plane, then the following subroutine is available for finding the smallest convex polygon which with its interior contains the points.

CALL HULL (X,Y,m, BX,BY, k,VX,VY, n)

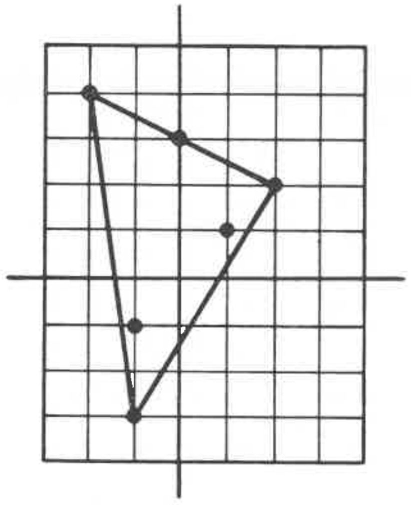
It is assumed that $m \geq 2$. X and Y are arrays containing the abcissas $x_1,\dots,x_m$ and ordinates $y_1,\dots,y_m$ respectively. When HULL is called the points are reordered so that $y_1 \leq \cdots \leq y_m$. This reordering is permanent. Thus X and Y may be modified when the routine terminates.

BX and BY are arrays of dimension m + 1 or larger, and k is a variable. When HULL terminates, BX and BY will contain the abcissas and ordinates of the points (x_i,y_i) which lie on the desired convex polygon, and k will equal the number of points stored in BX and BY. If BX and BY contain the abcissas $x'_1,\dots,x'_k$ and ordinates $y'_1,\dots,y'_k$, then the points (x'_i,y'_i) are indexed in the order they occur when traversing the convex polygon in a counterclockwise manner. The last point of the sequence, namely (x'_k,y'_k), will be the same as the first point of the sequence; i.e., $x'_k = x'_1$ and $y'_k = y'_1$.

VX and VY are arrays of dimension m + 1 or larger, and n is a variable. When HULL terminates, VX and VY will contain the abcissas and ordinates of the vertices of the desired convex polygon, and n will equal the number of points stored in VX and VY. If VX and VY contain the abscissas $x''_1,\dots,x''_n$ and ordinates $y''_1,\dots,y''_n$, then the vertices (x''_i,y''_i) are indexed in the order they occur when traversing the convex polygon in a counterclockwise manner. The last vertex, namely (x''_n,y''_n), will be the same as the first vertex; i.e., $x''_n = x''_1$ and $y''_n = y''_1$.

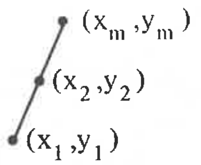
Example. Assume that we are given the points (-1,-3), (1,1), (0,3), (2,2), (-2,4), (-1,-1). When HULL is called X and Y are reordered so that:

X contains	-1,-1,1,2,0,-2
Y contains	-3,-1,1,2,3,4
BX contains	-1,2,0,-2,-1
BY contains	-3,2,3,4,-3
VX contains	-1,2,-2,-1
VY contains	-3,2,4,-3



Remarks. If the points all lie on a single straight line, then either the ordinates will all be different or they will all be the same. If they are different, then after the points are reordered $y_1 < \cdots < y_m$. Otherwise if $y_1 = \cdots = y_m$ then the routines will reorder the points so that $x_1 < \cdots < x_m$. In either case

BX will contain	$x_1, \ldots, x_m, x_1$
BY will contain	$y_1, \ldots, y_m, y_1$
VX will contain	x_1, x_m, x_1
VY will contain	y_1, y_m, y_1



and the variables k and n will be assigned the values $k = m + 1$ and $n = 3$.

Programming. HULL was written by Richard K. Hageman and modified by A. H. Morris. The routine calls the function SPMPAR and the subroutines SHELL and SHELL2.

Reference. DiDonato, A. R., and Hageman R. K., ***An Algorithm for Finding the Convex Hull of a Finite Point Set in the Plane***, Technical Note TN 79-42, Naval Surface Weapons Center, Dahlgren, Virginia, 1979.

AREAS OF PLANAR POLYGONS

Given a sequence of points $\nu_i = (x_i,y_i)$ $(i = 1,\ldots,n + 1)$ where $n \geqslant 1$ and $\nu_{n+1} = \nu_1$. Let τ denote the polygon whose boundary $\partial\tau$ is a polygonal line which begins at point ν_1, traverses the points ν_i in the order that they are indexed, and is the straight line segment connecting ν_i to ν_{i+1} for $i = 1,\ldots,n$. Then the function PAREA is available for computing $A(\tau) = \iint_\tau dxdy$. If the boundary $\partial\tau$ is a positively (negatively) oriented simple closed curve, then $A(\tau)$ is positive (negative) and $|A(\tau)|$ = the area of τ. However, $\partial\tau$ need not be simple. It may be self-intersecting or have overlapping line segments.

PAREA(X,Y,NB)

X and Y are arrays containing the abscissas $x_1,\ldots,x_{NB}$ and ordinates $y_1,\ldots,y_{NB}$ respectively. The argument NB may have the value n or n + 1. Since $\nu_{n+1} = \nu_1$, x_{n+1} and y_{n+1} are not required to appear in X and Y. PAREA(X,Y,NB) is assigned the value $A(\tau)$.

Programmer. A. H. Morris

Reference. DiDonato, A. R., and Hageman, R. K., ***Computation of the Integral of the Bivariate Normal Distribution over Arbitrary Polygons,*** Technical Report TR 80-166, Naval Surface Weapons Center, Dahlgren, Virginia, 1980.

ERROR FUNCTION

For any complex z the error function is defined by

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\, dt$$

and its complement by erfc(z) = 1 − erf(z). The subroutine CERF is available for computing erf(z) and erfc(z) when z is complex, and the functions ERF, ERFC, and ERFC1 are available for computing erf(z) and erfc(z) when z is real.

ERF(x)

ERF(x) = erf(x) for any real x.

Precision. ERF maintains accuracy to within 2 units of the 14th significant digit.

Programmer. A. H. Morris

Reference. Cody, W. J., "Rational Chebychev Approximations for the Error Function," ***Mathematics of Computation 23*** (1969), pp. 631-637.

ERFC(x)

ERFC(x) = erfc(x) for any real x.

Precision. ERFC(x) is accurate to within 2 units of the 14th significant digit for $x \leqslant 1$.

Programmer. A. H. Morris

Reference. Cody, W. J., "Rational Chebychev Approximations for the Error Function," ***Mathematics of Computation 23*** (1969), pp. 631-637.

ERFC1(IND,x)

IND is an integer and x a real number. ERFC1(IND,x) = erfc(x) when IND = 0 and ERFC1(IND,x) = e^{x^2} erfc(x) when IND $\neq$ 0.

Precision. ERFC1(IND,x) is accurate to within 2 units of the 14th significant digit when IND = 0 and $x \leqslant 1$, or when IND $\neq$ 0 and $x \geqslant -1$.

Programmer. A. H. Morris

Reference. Cody, W. J., "Rational Chebychev Approximations for the Error Function," *Mathematics of Computation 23* (1969), pp. 631–637.

CALL CERF(MO,z,w)

MO is an integer, z a complex number, and w a complex variable. When CERF is called, w is assigned the value erf(z) if MO = 0 and the value erfc(z) if MO $\neq$ 0.

Algorithm. For $z = x + iy$ where $x \geqslant 0$, if z satisfies $|z| \leqslant 1$ or both of the inequalities $1 < |z| < \sqrt{38}$ and $x^2 - y^2 + .256\, x^2 y^2 \leqslant 0$, then the series

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n!(2n+1)}$$

is used. If $1 < |z| < \sqrt{38}$ and $x^2 - y^2 + .256\, x^2 y^2 > 0$ then

$$\operatorname{erf}(z) = 1 - \frac{ze^{-z^2}}{\sqrt{\pi}} \sum_{n=1}^{18} \frac{r_n}{z^2 + \lambda_n}$$

is employed. λ_n and r_n are the poles and residues of the rational function approximation for the complex Fresnel integral E(z) given in the reference. The error function is related to E(z) by $\operatorname{erf}(z) = 1 - i\sqrt{2}\, E(-z^2)$ for $|\arg(z)| < \pi/2$. If $|z| \geqslant \sqrt{38}$ and $x \geqslant .01$ then erf(z) is computed by the asymptotic expansion $\operatorname{erf}(z) = 1 - \psi(z)$. Here

$$\psi(z) = \frac{e^{-z^2}}{\sqrt{\pi}} \left[\frac{1}{z} + \sum_{n=1}^{m} (-1)^n \frac{1 \cdot 3 \cdots (2n-1)}{2^n z^{2n+1}} \right] \qquad |\arg(z)| < \frac{\pi}{2}$$

where m = 37. Otherwise, if $|z| \geqslant \sqrt{38}$ and $0 \leqslant x < .01$ then the modified asymptotic expansion $\operatorname{erf}(z) = -\psi(z)$ is employed. When $x < 0$ then the relation $\operatorname{erf}(-z) = -\operatorname{erf}(z)$ is applied.

Precision. CERF is accurate to within 4 units of the 14th significant digit when z is real, and to within 5 units of the 13th significant digit when Re(z) = 0.

Programming. Written by A. V. Hershey. Modified by A. H. Morris.

Reference. Hershey, A. V., *Approximation of Functions by Sets of Poles*, Technical Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, July 1971.

NORMAL DISTRIBUTION FUNCTION

For any real x the normal probability distribution function P(x) is defined by

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$$

and its complementary function by Q(x) = 1 − P(x). The following function is available for computing P(x) and Q(x).

<u>PNDF(x,IND)</u>

IND is an integer and x a real number. If IND = 0 then

$$\text{PNDF}(x,0) = \begin{cases} P(x) & \text{if } x \geqslant -8 \\ \dfrac{P'(x)}{P(x)} & \text{if } x < -8 \end{cases}$$

where P′(x) is the derivative of P(x). Otherwise, if IND ≠ 0 then

$$\text{PNDF}(x,\text{IND}) = \begin{cases} Q(x) & \text{if } x \leqslant 8 \\ \dfrac{Q'(x)}{Q(x)} & \text{if } x > 8 \end{cases}$$

where Q′(x) is the derivative of Q(x).

Algorithm. The identities $P(x) = \frac{1}{2}\,\text{erfc}(-x/\sqrt{2})$ and $Q(x) = \frac{1}{2}\,\text{erfc}(x/\sqrt{2})$ are used.

Programming. PNDF calls the function ERFC1. These functions were written by A. H. Morris.

COMPLEX FRESNEL INTEGRAL

For any complex z not on the positive real axis the complex Fresnel integral E(z) can be defined by

$$E(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \frac{e^t}{\sqrt{t}}\, dt.$$

Here it is assumed that $0 < \arg(z) < 2\pi$ and $\arg(\sqrt{z}) = 1/2 \arg(z)$. E(z) can be extended to the positive real axis by letting $0 \leqslant \arg(z) < 2\pi$. Then $\operatorname{erf}(z) = 1 - i\sqrt{2}\, E(-z^2)$ for $-\pi/2 \leqslant \arg(z) < \pi/2$ where erf(z) is the error function and $\arg(-z^2) = \pi + 2\arg(z)$. The following subroutine is available for computing E(z).

<u>CALL CFRNLI(MO,z,w)</u>

MO is an integer, z a complex number, and w a complex variable. When CFRNLI is called, w is assigned the value E(z) if MO = 0 and the value $e^{-z}E(z)$ if MO ≠ 0.

Algorithm. If $z = x + iy$ satisfies $|z| \leqslant 1$ or both of the inequalities $1 < |z| < 38$ and $-x + .064\, y^2 \leqslant 0$, then the series

$$E(z) = -\frac{i}{\sqrt{2}} + \sqrt{2z/\pi} \sum_{n=0}^{\infty} \frac{z^n}{n!(2n+1)}$$

is used. If $1 < |z| < 38$ and $-x + .064\, y^2 > 0$ then

$$E(z) = \sqrt{\frac{z}{2\pi}}\, e^z \sum_{n=1}^{18} \frac{r_n}{z - \delta_n}$$

is employed. If $|z| \geqslant 38$ and either $x \leqslant 0$ or $|\mathrm{Im}\sqrt{2z/\pi}| \geqslant .008$, then E(z) is computed by the asymptotic expansion $E(z) = \psi(z)$. Here

$$\psi(z) = \frac{e^z}{\sqrt{2\pi z}} \left[1 + \sum_{n=1}^{m} \frac{1 \cdot 3 \cdots (2n-1)}{(2z)^n} \right]$$

where m = 37. Otherwise, if $|z| \geqslant 38$ and $|\mathrm{Im}\sqrt{2z/\pi}| < .008$ then the modified asymptotic expansion $E(z) = -i/\sqrt{2} + \psi(z)$ is employed.

Precision. For real z, E(z) is accurate to within 4 units of the 14th significant digit when z is negative, and to within 5 units of the 13th significant digit when z is positive.

Programming. Written by A. V. Hershey. Modified by A. H. Morris.

Reference. Hershey, A. V., ***Approximation of Functions by Sets of Poles***, Technical Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, July 1971.

REAL FRESNEL INTEGRALS

For any complex z the Fresnel integrals C(z) and S(z) can be defined by

$$C(z) = \int_0^z \cos\left(\frac{\pi}{2} t^2\right) dt$$

$$S(z) = \int_0^z \sin\left(\frac{\pi}{2} t^2\right) dt .$$

The following subroutine is available for computing C(z) and S(z) when z is real.

CALL FRNL(x,C,S)

The argument x may be any real number. C and S are variables. When FRNL is called C is assigned the value C(x) and S is assigned the value S(x).

Algorithm. If $0 \leqslant x < 1.65$ then x^{-1} C(x) and x^{-3} S(x) are computed by minimax polynomial approximations. Otherwise, if $x \geqslant 1.65$ then the relations

$$C(x) = \frac{1}{2} + f(x) \sin \frac{\pi}{2} x^2 - g(x) \cos \frac{\pi}{2} x^2$$

$$S(x) = \frac{1}{2} - f(x) \cos \frac{\pi}{2} x^2 - g(x) \sin \frac{\pi}{2} x^2$$

are invoked. For $1.65 \leqslant x < 6$, xf(x) and $x^3g(x)$ are computed by rational minimax approximations. Otherwise, for $x \geqslant 6$ the auxiliary functions f(x) and g(x) are computed by the asymptotic expansions:

$$f(x) = \frac{1}{\pi x}\left[1 + \sum_{i=1}^{m} (-1)^i \frac{1\cdot 3\cdots(4i-1)}{(\pi x^2)^{2i}}\right]$$

$$g(x) = \frac{1}{\pi x} \sum_{i=0}^{m} (-1)^i \frac{1\cdot 3\cdots(4i+1)}{(\pi x^2)^{2i+1}}$$

Here m = 5. If $x < 0$ then the relations $C(-x) = -C(x)$ and $S(-x) = -S(x)$ are applied.

Precision. If $|x| < 1.65$ then FRNL is accurate to within 3 units of the 14th significant digit. Otherwise, if $|x| \geqslant 1.65$ then FRNL is accurate to within 1 unit of the 14th significant digit.

Programming. FRNL calls the functions SPMPAR and I1MACH. FRNL was written by A. H. Morris.

EXPONENTIAL INTEGRAL FUNCTION

For any complex $z \neq 0$ not on the positive real axis the exponential integral function Ei(z) is defined by

$$\mathrm{Ei}(z) = \int_{-\infty}^{z} \frac{e^t}{t}\, dt.$$

Ei(z) is an analytic function. If z is replaced by $-z$ and t by $-t$ we obtain the related function

$$E_1(z) = -\,\mathrm{Ei}(-z) = \int_z^{\infty} \frac{e^{-t}}{t}\, dt$$

which is defined for all $z \neq 0$ not on the negative real axis. It can be verified that

$$\mathrm{Ei}(z) = \nu + \ln(-z) + \sum_{n=1}^{\infty} \frac{z^n}{n \cdot n!}$$

everywhere in the plane cut along the positive real axis.[1] Thus the values of Ei(z) on the upper and lower edges of the cut are

$$\mathrm{Ei}(x \pm i0) = \mathrm{ei}(x) \mp \pi i$$

where ei(x) is the real function defined by

$$\mathrm{ei}(x) = \nu + \ln x + \sum_{n=1}^{\infty} \frac{x^n}{n \cdot n!}$$

for $x > 0$. ei(x) is also known as the exponential integral function. It is a strictly monotone function having a zero at the point x_0 = .37250 74107 81367. ei(x) has the integral representation

$$\mathrm{ei}(x) = \lim_{\epsilon \to 0} \left[\int_{-\infty}^{-\epsilon} \frac{e^t}{t}\, dt + \int_{\epsilon}^{\infty} \frac{e^t}{t}\, dt \right]$$

for all $x > 0$. Ei(z), $E_1(z)$, and ei(x) may be computed by the subroutines CEXPLI and EXPLI. CEXPLI handles complex arguments and EXPLI handles real arguments.

<u>CALL CEXPLI(MO,z,w)</u>

MO may be 0 or 1, $z \neq 0$ is a complex number not on the positive real azis, and w is a complex variable. When CEXPLI is called, if MO = 0 then w is assigned the value Ei(z). Otherwise, if MO = 1 then w is assigned the value $e^{-z}\mathrm{Ei}(z)$.

[1] ν is the Euler constant .57721 56649 ... and ln z denotes the single-valued branch of the logarithm defined by $\ln z = \ln |z| + i \arg(z)$ where $|\arg(z)| < \pi$.

Precision. For real z, $e^{-z}Ei(z)$ is accurate to within 3 units of the 14th significant digit if $-9 \leqslant z < 0$, and accurate to within 4 units of the 14th significant digit if $z < -9$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Approximations of Functions by Sets of Poles***, Technical Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

CALL EXPLI(MO,x,y,IERR)

MO may have the values 1,2, or 3. The argument x is a non-zero real number, and y is a real variable. When EXPLI is called, if MO = 1 then y is assigned the value Ei(x) for $x < 0$ and the value ei(x) for $x > 0$. If MO = 2 then it is assumed that $x > 0$. In this case y is assigned the value $E_1(x)$. Otherwise, if MO = 3 then y is assigned the value $e^{-x}Ei(x)$ for $x < 0$ and the value $e^{-x}ei(x)$ for $x > 0$.

Error Return. IERR is an integer variable that is set by the routine. If no input errors occur and underflow/overflow do not occur, then IERR is assigned the value 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	if underflow occurs
IERR = 2	if overflow occurs
IERR = 3	if x = 0
IERR = 4	if MO = 2 and $x < 0$

Underflow occurs if MO = 1 and $x < -669.31$, or if MO = 2 and $x > 669.31$. When it occurs, y will have the value 0. Overflow can occur only when MO = 1 and $x > 748.28$. If overflow occurs then y will not be assigned a value. The settings IERR = 3 and 4 indicate input errors. In these two cases y is not assigned a value.

Precision. e^{-x} Ei(x) is accurate to within 2 units of the 14th significant digit for $-10 < x < 0$, and to within 1 unit of the 14th significant digit for $x \leqslant -10$. $e^{-x}ei(x)$ is accurate to within 2 units of the 14th significant digit for $0 < x < 12$, and to within 1 unit of the 14th significant digit for $x \geqslant 12$. The only exceptions are for the intervals $.408 < x < .413$ and $|x - x_0| < 4 \cdot 10^{-13}$ where x_0 is the zero of ei(x). Accuracy is maintained to within 5 units of the 14th significant digit for $.408 < x < .413$, and to within 2 units of the 14th significant digit for $|x - x_0| < 4 \cdot 10^{-13}$.

Programming. EXPLI belongs to the FUNPACK package of routines developed at Argonne National Laboratory. The routine was modified by A. H. Morris. EXPLI employs the functions EXPARG and I1MACH.

References.

(1) Cody, W. J., and Thacher, H. C., "Rational Chebychev Approximations for the Exponential Integral E_1 (x)", ***Mathematics of Computation 22*** (1968), pp. 641-649.

(2) ______________,"Chebychev Approximations for the Exponential Integral Ei(x)", ***Mathematics of Computation 23*** (1969), pp. 289-303.

SINE AND COSINE INTEGRAL FUNCTIONS

For any complex z the sine integral and cosine integral functions Si(z) and Cin(z) are defined by

$$\mathrm{Si}(z) = \int_0^z \frac{\sin t}{t}\, dt$$

$$\mathrm{Cin}(z) = \int_0^z \frac{1 - \cos t}{t}\, dt.$$

These are entire functions. The following functions are available for computing Si(z) and Cin(z) when z is real.

<u>SI(x)</u>

SI(x) = Si(x) for all real x.

Precision. SI is accurate to within 2 units of the 14th significant digit.

Programming. SI calls the function SPMPAR. SI was written by Donald E. Amos and Sharon L. Daniel (Sandia Laboratories), and modified by A. H. Morris.

<u>CIN(x)</u>

CIN(x) = Cin(x) for all real x.

Precision. CIN is accurate to within 2 units of the 14th significant digit.

Programming. CIN calls the function SPMPAR. CIN was written by Donald E. Amos and Sharon L. Daniel (Sandia Laboratories), and modified by A. H. Morris.

GAMMA FUNCTION

For any complex z the gamma function $\Gamma(z)$ is defined by

$$\frac{1}{\Gamma(z)} = ze^{Cz} \prod_{n=1}^{\infty} \left(1 + \frac{z}{n}\right) e^{-z/n}$$

where C is Euler's constant. $\Gamma(z)$ has a simple pole at $z = 0,-1,-2, \dots$ and $1/\Gamma(z)$ is an entire function. If $\mathrm{Re}(z) > 0$ then

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t}\, dt .$$

The subroutine CGAMMA is available for computing $\Gamma(z)$ and $\ln \Gamma(z)$ when z is complex, and the functions GAMMA and GAMLN are available for computing $\Gamma(z)$ and $\ln \Gamma(z)$ when z is real. CGAMMA is less efficient and accurate than GAMMA and GAMLN.

CALL CGAMMA(MO,z,w)

MO is an integer, z a complex number satisfying $z \neq 0, -1, -2, \dots$, and w a complex variable. When CGAMMA is called, w is assigned the value $\Gamma(z)$ if MO = 0 and the value $\ln \Gamma(z)$ if MO $\neq$ 0.

Programmer. A. H. Morris

Reference. Kuki, Hirondo, "Complex Gamma Function with Error Control," ***Comm. ACM 15*** (1972), pp. 262-267.

GAMMA(x)

The argument x is a real number. If $\Gamma(x)$ can be computed then GAMMA(x) is assigned the value $\Gamma(x)$. Otherwise, if $\Gamma(x)$ cannot be computed, then GAMMA(x) is set to 0.

Remark. On the CDC 6000-7000 series computers $\Gamma(x)$ can be evaluated only when $x \neq 0, -1, -2, \dots$ and $|x| \leqslant 177.8$.

Algorithm. If $|x| < 15$ then x is reduced to the interval $[1,2)$ by $\Gamma(a + 1) = a\,\Gamma(a)$, and a rational minimax approximation is employed. If $x < -15$ then

$$\Gamma(x) = \frac{(-1)^{n+1}\pi}{\sin(\pi\lambda)\,|x|\,\Gamma(|x|)}$$

is applied. Here $|x| = n + \lambda$ where n is the largest integer less than $|x|$. For $x \geqslant 15$

$$\ln \Gamma(x) = (x - 1/2)\ln x - x + \frac{1}{2}\ln 2\pi + g(x)$$

is computed where g(x) is a minimax approximation. The function g(x) is evaluated in single precision, and a double precision value is obtained for ln x. This yields a double precision value for ln Γ(x). If $\ln \Gamma(x) = \alpha + \delta$ where α is the leading portion of ln Γ(x), then Γ(x) is set to $e^{\alpha}(1 + \delta)$. This is permissible since $1 + \delta$ are the only terms of the Taylor series expansion for e^{δ} that are significant.

The logarithm ln x is evaluated as follows: Let n be the largest integer less than or equal to x, and let $t = (x - n)/(x + n)$. Then $x = n(1 + t)/(1 - t)$ so that $\ln x = \ln n + \ln[(1 + t)/(1 - t)]$. Also $0 \leqslant t < 1/(2n)$. The function $\ln[(1 + t)/(1 - t)]$ is computed by a polynomial minimax approximation in single precision, and the value ln n is stored in double precision.

Precision. If $0 < x < 2$ then GAMMA(x) is accurate to within 2 units of the 14th significant digit. If $x \geqslant 15$ then GAMMA(x) is accurate to within 3 units of the 14th significant digit. Otherwise, GAMMA(x) is accurate to within 5 units of the 14th significant digit.

Programming. GAMMA calls the functions GLOG and EXPARG. These functions were written by A. H. Morris. The function I1MACH is also used.

GAMLN(x)

GAMLN(x) = ln Γ(x) for all positive real x.

Algorithm. Rational minimax approximations are used for ln Γ(1 + a)/a and ln Γ(2 + a)/a when $-.2 \leqslant a \leqslant .6$ and $-.4 \leqslant a \leqslant .25$. For $x \geqslant 15$

$$\ln \Gamma(x) = (x - 1/2)\ln x - x + \frac{1}{2}\ln 2\pi + g(x)$$

is computed where g(x) is a minimax approximation.

Precision. GAMLN(x) is accurate to within 2 units of the 14th significant digit when $\text{GAMLN}(x) \neq 0$.

Programming. GAMLN employs the function GAMLN1. These functions were written by A. H. Morris.

DIGAMMA FUNCTION

For any complex $z \neq 0, -1, -2, \ldots$, the digamma (or psi) function $\psi(z)$ is defined by

$$\psi(z) = \Gamma'(z)/\Gamma(z)$$

where $\Gamma(z)$ is the gamma function. The subroutine CPSI is available for computing $\psi(z)$ when z is complex and the function PSI is available for computing $\psi(z)$ when z is real.

CALL CPSI(z,w)

The argument z is a complex number satisfying $z \neq 0, -1, -2, \ldots$, and w is a complex variable. When CPSI is called, w is assigned the value $\psi(z)$.

Algorithm. If $z = x + iy$ satisfies $x \geqslant 0$ and $|z| \geqslant 8$, then the asymptotic expansion

$$\psi(z) = \ln z - \frac{1}{2z} - \sum_{m=1}^{\infty} \frac{B_{2m}}{2mz^{2m}}$$

is employed. Otherwise, if $x \geqslant 0$ then the smallest nonnegative integer n is found for which $|z + n| \geqslant 8$, and the relation

$$\psi(z) = -\sum_{j=0}^{n-1} \frac{1}{z+j} + \psi(z+n)$$

is applied. When $x < 0$ then

$$\psi(z) = \psi(1-z) - \pi \cot(\pi z)$$

is also invoked.

Programmer. A. H. Morris

PSI(x)

The argument x is a real number. If $x \neq 0,-1,-2,\ldots$ then PSI(x) is assigned the value $\psi(x)$. Otherwise, PSI(x) is assigned the value 0.

Precision. For $x > 0$ PSI(x) is accurate to within 2 units of the 14th significant digit.

Programming. PSI belongs to the FUNPACK package of routines developed at Argonne National Laboratory. The format of the routine was modified by A. H. Morris. The function SPMPAR is used.

Reference. Cody, W. J., Strecok, A. J., and Thacher, H. C., "Chebychev Approximations for the Psi Function," ***Mathematics of Computation*** *27* (1973), pp. 123–127.

LOGARITHM OF THE BETA FUNCTION

For a,b $>$ 0 the beta function B(a,b) can be defined by

$$B(a,b) = \int_0^1 t^{a-1} (1-t)^{b-1}\, dt.$$

From this it follows that $B(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ where $\Gamma(a)$ is the gamma function. The following function is available for computing ln B(a,b).

BETALN(a,b)

BETALN(a,b) = ln B(a,b) for a,b $>$ 0.

Precision. BETALN(a,b) is accurate to within 4 units of the 14th significant digit when a,b $\geqslant$ 1 and BETALN(a,b) $\neq$ 0. In particular, when a,b $\geqslant$ 15, BETALN(a,b) is accurate to within 2 units of the 14th significant digit.

Programming. BETALN employs the functions ALNREL, ALGDIV, BCORR, GAMLN, GAMLN1, and GSUMLN. These functions were written by A. H. Morris.

INCOMPLETE GAMMA RATIO FUNCTIONS

For $a \geqslant 0$ and $x \geqslant 0$, where a and x are not both 0, let P(a,x) and Q(a,x) denote the functions defined by

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

$$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt .$$

Then $0 \leqslant P(a,x) \leqslant 1$ and $P(a,x) + Q(a,x) = 1$. The following subroutine is available for computing P(a,x) and Q(a,x).

CALL GRATIO(a,x,P,Q,i)

P and Q are variables. GRATIO assigns P the value P(a,x) and Q the value Q(a,x). The argument i may be any integer. This argument specifies the desired accuracy of the results. If $i = 0$ then the user is requesting as much accuracy as possible (up to 14 significant digits). Otherwise, if $i = 1$ then accuracy is requested to within 1 unit of the 6th significant digit, and if $i \neq 0,1$ then accuracy is requested to within 1 unit of the 3rd significant digit.

Error Return. P is assigned the value 2 when a or x is negative, when $a = x = 0$, or when P(a,x) and Q(a,x) are indeterminant. P(a,x) and Q(a,x) are indeterminant when $x \approx a$ and a is exceedingly large. On the CDC 6000-7000 series computers this occurs when $|x/a - 1| \leqslant 10^{-14}$ and $a \geqslant 6.6E25$.

Programming. GRATIO calls the functions ERF, ERFC1, REXP, RLOG, GAMMA, GAM1, and SPMPAR. GAMMA employs the functions GLOG, EXPARG, and I1MACH. GRATIO was written by A. H. Morris.

Reference. DiDonato, A. R. and Morris, A. H., "Computation of the Incomplete Gamma Function Ratios and their Inverse," ***ACM Trans. Math Software 12*** (1986).

INCOMPLETE BETA FUNCTION

For $a,b > 0$ and $0 \leqslant x \leqslant 1$ the incomplete beta function is defined by

$$I_x(a,b) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}dt$$

where $B(a,b)$ is the beta function. Then we note that $0 \leqslant I_x(a,b) \leqslant 1$ and

$$\lim_{a \to 0} I_x(a,b) = 1 \quad \text{for } x \neq 0$$

$$\lim_{b \to 0} I_x(a,b) = 0 \quad \text{for } x \neq 1.$$

These limits permit $I_x(a,b)$ to be defined to be 1 when $a = 0$ and $b \neq 0$, $x \neq 0$, and for $I_x(a,b)$ to be defined to be 0 when $b = 0$ and $a \neq 0$, $x \neq 1$. The subroutine BRATIO is available for computing $I_x(a,b)$ for arbitrary $a,b \geqslant 0$, and the subroutine ISUBX is available for computing $I_x(a,b)$ for the highly specialized case when a and b are integers or half-integers.

CALL BRATIO(a,b,x,y,W,W1,IERR)

It is assumed that $a \geqslant 0$, $b \geqslant 0$, $0 \leqslant x \leqslant 1$, and $y = 1 - x$. W, W1, and IERR are variables. If no input errors are detected then IERR is set to 0, W is assigned the value $I_x(a,b)$, and W1 is assigned the value $1 - I_x(a,b)$.

Error Return. When an input error is detected, then W and W1 are assigned the value 0 and IERR is set to one of the following values:

IERR = 1	if $a < 0$ or $b < 0$
IERR = 2	if $a = b = 0$
IERR = 3	if $x < 0$ or $x > 1$
IERR = 4	if $y < 0$ or $y > 1$
IERR = 5	if $x + y \neq 1$
IERR = 6	if $x = a = 0$
IERR = 7	if $y = b = 0$

The setting IERR = 5 occurs when x and y are not given to sufficient accuracy so that $x + y$ adequately approximates 1 in the floating point arithmetic being used.

Programming. BRATIO was formulated by A. R. DiDonato and A. H. Morris. BRATIO employs the subroutines BGRAT and GRAT1, and the functions ALGDIV, ALNREL, BASYM, BCORR, BETALN, BFRAC, BPSER, BRCOMP, BUP, ERF, ERFC1, GAMLN, GAMLN1, GAMMA, GSUMLN, RCOMP, GAM1, and RLOG. Also GAMMA calls the functions GLOG and EXPARG. These routines and functions were written by A. H. Morris. The functions SPMPAR and I1MACH are also used.

<u>CALL ISUBX(a,b,x,W,IERR,EPS)</u>

It is assumed that a, b, and x satisfy the following restrictions:

(1) $a > 0$, $b > 0$, and $x \geq 0$

(2) $a \geq ½$, $½ \leq b \leq 70$, and $x \leq 1$

(3) a and b are integers or half-integers

EPS specifies the (absolute) accuracy that is desired. W is a real variable and IERR an integer variable. When ISUBX is called, if there are no input errors then W is assigned the value $I_x(a,b)$ and IERR is assigned the value 1.

Error Return. If an error is detected then IERR is assigned one of the following values:

IERR = 2	if restrictions (1) are violated.
IERR = 3	if restrictions (2) are violated or a is too large.
IERR = 4	if restrictions (3) are violated.

Also W is assigned the value 0.

Remarks. ISUBX was designed for a maximum precision $EPS = 10^{-10}$

Programming. ISUBX employs the functions ALGDIV, ALNREL, BLND, I1MACH, and LOGAM. ISUBX was written by A. H. Morris.

Reference. DiDonato, A.R. and Jarnagin, M.P., "The Efficient Calculation of the Incomplete Beta-function Ratio for Half-Integer Values of the parameters a, b," ***Math. Comp. 21*** (1967), pp. 652-662.

BESSEL FUNCTION $J_\nu(z)$

If ν is complex then $J_\nu(z)$ is defined by

$$J_\nu(z) = \sum_{k=0}^{\infty} \frac{(-1)^k (z/2)^{\nu+2k}}{k!\,\Gamma(\nu+k+1)}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. $J_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$, and $J_\nu(z)$ is an entire function of ν for any fixed z. If ν is an integer then $J_\nu(z)$ is also defined at 0 and is an entire function of z. The following subroutines are available for computing $J_\nu(z)$.

CALL CBSSLJ(z,ν,w)

The arguments z and ν are complex numbers and w is a complex variable. It is assumed that $|\arg(z)| < \pi$. When CBSSLJ is called, w is assigned the value $J_\nu(z)$.

Precision. CBSSLJ is accurate to within $4 \cdot 10^{-13}$ for real $0 < z \leq 35$ and $0 \leq \nu \leq 1$.

Note. CBSSLJ employs the subroutine CGAMMA.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Technical Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

CALL BSSLJ (z,n,w)

The argument z is a complex number, n is an integer, and w is a complex variable. When BSSLJ is called, w is assigned the value $J_n(z)$.

Precision. BSSLJ is accurate to within $5 \cdot 10^{-14}$ for real $0 < z \leq 35$ and $n = 0,1$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Technical Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

CALL BESJ(x,α,n,W,k)

The arguments x and α are nonnegative real numbers, n is a positive integer, and W is an array of dimension n or larger. When BESJ is called $J_{\alpha+i-1}(x)$ is computed and stored in W(i) for i = 1,..., n.

The argument k is an integer variable that is set by the routine. If all $J_{\alpha+i-1}(x)$ are successfully computed then k is set to 0. Otherwise, k is assigned one of the following values:

$k = -1$	The argument x is negative.
$k = -2$	The argument α is negative.
$k = -3$	The requirement $n \geqslant 1$ is violated.
$k > 0$	The last k components of W have been set to 0 because of underflow.

Precision. For $0 < x \leqslant 35$ and $0 \leqslant \alpha \leqslant 1$, BESJ is accurate to within $8 \cdot 10^{-13}$.

Programming. BESJ calls the subroutines ASJY and JAIRY, and the functions GAMLN, SPMPAR, and I1MACH. The subroutines were written by Donald E. Amos, Sharon L. Daniel, and M. Katherine Weston (Sandia Laboratories).

References

(1) Amos, D. E., Daniel, S. L., and Weston, M. K., ***CDC 6600 Subroutines for Bessel Functions $J_\nu(x)$, $x \geqslant 0$, $\nu \geqslant 0$ and Airy Functions $A_i(x)$, $A_i'(x)$, $-\infty < x < \infty$.*** Report SAND 75-0147, Sandia Laboratories, Albuquerque, New Mexico, 1975.

(2) ____________, "CDC 6600 Subroutines IBESS and JBESS for Bessel Functions $I_\nu(x)$ and $J_\nu(x)$, $x \geqslant 0$, $\nu \geqslant 0$," ***ACM Trans. Math Software 3*** (1977), pp. 76–92.

BESSEL FUNCTION $Y_\nu(z)$

If ν is any complex number not an integer, then $Y_\nu(z)$ can be defined by

$$Y_\nu(z) = \frac{J_\nu(z)\cos\nu\pi - J_{-\nu}(z)}{\sin\nu\pi}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. For any integer n we can also define $Y_n(z) = \lim_{\nu \to n} Y_\nu(z)$. Then for any complex ν, $Y_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$. Also, $Y_\nu(z)$ is an entire function of ν for any fixed z. The following subroutine is available for computing $Y_\nu(z)$ when ν is an integer.

<u>CALL BSSLY(z,n,w)</u>

The argument z is a complex number, n is an integer, and w is a complex variable. It is assumed that $|\arg(z)| < \pi$. When BSSLY is called, w is assigned the value $Y_n(z)$.

Precision. If $.005 \leqslant x \leqslant .785$ then $Y_0(x)$ and $Y_1(x)$ are accurate to within 3 units of the 14th significant digit. Otherwise, if $x > .785$ then $Y_0(x)$ and $Y_1(x)$ are accurate to within $4 \cdot 10^{-14}$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Technical Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

MODIFIED BESSEL FUNCTION $I_\nu(z)$

If ν is complex then $I_\nu(z)$ is defined by

$$I_\nu(z) = \sum_{k=0}^{\infty} \frac{(z/2)^{\nu+2k}}{k!\,\Gamma(\nu+k+1)}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. $I_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$, and $I_\nu(z)$ is an entire function of ν for any fixed z. If ν is an integer then $I_\nu(z)$ is also defined at 0 and is an entire function of z. The following subroutines are available for computing $I_\nu(z)$.

<u>CALL BSSLI(MO,z,n,w)</u>

MO is an integer, z a complex number, n an integer, and w a complex variable. If MO $\neq 0$ then it is assumed that $|\arg(z)| < \pi$. When BSSLI is called, w is assigned the value $I_n(z)$ if MO = 0 and the value $e^{-z} I_n(z)$ if MO $\neq 0$.

Precision. BSSLI is accurate to within 5 units of the 13th significant digit for real $0 < z \leqslant 35$ and n = 0, 1, ..., 40.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Technical Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

<u>CALL BESI(x,α,MO,n,W,k)</u>

MO may be 1 or 2. The arguments x and α are nonnegative real numbers, n is a positive integer, and W is an array of dimension n or larger. When BESI is called, if MO = 1 then $I_{\alpha+i-1}(x)$ is computed and stored in W(i) for i = 1, ..., n. Otherwise, if MO = 2 then $e^{-x} I_{\alpha+i-1}(x)$ is computed and stored in W(i).

The argument k is an integer variable that is set by the routine. If all $I_{\alpha+i-1}(x)$ or $e^{-x} I_{\alpha+i-1}(x)$ are successfully computed then k is set to 0. Otherwise, k is assigned one of the following values:

$k = -1$	The argument x is negative.
$k = -2$	The argument α is negative.
$k = -3$	The requirement $n \geqslant 1$ is violated.
$k = -4$	MO is not 1 or 2.
$k = -5$	The argument x is too large for MO = 1.
$k > 0$	The last k components of W have been set to 0 because of underflow.

Precision. For $0 < x \leqslant 35$ and $0 \leqslant \alpha \leqslant 1$ (or $0 < x \leqslant 35$ and $\alpha = 1,2,\ldots,40$) $I_\alpha(x)$ is accurate to within 2 units of the 12th significant digit.

Programming. BESI calls the subroutine ASIK and the functions GAMLN, SPMPAR, and I1MACH. BESI and ASIK were written by D. E. Amos and S. L. Daniel (Sandia Laboratories).

References

(1) Amos, D. E. and Daniel, S. L., ***A CDC 6600 Subroutine for Bessel Functions $I_\nu(x)$, $\nu \geqslant 0$, $x \geqslant 0$.*** Report SAND 75-0152, Sandia Laboratories, Albuquerque, New Mexico, 1975.

(2) Amos, D. E., Daniel, S. L., and Weston, M. K., "CDC 6600 Subroutines IBESS and JBESS for Bessel Functions $I_\nu(x)$ and $J_\nu(x)$, $x \geqslant 0$, $\nu \geqslant 0$," ***ACM Trans. Math Software 3*** (1977), pp. 76–92.

MODIFIED BESSEL FUNCTION $K_\nu(z)$

If ν is any complex number not an integer, then $K_\nu(z)$ can be defined by

$$K_\nu(z) = \frac{\pi}{2} \frac{I_{-\nu}(z) - I_\nu(z)}{\sin \nu\pi}$$

for any $z \neq 0$ in the complex plane cut along the negative real axis. For any integer n we can also define $K_n(z) = \lim_{\nu \to n} K_\nu(z)$. Then for any complex ν, $K_\nu(z)$ is analytic in the region $|\arg(z)| < \pi$. Also, $K_\nu(z)$ is an entire function of ν for any fixed z. The following subroutines are available for computing $K_\nu(z)$.

<u>CALL CBSSLK(z,r,w)</u>

The argument z is a complex number, r is a real number, and w is a complex variable. It is assumed that $|\arg(z)| < \pi$. When CBSSLK is called, w is assigned the value $K_r(z)$.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Approximations of Functions by Sets of Poles***, Technical Report TR-2564, Naval Weapons Laboratory, Dahlgren, Virginia, 1971.

<u>CALL BSSLK(MO,z,n,w)</u>

MO is an integer, z a complex number, n an integer, and w a complex variable. It is assumed that $|\arg(z)| < \pi$. When BSSLK is called, w is assigned the value $K_n(z)$ if MO = 0 and the value $e^z K_n(z)$ if MO $\neq$ 0.

Precision. BSSLK is accurate to within 6 units of the 14th significant digit for real z and n = 0, 1.

Programmer. A. V. Hershey

Reference. Hershey, A. V., ***Computation of Special Functions***, Technical Report TR-3788, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.

COMPLETE COMPLEX ELLIPTIC INTEGRALS OF THE FIRST AND SECOND KINDS

If k is complex then the complete elliptic integrals of the first and second kinds can be defined by

$$K(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 t)^{-1/2}\, dt$$

$$E(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 t)^{1/2}\, dt$$

for $|\arg(1 - k^2)| < \pi$. K(k) and E(k) can be extended to $-\pi \leq \arg(1 - k^2) < \pi$. For $|k| < 1$

$$(1)\qquad \begin{aligned} K(k) &= \frac{\pi}{2} \sum_{n\geq 0} c_n k^{2n} \\ E(k) &= -\frac{\pi}{2} \sum_{n\geq 0} c_n \frac{k^{2n}}{2n-1} \end{aligned}$$

where $c_n = \left[\dfrac{(2n)!}{4^n (n!)^2}\right]^2$. Also, if $\ell^2 = 1 - k^2$ where $|\ell| \leq 1$ and $-\pi \leq \arg(\ell^2) < \pi$, then

$$(2)\qquad \begin{aligned} K(k) &= \frac{1}{\pi} K(\ell) \ln \frac{16}{\ell^2} - \sum_{n\geq 1} c_n \sum_{m=1}^{n} \frac{\ell^{2n}}{m(2m-1)} \\ E(k) &= \frac{1}{\pi} [K(\ell) - E(\ell)] \ln \frac{16}{\ell^2} - \sum_{n\geq 1} c_n \frac{2n}{2n-1} \sum_{m=1}^{n} \frac{\ell^{2n}}{m(2m-1)} + \sum_{n\geq 0} c_n \frac{\ell^{2n}}{(2n-1)^2} . \end{aligned}$$

The function CK is available for computing K(k), and the subroutine CKE for computing K(k) and E(k).

<u>CK(k,ℓ)</u>

CK(k,ℓ) = K(k) for any complex k and ℓ where $k^2 + \ell^2 = 1$ and $\ell \neq 0$. CK is a complex valued function which must be declared in the calling program to be of type COMPLEX.

Error Return. CK(k,ℓ) = 0 if $\ell = 0$ or $k^2 + \ell^2 \neq 1$.

Remarks

(1) CK(k,ℓ) may underflow, yielding the value 0, when $|k|$ is sufficiently large.

(2) CK and the subroutine CKE employ the same algorithm for K(k).

Precision. If k is real and $|k| < 1$ then the relative error of CK is less than 10^{-13}. Also, if k is purely imaginary then the relative error is less than 10^{-13}. K(k) is real-valued for only these values of k. Otherwise, let $\epsilon_k = 10^{-12}$ if $|k| < 0.8$, $\epsilon_k = 2 \cdot 10^{-13}$ if $0.8 \leqslant |k| < 2$, and $\epsilon_k = 10^{-13}$ if $|k| \geqslant 2$. Then the relative errors of the real and imaginary parts of CK are less than ϵ_k except when underflow occurs, $|k| < 1$ and $|\arg(\pm k)| < 10^{-287}$, or $|k| < 10^{15}$ and $|\pi/2 - \arg(\pm k)| < 10^{-280}$. In the latter two cases the relative error of the real part of CK is less than ϵ_k, but all relative accuracy for the imaginary part may be lost.

Programming. CK calls the subroutine KL and functions ALNREL, CFLECT, KM, and SPMPAR. CK, KL, and KM were written by Andrew H. van Tuyl (Naval Surface Weapons Center) and modified by A. H. Morris.

CALL CKE(k,ℓ,K,E,IERR)

The arguments k and ℓ are complex numbers where $k^2 + \ell^2 = 1$ and $\ell \neq 0$, K and E are complex variables, and IERR an integer variable. When CKE is called, if no errors are detected then IERR is set to 0, K is assigned the value K(k), and E is assigned the value E(k).

Error Return. IERR = 1 if $\ell = 0$ and IERR = 2 if $k^2 + \ell^2 \neq 1$. In these cases, K and E are not defined.

Algorithm. For k = 0 or $-\pi/2 < \arg(k) \leqslant \pi/2$, formulae (2) are used if $|\ell| \leqslant .55$, (1) are used if $|\ell| > .55$ and $|k| \leqslant .55$, and approximations of the form

$$\text{(3)} \qquad \begin{aligned} K(k) &= \ell \sum_{n=1}^{N} \frac{a_n}{\ell^2 + b_n^2 k^2} \\ E(k) &= \ell \sum_{n=1}^{N} a_n \left[1 + \frac{b_n k}{\ell} \tan^{-1} \frac{b_n k}{\ell}\right] \end{aligned}$$

are used if $|\ell| > .55$ and $.55 \leqslant |k| \leqslant 1$. (3) are obtained from integral representations for K(k) and E(k) by numerical quadrature. If $|\ell| > .55$, $|k| > 1$, and $|k| \leqslant |\ell|$ then

$$\text{(4)} \qquad \begin{aligned} K(k) &= \ell_1 \; K(k_1) \qquad & k_1 &= \pm ik/\ell \\ E(k) &= E(k_1)/\ell_1 \qquad & \ell_1 &= 1/\ell \end{aligned}$$

are applied where the sign in k_1 is selected so that $-\pi/2 < \arg(k_1) \leqslant \pi/2$. Otherwise, if $|\ell| > .55$, $|k| > 1$, and $|k| > |\ell|$ let $k_1 = 1/k$ and $\ell_1 = \pm i\ell/k$ where the sign is selected so that $-\pi/2 < \arg(\ell_1) \leqslant \pi/2$. Then

$$
(5) \qquad
\begin{aligned}
K(k) &= k_1 \, [K(k_1) + i\, s\, K(\ell_1)] \\
E(k) &= \frac{1}{k_1} \, [E(k_1) - \ell_1^2 \, K(k_1) - i\, s\, (E(\ell_1) - k_1^2 \, K(\ell_1))]
\end{aligned}
$$

are applied where $s = 1$ if $\mathrm{Im}(k) \geqslant 0$ and $s = -1$ if $\mathrm{Im}(k) < 0$. If $\arg(k) > \pi/2$ or $\arg(k) \leqslant -\pi/2$, then $K(k) = K(-k)$ and $E(k) = E(-k)$ are applied.

Precision. If k is real and $|k| < 1$, or k is purely imaginary, then the relative error of E is less than 10^{-13}. E(k) is real-valued for only these values of k. Otherwise, let $\epsilon_k = 10^{-12}$ if $|k| < 2$ and $\epsilon_k = 10^{-13}$ if $|k| \geqslant 2$. Then the relative errors of the real and imaginary parts of E are less than ϵ_k except when underflow occurs, $|k| < 1$ and $|\arg(\pm k)| < 10^{-280}$, or $|k| < 10^{15}$ and $|\pi/2 - \arg(\pm k)| < 10^{-280}$. In the latter two cases the relative error of the real part of E is less than ϵ_k, but all relative accuracy for the imaginary part may be lost.

Programming. CKE calls the subroutines EKL and EKM, and the functions ALNREL, ATN, CFLECT, and SPMPAR. CKE was written by Andrew H. van Tuyl (Naval Surface Weapons Center).

REAL ELLIPTIC INTEGRALS OF THE FIRST AND SECOND KINDS

If $0 \leqslant \phi \leqslant \pi/2$, then the elliptic integrals of the first and second kinds are defined by

$$F(\phi,k) = \int_0^\phi (1 - k^2 \sin^2 t)^{-1/2}\, dt$$

$$E(\phi,k) = \int_0^\phi (1 - k^2 \sin^2 t)^{1/2}\, dt$$

for any real k where $k^2 \leqslant 1$ and $|k \sin \phi| \neq 1$. If $\phi = \pi/2$ then the integrals are said to be ***complete***. Otherwise, if $\phi \neq \pi/2$ then the integrals are said to be ***incomplete***. The following subroutine is available for computing $F(\phi,k)$ and $E(\phi,k)$.

CALL ELLPI(ϕ,ψ,k,ℓ,F,E,IERR)

The arguments ϕ, ψ, k, ℓ are real numbers which satisfy $\phi \geqslant 0$, $\psi \geqslant 0$, $\phi + \psi = \pi/2$, and $k^2 + \ell^2 = 1$. Also, if $\psi = 0$ then it is further assumed that $\ell \neq 0$. F, E, and IERR are variables. When ELLPI is called, if no input errors are detected then IERR is assigned the value 0, F is assigned the value $F(\phi,k)$, and E is assigned the value $E(\phi,k)$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1	$\phi < 0$ or $\psi < 0$
IERR = 2	$\lvert k\rvert > 1$ or $\lvert \ell \rvert > 1$
IERR = 3	$\psi = 0$ and $\ell = 0$

Precision. ELLPI maintains accuracy to within 5 units of the 14th significant digit.

Programming. ELLPI calls the functions ALNREL and CPABS. ELLPI was written by Allen V. Hershey and modified by A. H. Morris.

Reference. DiDonato, A. R. and Hershey, A. V., "New Formulas for Computing Incomplete Elliptic Integrals of the First and Second Kind," ***JACM 6*** (October 1959), pp. 515-526.

REAL ELLIPTIC INTEGRALS OF THE THIRD KIND

For any $0 \leqslant \phi \leqslant \pi/2$ the elliptic integral $\Pi(\phi,n,k)$ is defined by

$$\Pi(\phi,n,k) = \int_0^\phi (1-n \sin^2\theta)^{-1} (1-k^2 \sin^2\theta)^{-1/2} \, d\theta$$

where n is any real number such that $1-n \sin^2\phi \neq 0$, and k any real number such that $k^2 \leqslant 1$ and $1-k^2 \sin^2\phi \neq 0$. Alternatively, for any $r \neq 0$ we may consider

$$R_J(a,b,c,r) = \frac{3}{2} \int_0^\infty (t+r)^{-1} \, [(t+a)(t+b)(t+c)]^{-1/2} \, dt$$

where a,b,c are nonnegative and at most one of them is 0. If $a \leqslant b \leqslant c$ and $a < c$ then

$$R_J(a,b,c,r) = \frac{3\,c^{-3/2}}{n \sin^3\phi} \, [\Pi(\phi,n,k) - F(\phi,k)]$$

where $F(\phi,k)$ is the elliptic integral of the first kind and $\cos^2\phi = a/c$, $k^2 = (c-b)/(c-a)$, $n = (c-r)/(c-a)$. If $\phi = \pi/2$ then the elliptic integral $\Pi(\phi,n,k)$ is said to be ***complete***. Otherwise, if $\phi < \pi/2$ then the integral is said to be ***incomplete***. The subroutine EPI is available for computing $\Pi(\phi,n,k)$, and the function RJ is available for computing $R_J(a,b,c,r)$.

CALL EPI($\phi,\pi/2-\phi,k^2,1-k^2,n,1-n$,p,IERR)

The arguments ϕ,n,k^2 are real numbers which satisfy $0 \leqslant \phi \leqslant \pi/2$, $|n| \leqslant 1$, and $k^2 \leqslant 1$. Also, if $\phi = \pi/2$ then it is further assumed that $n \neq 1$ and $k^2 \neq 1$. IERR and p are variables. When EPI is called, if no input errors are detected then IERR is assigned the value 0 and p is assigned the value $\Pi(\phi,n,k)$.

Remark. If $\phi < .79$ then the argument $\pi/2 - \phi$ is not employed in the calculation of $\Pi(\phi,n,k)$. Otherwise, if $\phi \geqslant .79$ then ϕ is not used.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1	Either ϕ or the argument $\pi/2 - \phi$ is negative.
IERR = 2	$\lvert n\rvert > 1$.
IERR = 3	Either $k^2 < 0$ or $k^2 > 1$.
IERR = 4	ϕ and n are too close to $\pi/2$ and 1, or ϕ and k^2 are too close to $\pi/2$ and 1.

The arguments $1-k^2$ and $1-n$ are not checked. If these arguments are improperly set then IERR = 4 may occur.

Precision. EPI is accurate to within 5 units of the 14th significant digit.

Programming. EPI employs the functions RF, RC, and RJ. These functions were written by B. C. Carlson and Elaine M. Notis (Iowa State University). EPI was written by A. H. Morris. The function SPMPAR is also used.

Reference. Carlson, B. C. and Notis, E. M., "Algorithm 577, Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software*** *7* (1981), pp. 398-403.

RJ(a,b,c,r,τ,IERR)

The arguments a,b,c are nonnegative and at most one of them is 0. It is also assumed that $r > 0$. Then RJ(a,b,c,r,τ,IERR) = R_J (a,b,c,r).

The argument τ is used for setting the desired precision. The relative error due to truncation of the Taylor series expansion used in RJ is less than $3\tau^6/(1-\tau)^{3/2}$. Thus, decreasing τ by a factor of 10 will yield six more digits of accuracy. For example:

τ	Relative Truncation error less than
3.E–3	3.E–15
1.E–2	4.E–12
3.E–2	3.E–9
1.E–1	4.E–6

IERR is an integer variable that is set by the routine. If R_J(a,b,c,r) is successfully computed then IERR is assigned the value 0. Otherwise, IERR = 1 when one of the following situations occurs:

(1) At least one of the arguments a,b,c is negative.

(2) At least one of the values a + b, a + c, b + c, r is too small. On the CDC 6000–7000 series computers this normally occurs when one of these values is less than .25E-97.

(3) At least one of the arguments a,b,c,r is too large. On the CDC 6000–7000 series computers this occurs when one of the arguments is greater than .4E + 107.

If IERR = 1 then the value of RJ is meaningless.

Programming. RJ calls the function RC. These functions were written by B. C. Carlson and Elaine M. Notis (Iowa State University). The function SPMPAR is also used.

References

(1) Carlson, B. C., "Computing Elliptic Integrals by Duplication," ***Numerische Mathematik 33*** (1979), pp. 1–16.

(2) ____________ and Notis, E. M., "Algorithm 577, Algorithms for Incomplete Elliptic Integrals," ***ACM Trans. Math Software 7*** (1981), pp. 398–403.

JACOBIAN ELLIPTIC FUNCTIONS

For $|k| \leqslant 1$ consider the upper limit ϕ of the integral

$$u = \int_0^{\phi} \frac{dt}{\sqrt{1 - k^2 \sin^2 t}}$$

as a function of u. Then for any real u the elliptic functions sn(u,k), cn(u,k), and dn(u,k) may be defined by

$$\begin{aligned} \mathrm{sn}(u,k) &= \sin\phi \\ \mathrm{cn}(u,k) &= \cos\phi \\ \mathrm{dn}(u,k) &= \sqrt{1 - k^2 \sin^2 \phi} = \frac{d\phi}{du}. \end{aligned}$$

For $|k| = 1$
$$\begin{aligned} \mathrm{sn}(u,1) &= \tanh u \\ \mathrm{cn}(u,1) &= 1/\cosh u \\ \mathrm{dn}(u,1) &= \mathrm{cn}(u,1). \end{aligned}$$

Otherwise, for $|k| < 1$ we have

$$\begin{aligned} \mathrm{sn}(-u,k) &= -\mathrm{sn}(u,k) \\ \mathrm{cn}(-u,k) &= \mathrm{cn}(u,k) \\ \mathrm{dn}(-u,k) &= \mathrm{dn}(u,k) \end{aligned}$$

$$\begin{aligned} \mathrm{sn}(u + 2K,k) &= -\mathrm{sn}(u,k) \\ \mathrm{cn}(u + 2K,k) &= -\mathrm{cn}(u,k) \\ \mathrm{dn}(u + 2K,k) &= \mathrm{dn}(u,k) \end{aligned}$$

where K = K(k) is the complete elliptic integral of the first kind. The following subroutine is available for computing sn(u,k), cn(u,k), and dn(u,k).

CALL ELLPF (u,k,ℓ,S,C,D,IERR)

It is assumed that k and ℓ are real numbers where $k^2 + \ell^2 = 1$. S, C, and D are variables. When ELLPF is called, S, C, and D are assigned the values S = sn(u,k), C = cn(u,k), and D = dn(u,k).

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0	The elliptic functions were computed.
IERR = 1	(Input error) $k^2 + \ell^2 \neq 1$
IERR = 2	u is too large for k.

When IERR $\geqslant$ 1, no computation is performed.

Precision. For $|k| < .99995$ the relative errors of sn(u,k) and dn(u,k) are less than 10^{-12} when $0 < u \leqslant K$, and the relative error of cn(u,k) is less than 10^{-12} when $0 \leqslant u \leqslant .97K$.

Algorithm. For $0 \leqslant u \leqslant K/2$ (when $\ell \neq 0$), the Maclaurin expansion

$$\mathrm{sn}(u,k) = u - \frac{(1+k^2)u^3}{3!} + \frac{(1+14k^2+k^4)u^5}{5!} - \cdots$$

is employed when $u \leqslant .01$. Otherwise, if $u > .01$ let $K' = K(\ell)$, $q = \exp(-\pi K'/K)$, and $r = \exp(-\pi K/K')$. Then

$$\mathrm{sn}(u,k) = \frac{2\pi}{kK} \sum_{n \geqslant 0} \frac{q^{n+\frac{1}{2}}}{1-q^{2n+1}} \sin \frac{(2n+1)\pi u}{2K}$$

is used when $k \leqslant \ell$ and

$$\mathrm{sn}(u,k) = \frac{\pi}{2kK'} \left[\tanh \frac{\pi u}{2K'} + 4 \sum_{n \geqslant 1} \frac{(-1)^n r^{2n}}{1+r^{2n}} \sinh \frac{2\pi u}{K'} \right]$$

is used when $k > \ell$. The functions cn(u,k) and dn(u,k) are obtained from

$$\mathrm{sn}(u,k)^2 + \mathrm{cn}(u,k)^2 = 1$$
$$\mathrm{dn}(u,k)^2 + k^2\,\mathrm{sn}(u,k)^2 = 1.$$

For $K/2 < u \leqslant K$ the identities

$$\mathrm{sn}(u,k) = \mathrm{cn}(v,k)/\mathrm{dn}(v,k)$$
$$\mathrm{cn}(u,k) = |\ell|\,\mathrm{sn}(v,k)/\mathrm{dn}(v,k)$$
$$\mathrm{dn}(u,k) = |\ell|/\mathrm{dn}(v,k)$$

are applied. Here $v = K - u$.

Programming. ELLPF employs the subroutines SCD, SCDF, SCDJ, SCDM, ELLPI, SNHCSH and functions ALNREL, CPABS, SPMPAR, I1MACH. ELLPF was written by Andrew H. van Tuyl and modified by A. H. Morris.

WEIERSTRASS ELLIPTIC FUNCTION FOR THE EQUIANHARMONIC AND LEMNISCATIC CASES

Let w and w′ be complex numbers where $\text{Im}(w'/w) > 0$, and let $w_{mn} = 2mw + 2nw'$ for all integers m,n. Then for any complex z, the Weierstrass elliptic function $\wp(z;w,w')$ can be defined by

$$\wp(z;w,w') = \frac{1}{z^2} + \Sigma' \left[\frac{1}{(z - w_{mn})^2} - \frac{1}{w_{mn}^2}\right]$$

where Σ' denotes the sum for all m,n = 0, ±1, ±2,... except m = n = 0. If $w = re^{i\phi}$ and $w' = r'e^{i\phi'}$ where $\phi' = \phi + \theta$ for $0 \leqslant \theta < 2\pi$, then the restriction $\text{Im}(w'/w) > 0$ is equivalent to assuming that $0 < \theta < \pi$. $\wp(z;w,w')$ is analytic everywhere except at the points w_{mn}, which are poles, and

$$\wp(z + 2w;w,w') = \wp(z;w,w')$$
$$\wp(z + 2w';w,w') = \wp(z;w,w')$$

for all z. The relations

$$\wp(-z;w,w') = \wp(z;w,w')$$
$$\wp(\lambda z;\lambda w,\lambda w') = \lambda^{-2}\wp(z;w,w') \qquad \lambda \neq 0$$

also hold. A somewhat surprising fact is that only the values $g_2 = 60\ \Sigma'\ w_{mn}^{-4}$ and $g_3 = 140\ \Sigma'\ w_{mn}^{-6}$ are needed for computing $\wp(z;w,w')$ at a point z. Hence, $\wp(z;w,w')$ is frequently denoted by $\wp(z;g_2,g_3)$. For $\lambda \neq 0$

$$g_2(\lambda w,\lambda w') = \lambda^{-4}g_2(w,w')$$
$$g_3(\lambda w,\lambda w') = \lambda^{-6}g_3(w,w')$$

also hold. We now consider the following cases:

(1) Equianharmonic ($g_2 = 0$ and g_3 is a positive real number)

(2) Lemniscatic (g_2 is a positive real number and $g_3 = 0$)

(1) occurs when $2w = 1/2 - \frac{\sqrt{3}}{2}i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2}i$, and (2) occurs when 2w = 1 and 2w′ = i. The following subroutines are available for computing $\wp(z;w,w')$ and its deviative $\wp'(z;w,w')$ for these two choices of (w,w′).

<u>CALL PEQ(z,e,IERR)</u>

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = \frac{1}{2} - \frac{\sqrt{3}}{2}i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2}i$. When PEQ is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\wp(z;w,w')$.

Error Return. If $z = w_{mn}$ for some m,n then IERR is assigned the value 1 and e = 0.

Precision. If $|\wp(z;w,w')| \leqslant 1$ then the absolute error is less than $7 \cdot 10^{-13}$. Otherwise, the relative error is less than $7 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References

(1) Eckhardt, Ulrich, "Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software 4*** (1980), pp. 112–120.

(2) ____________, "A Rational Approximation to Weierstrass' $\mathscr{P}$-Function," ***Mathematics of Computation 30*** (1976), pp. 818–826.

CALL PEQ1(z,e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = \frac{1}{2} - \frac{\sqrt{3}}{2} i$ and $2w' = \frac{1}{2} + \frac{\sqrt{3}}{2} i$. When PEQ1 is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathscr{P}'(z;w,w')$.

Error Return. If $z = w_{mn}$ for some m,n then IERR is assigned the value 1 and e = 0.

Precision. If $|\mathscr{P}'(z;w,w')| \leqslant 1$ then the absolute error is less than $7 \cdot 10^{-13}$. Otherwise, the relative error is less than $7 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References

(1) Eckhardt, Ulrich, "Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software 4*** (1980), pp. 112–120.

(2) ____________, "A Rational Approximation to Weierstrass' $\mathscr{P}$-Function," ***Mathematics of Computation 30*** (1976), pp. 818–826.

CALL PLEM(z,e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are 2w = 1 and 2w′ = i. When PLEM is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\mathscr{P}(z;w,w')$.

Error Return. If $z = w_{mn}$ for some m,n then IERR is assigned the value 1 and e = 0.

Precision. If $|\mathscr{P}(z;w,w')| \leqslant 1$ then the absolute error is less than $6 \cdot 10^{-13}$. Otherwise, the relative error is less than $6 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References

(1) Eckhardt, Ulrich, "Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software** 4* (1980), pp. 112–120.

(2) ____________, "A Rational Approximation to Weierstrass' $\wp$-Function. II: The Lemniscatic Case," ***Computing (Arch. Elektron. Rechnen)** 18* (1977), pp. 341–349.

CALL PLEM1(z,e,IERR)

The argument z is a complex number, e is a complex variable, and IERR is an integer variable. It is assumed that the periods are $2w = 1$ and $2w' = i$. When PLEM1 is called, if z is not a pole then IERR is assigned the value 0 and e is assigned the value $\wp'(z; w, w')$.

Error Return. If $z = w_{mn}$ for some m,n then IERR is assigned the value 1 and e = 0.

Precision. If $|\wp'(z; w, w')| \leqslant 1$ then the absolute error is less than $6 \cdot 10^{-13}$. Otherwise, the relative error is less than $6 \cdot 10^{-13}$.

Programming. Written by Ulrich Eckhardt (University of Hamburg, West Germany). Modified by A. H. Morris.

References

(1) Eckhardt, Ulrich, "Algorithm 549, Weierstrass' Elliptic Functions," ***ACM Trans. Math Software** 4* (1980), pp. 112–120.

(2) ____________, "A Rational Approximation to Weierstrass' $\wp$-Function. II: The Lemniscatic Case," ***Computing (Arch. Elektron. Rechnen)** 18* (1977), pp. 341–349.

INTEGRAL OF THE BIVARIATE DENSITY FUNCTION OVER ARBITRARY POLYGONS AND SEMI-INFINITE ANGULAR REGIONS

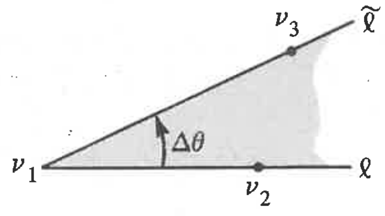
Given a sequence of points $\nu_i = (x_i, y_i)$ $(i = 1, \dots, n + 1)$ where $n \geqslant 3$ and $\nu_{n+1} = \nu_1$. Let τ denote the polygon whose boundary $\partial\tau$ is a polygonal line which begins at point ν_1, traverses the points ν_i in the order that they are indexed, and is the straight line segment connecting ν_i to ν_{i+1} for each $i = 1, \dots, n$ where $\nu_i \neq \nu_{i+1}$. Then the subroutine VALR2 is available for computing the integral

$$P(\tau) = \frac{1}{2\pi} \iint_{\tau} \exp\left[-(x^2 + y^2)/2\right] dx\, dy$$

and the associated function $A(\tau) = \iint_{\tau} dx\, dy$. If the boundary $\partial\tau$ is a simple positively (negatively) oriented closed curve, then $P(\tau)$ and $A(\tau)$ are positive (negative) and $|A(\tau)|$ = the area of τ. However, $\partial\tau$ need not be simple. It may be self-intersecting or have overlapping line segments. If $\Delta\theta_i$ is the angle between the vectors $\nu_i - \nu_{i-1}$ and $\nu_{i+1} - \nu_i$ (where $\nu_0 = \nu_n$), then it may occur that $\Delta\theta_i = \pi$ for some i, in which case a portion of the polygon may be degenerate. In general, $-\pi < \Delta\theta_i \leqslant \pi$ for each i where the sign of the angle is positive (negative) if the angle is measured in a counterclockwise (clockwise) direction from $\nu_i - \nu_{i-1}$ to $\nu_{i+1} - \nu_i$. VALR2 also computes the value $k(\tau) = \frac{1}{2\pi} \sum_{i=1}^{n} \Delta\theta_i$, which is an integer. If the boundary is a simple closed curve, then $k(\tau)$ is the winding number of the curve around any interior point of the polygon τ.

Alternatively, assume that we are given three points $\nu_i = (x_i, y_i)$ $(i = 1,2,3)$ and let $\Delta\theta$ denote the angle between the vectors $\nu_2 - \nu_1$ and $\nu_3 - \nu_1$. In this case, assume that the angle $\Delta\theta$ is measured in a counterclockwise direction from $\nu_2 - \nu_1$ to $\nu_3 - \nu_1$, so that $0 \leqslant \Delta\theta < 2\pi$. Let ℓ denote the straight line beginning at point ν_1 and passing through point ν_2, and let $\tilde{\ell}$ denote the straight line beginning at ν_1 and passing through ν_3. Then the subroutine VALR2 is also available for computing $P(\tau)$ when τ is the semi-infinite angular region bounded by ℓ and $\tilde{\ell}$, and having the angle $\Delta\theta$. $0 \leqslant P(\tau) \leqslant 1$ for any angular region τ, and $P(\tau) \to 1$ when $\Delta\theta \to 2\pi$.

<ins>Angular region τ</ins>



<ins>CALL VALR2(X,Y,n,P,IOP,A,IND,k)</ins>

The argument n is either 1 or the number of points involved in defining a polygon. If n = 1 then it is assumed that τ is a semi-infinite angular region defined by the points

$\nu_i = (x_i, y_i)$ $(i = 1,2,3)$, and that X and Y are arrays containing x_1, x_2, x_3 and y_1, y_2, y_3. Otherwise, if $n \neq 1$ then it is assumed that τ is a polygon defined by the points $\nu_i = (x_i, y_i)$ $(i = 1, \ldots, n + 1)$ where $n \geqslant 3$ and $\nu_{n+1} = \nu_1$. In this case, X and Y are arrays containing the abscissae $x_1, \ldots, x_{n+1}$ and ordinates $y_1, \ldots, y_{n+1}$. Since $\nu_{n+1} = \nu_1$, the values x_{n+1} and y_{n+1} need not be supplied by the user. The routine automatically stores x_1 and y_1 in X(n+1) and Y(n+1).

P, A, and k are variables. If $n = 1$ then P is assigned the value $P(\tau)$ for the angular region τ and A is assigned the value 0. In this case, k is not defined. Otherwise, if $n \geqslant 3$ then P is assigned the value $P(\tau)$, A is assigned the value $A(\tau)$, and k is assigned the value $k(\tau)$ for the polygon τ.

IOP is an input argument which specifies the (relative) precision to which $P(\tau)$ is to be computed. IOP is set to 1, 2, or 3 for 3, 6, or 9 decimal digit accuracy.

IND is a variable that reports the status of the results. The routine assigns IND one of the following values:

IND = 0	The desired values were obtained.
IND = 1	(Input error) Point ν_1 is either equal to ν_2 or ν_3, or is too close to ν_2 or ν_3 to compute $P(\tau)$ for the angular region τ.
IND = 2	The desired values were obtained. If $n = 1$ then $\lvert \Delta\theta - \pi \rvert < \epsilon$ where ϵ is a tolerance whose value depends on the value for IOP. Otherwise, if $n \geqslant 3$ then $\lvert \Delta\theta_i \rvert > \pi - \epsilon$ for some i.
IND = 3	(Input error) Either $n < 1$ or $n = 2$.

When IND = 1 occurs, P is assigned the value 5.

Remarks: VALR2 can be used for computing the integral of the general bivariate density function over an arbitrary polygon or semi-infinite angular region $\hat{\tau}$. Consider

$$\hat{P}(\hat{\tau}) = \frac{(1-\rho^2)^{-1/2}}{2\pi\sigma_\omega\sigma_z}\iint_{\hat{\tau}} \exp\left\{\frac{-1}{2(1-\rho^2)}\left[\left(\frac{\omega-\mu_\omega}{\sigma_\omega}\right)^2 - 2\rho\frac{(\omega-\mu_\omega)(z-\mu_z)}{\sigma_\omega\sigma_z} + \left(\frac{z-\mu_z}{\sigma_z}\right)^2\right]\right\} d\omega\, dz$$

where (μ_ω, μ_z) is the mean, σ_ω and σ_z are the (nonzero) variances, and ρ is the correlation coefficient satisfying $\lvert\rho\rvert < 1$. Consider also the transformation

$$x = (1-\rho^2)^{-1/2}\left[\frac{\omega-\mu_\omega}{\sigma_\omega} - \rho\frac{z-\mu_z}{\sigma_z}\right]$$

$$y = \frac{z-\mu_z}{\sigma_z}.$$

Since this transformation maps straight lines into straight lines, $\hat{\tau}$ is mapped onto a polygon or angular region τ and we obtain $\hat{P}(\hat{\tau}) = P(\tau)$. Moreover, if $\hat{\tau}$ is a polygon then $A(\hat{\tau}) = \iint_{\hat{\tau}} d\omega\, dz = \sigma_\omega\sigma_z\sqrt{1-\rho^2}\; A(\tau)$.

Programming. VALR2 employs the functions ERF, ERFC1, and SPMPAR. VALR2 was designed by Armido R. DiDonato and Richard K. Hageman, and modified by A. H. Morris.

Reference. DiDonato, A. R., and Hageman, R. K., ***Computation of the Integral of the Bivariate Normal Distribution over Arbitrary Polygons,*** Technical Report TR 80-166, Naval Surface Weapons Center, Dahlgren, Virginia, 1980.

CIRCULAR COVERAGE FUNCTION

The subroutine CIRCV is available for computing the circular coverage function P(R,d) and the generalized circular error function V(K,c). V is the integral of an uncorrelated elliptical Gaussian distribution with standard deviations σ_x and σ_y over a circle of radius $K\sigma_x$ centered at the mean of the distribution. If $\sigma_x \geqslant \sigma_y$ then

$$V(K,c) = \frac{1}{\pi c} \int_0^K \int_0^\pi \exp\left\{\frac{-r^2}{4c^2}\left[1 + c^2 + (1 - c^2)\cos\theta\right]\right\} r\, dr\, d\theta$$

where $c = \sigma_y/\sigma_x$. P is the integral of a circular Gaussian distribution with common standard deviation σ over a circle of radius $R\sigma$ whose center is offset a distance $d\sigma$ from the mean of the distribution.

$$P(R,d) = \frac{1}{2\pi} \int_0^R \int_0^{2\pi} \exp\left\{-\frac{1}{2}\left[(d + r\cos\theta)^2 + r^2\sin^2\theta\right]\right\} r\, dr\, d\theta$$

<u>CALL CIRCV(x,a,i,w,IERR)</u>

The argument i may be any integer. If i = 0 then the arguments x and a are assumed to have the values x = K and a = c where $K \geqslant 0$ and $0 \leqslant c \leqslant 1$. Otherwise, if $i \neq 0$ then x = R and a = d where $R \geqslant 0$ and $d \geqslant 0$.

IERR and w are variables. When CIRCV is called, if no input errors are detected then IERR is assigned the value 0. Also, w = V(K,c) if i = 0 and w = P(R,d) if $i \neq 0$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1	$x \geqslant 0$ is not satisfied.
IERR = 2	$0 \leqslant c \leqslant 1$ or $d \geqslant 0$ is not satisfied.

When either of these errors is detected, w is assigned the value −1.

Precision. CIRCV is accurate to within 10^{-6}.

Note. If $\sigma_x \leqslant \sigma_y$ then reverse the roles of x and y.

Programming. CIRCV calls these functions ERF0 and ERFC0. The routine is an adaptation by A. H. Morris of the BASIC program CIRCV given in reference (1).

References

(1) DiDonato, A. R., ***Five Statistical Programs in BASIC for Desktop Computers***, Technical Report NSWC TR 83-13, Naval Surface Weapons Center, Dahlgren, Virginia, 1982.

(2) ________________ and Jarnagin, M. P., ***A Method for Computing the Generalized Circular Error Function and the Circular Coverage Function***, NWL Report 1768, Naval Weapons Laboratory, Dahlgren, Virginia, 1962.

COPYING POLYNOMIALS

If $p(x) = \sum_{j=0}^{m-1} a_j x^j$ and the coefficients a_j are stored in an array A, then the following subroutines are available for copying the first n coefficients a_j into an array B.

<u>CALL PLCOPY(A,ka,m,B,kb,n)</u>
<u>CALL DPCOPY(A,ka,m,B,kb,n)</u>

A and B are arrays. PLCOPY is used if A and B are single precision real arrays, and DPCOPY is used if A and B are double precision arrays.

The arguments m, n, ka, kb are positive integers. The coefficients a_j are assumed to be stored in A where $A(1 + j*ka) = a_j$ for $j = 0, 1, \ldots, m - 1$. The routine stores the first n coefficients a_j in B where $B(1 + j*kb) = a_j$ for $j = 0, 1, \ldots, n - 1$.

Note. If $n > m$ then $B(1 + j*kb) = 0$ for $j \geqslant m$.

Programmer. A. H. Morris

ADDITION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x) + q(x) = \Sigma_j c_j x^j$.

CALL PADD(A,ka,ℓ,B,kb,m,C,kc,n)
CALL DPADD(A,ka,ℓ,B,kb,m,C,kc,n)

A, B, C are arrays. PADD is used if A, B, C are single precision real arrays and DPADD is used if A, B, C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$$A(1 + j*ka) = a_j \quad (j = 0,1,\dots,\ell - 1)$$
$$B(1 + j*kb) = b_j \quad (j = 0,1,\dots,m - 1).$$

The routine stores the first n coefficients c_j of $p(x) + q(x)$ in C where $C(1 + j*kc) = c_j$ for $j = 0, 1,\dots,n - 1$.

Remarks. The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris

SUBTRACTION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x) - q(x) = \Sigma_j c_j x^j$.

<u>CALL PSUBT(A,ka,ℓ,B,kb,m,C,kc,n)</u>
<u>CALL DPSUBT(A,ka,ℓ,B,kb,m,C,kc,n)</u>

A, B, C are arrays. PSUBT is used if A, B, C are single precision real arrays and DPSUBT is used if A, B, C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j*ka) = a_j \quad (j = 0, 1, \ldots, \ell - 1)$
$B(1 + j*kb) = b_j \quad (j = 0, 1, \ldots, m - 1)$

The routine stores the first n coefficients c_j of $p(x) - q(x)$ in C where $C(1 + j*kc) = c_j$ for $j = 0, 1, \ldots, n - 1$.

Remarks. The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris

MULTIPLICATION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ then the following subroutines are available for computing the first n coefficients of the polynomial $p(x)q(x) = \Sigma_j c_j x^j$.

CALL PMULT(A,ka,ℓ,B,kb,m,C,kc,n)
CALL DPMULT(A,ka,ℓ,B,kb,m,C,kc,n)

A, B, C are arrays. PMULT is used if A, B, C are single precision real arrays and DPMULT is used if A, B, C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j*ka) = a_j \quad (j = 0,1,\ldots,\ell - 1)$
$B(1 + j*kb) = b_j \quad (j = 0,1,\ldots,m - 1).$

The routine stores the first n coefficients c_j of $p(x)q(x)$ in C where $C(1 + j*kc) = c_j$ for $j = 0,1,\ldots,n - 1$.

Remarks. It is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris

DIVISION OF POLYNOMIALS

If $p(x) = \sum_{j=0}^{\ell-1} a_j x^j$ and $q(x) = \sum_{j=0}^{m-1} b_j x^j$ where $b_0 \neq 0$, then the following subroutines are available for computing the first n coefficients of the Taylor series $\frac{p(x)}{q(x)} = \Sigma_j c_j x^j$.

<u>CALL PDIV(A,ka,ℓ,B,kb,m,C,kc,n,IERR)</u>
<u>CALL DPDIV(A,ka,ℓ,B,kb,m,C,kc,n,IERR)</u>

A, B, C are arrays. PDIV is used if A, B, C are single precision real arrays and DPDIV is used if A, B, C are double precision arrays.

The arguments ℓ, m, n, ka, kb, kc are positive integers. The coefficients a_j and b_j are assumed to be stored in A and B where

$A(1 + j*ka) = a_j \quad (j = 0,1,\dots,\ell - 1)$
$B(1 + j*kb) = b_j \quad (j = 0,1,\dots,m - 1)$.

IERR is a variable. When the routine is called, if $b_0 \neq 0$ then IERR is assigned the value 0 and the first n coefficients c_j of p(x)/q(x) are stored in C where $C(1 + j*kc) = c_j$ for $j = 0,1,\dots,n - 1$.

Error Return. IERR = 1 if $b_0 = 0$. In this case, no computation is performed.

Remark. It is assumed that the array C does not overlap with the arrays A and B.

Programmer. A. H. Morris

REAL POWERS OF POLYNOMIALS

If r is real and $p(x) = \sum_{j=0}^{m-1} a_j x^j$ where $a_0 > 0$, then the following subroutines are available for computing the first n coefficients of the Taylor series $p(x)^r = \Sigma_j b_j x^j$.

CALL PLPWR(r,A,ka,m,B,kb,n,IERR)
CALL DPLPWR(r,A,ka,m,B,kb,n,IERR)

A and B are arrays. PLPWR is used if A and B are single precision real arrays and r a real number, and DPLPWR is used if A and B are double precision arrays and r a double precision number.

The arguments m, n, ka, kb are positive integers. The coefficients a_j are assumed to be stored in A where $A(1 + j*ka) = a_j$ for $j = 0,1,\ldots,m-1$. IERR is a variable. When the routine is called, if $a_0 > 0$ then IERR is assigned the value 0 and the first n coefficients b_j of $p(x)^r$ are stored in B where $B(1 + j*kb) = b_j$ for $j = 0,1,\ldots,n-1$.

Error Return. IERR = 1 if $a_0 \leqslant 0$. In this case, no computation is performed.

Remark. It is assumed that the arrays A and B do not overlap.

Algorithm. If $q = p^r$ then $pq' = rqp'$ where p' and q' are the derivatives of p and q. Consequently, $b_j = \frac{1}{ja_0} \sum_{i=1}^{j} (ri + i - j)\, a_i b_{j-i}$ is used for $j \geqslant 1$. Also $b_0 = a_0^r$.

Programmer. A. H. Morris

DERIVATIVES AND INTEGRALS OF POLYNOMIALS

Let $f(x) = \sum_{i=0}^{n-1} a_i x^i$ be a polynomial with real coefficients a_i. The polynomial can be differentiated and integrated by the following subroutine:

<u>CALL MPLNMV(MO,x_0,n,A,Y)</u>

A is an array containing the coefficients a_i where $A(i) = a_{i-1}$ for $i = 1,...,n$. The argument x_0 is an arbitrary real number and Y is a variable. MO may have the values -1, 0, 1, 2. When MPLNMV is called Y is assigned the value:

$$Y = \begin{cases} \int_0^{x_0} f(x)\,dx & \text{if } MO = -1 \\ f(x_0) & \text{if } MO = 0 \\ f'(x_0) & \text{if } MO = 1 \\ f''(x_0) & \text{if } MO = 2 \end{cases}$$

Programmer. A. V. Hershey

LAGRANGE POLYNOMIALS

Let $a_1,\dots,a_n$ be n distinct real numbers. Then the i^{th} ***Lagrange polynomial*** is defined by

$$\phi_i(x) = \frac{(x - a_1)(x - a_2)\cdots(x - a_{i-1})(x - a_{i+1})\cdots(x - a_n)}{(a_i - a_1)(a_i - a_2)\cdots(a_i - a_{i-1})(a_i - a_{i+1})\cdots(a_i - a_n)}$$

for $i = 1,2,\dots,n$. The Lagrange polynomials have the property that $\phi_i(a_i) = 1$, $\phi_i(a_j) = 0$ for $j \neq i$, and $p(x) = \sum_{i=1}^{n} p(a_i)\phi_i(x)$ for any polynomial of degree $n - 1$. For convenience,

$$d_i = (a_i - a_1)(a_i - a_2)\cdots(a_i - a_{i-1})(a_i - a_{i+1})\cdots(a_i - a_n)$$

is called the ***normalization divisor*** of $\phi_i(x)$. The following subroutines are available for computing the Lagrange polynomials and their normalization divisors.

CALL LGRNGN(A,n,D)

A and D are arrays of dimension n. The arguments $a_1,\dots,a_n$ are given in the array A. The normalization divisors $d_1,\dots,d_n$ are computed by the routine and stored in D.

Programmer. A. V. Hershey

CALL LGRNGV(MO, n, x_0, A, D, F, DF, DDF)

A and D are arrays of dimension n. The arguments $a_1,\dots,a_n$ are given in A and the normalization divisors $d_1,\dots,d_n$ are given in D. The argument x_0 is an arbitrary real number and F,DF,DDF are arrays of dimension n.

The argument MO may take the values 0,1,2. If MO = 0 then the polynomials $\phi_i(x)$ are evaluated and stored in the array F. If MO = 1 then both the function $\phi_i(x)$ and its derivative $\phi_i'(x)$ are computed at $x = x_0$. In this case $\phi_i(x_0)$ is stored in F(i) and $\phi_i'(x_0)$ is stored in DF(i) for $i = 1,\dots,n$. Similarly, if MO = 2 then the function $\phi_i(x)$ and its first and second derivatives are computed at x_0. The values $\phi_i(x_0)$ are stored in F, the first derivatives are stored in DF, and the second derivatives are stored in DDF.

Note. If MO = 0 then the variables DF and DDF can be ignored and we can write:
CALL LGRNGV(0, n, x_0, A, D, F)
Similarly, if MO = 1 then DDF can be ignored.

Programmer. A. V. Hershey

CALL LGRNGX(A,n,C)

A is an array of dimension n and C is an array of dimension n $\times$ (n + 1). The arguments $a_1,\ldots,a_n$ are given in A. The purpose of the routine is to compute the coefficients c_{ij} of each Lagrange polynomial

$$\phi_j(x) = \sum_{k=0}^{n-1} c_{k+1,j} x^k .$$

When the routine is called, the coefficients of $\phi_j(x)$ are stored in the jth column of C for $j \leqslant n$. Also, the first n coefficients of the polynomial

$$g(x) = (x - a_1) \cdots (x - a_n)$$

are stored in the (n + 1)-st column of C.

Programmer. A. V. Hershey

ORTHOGONAL POLYNOMIALS ON FINITE SETS

Let $u_1,\dots,u_n$ be n distinct real numbers. For any real-valued functions f,g defined on the points u_i let $(f,g) = \sum_{i=1}^{n} f(u_i)g(u_i)$. Then (f,g) is an inner product when f and g are regarded as functions defined only on u_i. Thus, an orthonormal set of polynomials $\{\phi_0, \phi_1,\dots,\phi_{n-1}\}$ exists where the degree of ϕ_j is j for $j < n$. The polynomials ϕ_j are defined recursively by

$$\phi_{j+1}(u) = \frac{1}{a_j}\left[(u - b_j)\phi_j(u) - a_{j-1}\phi_{j-1}(u)\right]$$

where $a_j = (\phi_{j+1}, u\phi_j)$ and $b_j = (\phi_j, u\phi_j)$. Here it is assumed that $\phi_{-1} = a_{-1} = 0$ and $\phi_0(u) = 1/\sqrt{n}$. The following subroutines are available for computing these polynomials.

CALL ORTHOS(U,m,P,n,R)

U is an array containing the values $u_1,\dots,u_n$ and m is an integer such that $1 \leqslant m \leqslant n$. P is an array of dimension n X m and R an array of dimension 2m – 2. When ORTHOS is called, $\phi_{j-1}(u_i)$ is computed and stored in P(i,j) for $i \leqslant n$ and $j \leqslant m$. Also the coefficients $a_0,b_0,a_1,b_1,\dots,a_{m-2},b_{m-2}$ are stored in R.

Programmer. A. V. Hershey.

CALL ORTHOV(MO,n,u,R,m,F,DF,DDF)

The argument u is a real number and m an integer such that $1 \leqslant m \leqslant n$. R is an array containing the coefficients $a_0,b_0,a_1,b_1,\dots,a_{m-2},b_{m-2}$ and F,DF,DDF are arrays of dimension m.

MO may take the values 0,1,2. If MO = 0 then $\phi_0,\phi_1,\dots,\phi_{m-1}$ are evaluated at u and stored in F. If MO = 1 then both ϕ_{j-1} and its derivative ϕ'_{j-1} are computed at point u. In this case $\phi_{j-1}(u)$ is stored in F(j) and $\phi'_{j-1}(u)$ is stored in DF(j) for j = 1,...,m. Similarly, if MO = 2 then ϕ_{j-1} and its first and second derivatives are evaluted at u. The function values $\phi_{j-1}(u)$ are stored in F, the first derivatives are stored in DF, and the second derivatives are stored in DDF.

Note. If MO = 0 then DF and DDF can be ignored and we can write:
CALL ORTHOV(0,n,u,R,m,F)
Similarly, if MO = 1 then DDF can be ignored.

Programmer. A. V. Hershey

CALL ORTHOX (n,R,m,C)

The argument m is an integer such that $1 \leqslant m \leqslant n$. R is an array containing the coefficients $a_0, b_0, a_1, b_1, \ldots, a_{m-2}, b_{m-2}$ and C an array of dimension $m \times m$. The purpose of the routine is to compute the coefficients c_{ij} of each polynomial

$$\phi_{j-1}(u) = \sum_{k=0}^{m-1} c_{k+1,j} u^k \qquad (j = 1,\ldots,m).$$

When ORTHOX is called, the coefficients of ϕ_{j-1} are stored in the jth column of C.

Programmer. A. V. Hershey

ZEROS OF CONTINUOUS FUNCTIONS

Let F(x) be a continuous real-valued function defined for $a \leqslant x \leqslant b$, and assume that F(a) and F(b) have opposite signs. Then the following function is available for finding a point x in the interval [a,b] for which F(x) = 0.

ZEROIN (F,a,b,AERR,RERR)

The arguments AERR and RERR are the absolute and relative error tolerances that are to be satisfied ($AERR \geqslant 0$ and $RERR \geqslant 0$). ZEROIN returns a value x in the interval [a,b] for which F(x) = 0. The result x is accurate to within $\max \{RERR, 4\epsilon\} \cdot |x| + AERR$ where ϵ is the smallest number for which $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6000-7000 series computers).

Note. The function F must be declared in the calling program to be of type EXTERNAL.

Programming. ZEROIN is a slightly modified translation of the ALGOL 60 procedure ZERO given in reference (1). The code was distributed by G. E. Forsythe, M. A. Malcolm, and C. B. Moler (University of New Mexico), and modified by A. H. Morris. The function SPMPAR is called.

References

(1) Brent, Richard, ***Algorithms for Minimization without Derivatives,*** Prentice-Hall, 1973.

(2) Forsythe, G. E., Malcolm, M. A., and Moler, C. B., ***Computer Methods for Mathematical Computations,*** Prentice-Hall, 1977.

SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS

Let $f_i(x) = 0$ $(i = 1,...,n)$ denote a system of n equations in n unknowns where $x = (x_1,...,x_n)$. Assume that each $f_i(x)$ is differentiable and that an initial guess $a = (a_1,..., a_n)$ to a solution of the equations is given. Then the following subroutine is available for solving the equations to within a specified tolerance.

<u>CALL HBRD(F,n,X,FVEC,EPS,TOL,INFO,WK,ℓ)</u>

X and FVEC are arrays of dimension n or larger. On input X contains the starting point $a = (a_1,...,a_n)$. When HBRD terminates, X contains the final estimate $x = (x_1,...,x_n)$ of the solution vector and FVEC contains the values of the functions $f_1,...,f_n$ at the output point in X.

The argument F is the name of a user defined subroutine that has the format:
CALL F(n,X,FVEC,IFLAG)
Here X and FVEC are arrays of dimension n and IFLAG is an integer variable. The array X contains a point $x = (x_1,...,x_n)$. Normally F will evaluate the functions $f_1,...,f_n$ at this point and store the results in FVEC. However, if x does not lie in the domain of $f_1,...,f_n$ then this cannot be done. In this case, the argument IFLAG (which will have been assigned a nonnegative value by HBRD) should be reset by F to a negative value. This will signal HBRD to terminate. F must be declared in the calling program to be of type EXTERNAL.

EPS is an input argument which specifies the relative accuracy of F. If it is estimated that the subroutine F produces results accurate to k significant decimal digits then one may set EPS = 10^{-k}. It is required that EPS $\geqslant 0$. If EPS = 0 then it is assumed that F produces results accurate to machine precision.

TOL is an input argument which specifies the desired accuracy of the solution. The Euclidean norm $\|x\| = \sqrt{\Sigma_i x_i^2}$ is employed. If $\bar{x}$ denotes an actual solution of the equations, then HBRD terminates when an iterate x is generated for which it is estimated that $\|x - \bar{x}\| \leqslant \text{TOL} \cdot \|\bar{x}\|$ is satisfied. It is required that TOL $\geqslant 0$. In order for the convergence test to work properly, it is recommended that TOL always be smaller than 10^{-5}.

WK is an array of dimension ℓ that is used for a work space. It is assumed that the argument ℓ is greater than or equal to $n(3n + 13)/2$.

INFO is an integer variable that reports the status of the results. When HBRD terminates, INFO will have one of the following values:

INFO < 0	This occurs when the user terminates the execution of HBRD by resetting the argument IFLAG in the subroutine F to a negative value. Then INFO = the negative value of IFLAG.

INFO = 0 (Input Error) Either $n \geqslant 1$, EPS $\geqslant 0$, TOL $\geqslant 0$, or $\ell \geqslant n(3n + 13)/2$ is violated.

INFO = 1 A solution having the desired accuracy was obtained.

INFO = 2 The number of calls to the subroutine F has reached or exceeded $200(n + 1)$.

INFO = 3 TOL is too small. No further improvement in the accuracy of x is possible.

INFO = 4 The routine is making very poor progress.

When HBRD terminates, if INFO $\neq 0$ then X contains the final iterate that was generated. Also, if INFO $\geqslant 1$ then FVEC contains the values of the functions $f_1,\ldots,f_n$ at this iterate. If INFO = 2 then it may be helpful to continue the procedure by recalling HBRD with the current point in X as the new starting point. However, this is not advisable when INFO = 4. This setting can arise when $x = 0$ is a solution or when entrapment occurs. HBRD searches for a solution to the equations by minimizing $\Sigma_i f_i(x)^2$. In doing this, it can become trapped in a region where the minimum does not correspond to a solution of the equations. This is what occurs when the equations have no solution. When entrapment occurs and the equations are known to have a solution, then it is recommended that the user try a different starting point.

Scaling. If the convergence criterion $\|x - \bar{x}\| \leqslant \text{TOL} \cdot \|\bar{x}\|$ is satisfied and TOL = 10^{-k}, then the larger components of the final iterate x may be accurate to k significant digits but not the smaller components. For example, if TOL = 10^{-5} and x = (1.2, .34E-4), then 1.2 may be accurate to 5 significant digits while .34E-4 is accurate to only 1 significant digit. If it is suspected that the smaller components do not have acceptable accuracy, then it is recommended that the variables in the original problem be rescaled and the problem rerun.

Algorithm. A modified form of the hybrid Powell procedure is employed.

Programming. HBRD is a slightly modified version of the MINPACK-1 subroutine HYBRD1. The MINPACK-1 subroutines HYBRD, SPMPAR, ENORM, DOGLEG, FDJAC1, QFORM, QRFAC, R1MPYQ, and R1UPDT are employed. The subroutines were written by Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom (Argonne National Laboratory).

References

(1) More, J. J., Garbow, B. S., and Hillstrom, K. E., ***User Guide for MINPACK-1***, Argonne National Laboratory Report ANL-80-74, Argonne, Illinois, 1980.

(2) Powell, M. J. D., "A Hybrid Method for Nonlinear Equations," ***Numerical Methods for Nonlinear Algebraic Equations***, P. Rabinowitz (ed.), Gordon and Breach, London, 1970.

SOLUTIONS OF QUADRATIC, CUBIC, AND QUARTIC EQUATIONS

Given a polynomial $a_0 + a_1 z + \cdots + a_n z^n$ with real coefficients where $a_n \neq 0$ and $n = 2,3$, or 4. The following subroutines are available for computing the roots $z_1, \ldots, z_n$ of the polynomial.

<u>CALL QDCRT(A,Z)</u>
<u>CALL CBCRT(A,Z)</u>
<u>CALL QTCRT(A,Z)</u>

QDCRT is used if $n = 2$, CBCRT is used if $n = 3$, and QTCRT is used if $n = 4$. A is the array of coefficients where $A(k) = a_{k-1}$ for $k = 1,2,\ldots,n+1$, and Z is a complex array of dimension n or larger. When the appropriate subroutine is called, the roots $z_1, \ldots, z_n$ are computed and stored in Z. The real roots precede the complex roots. The real roots are ordered so that $|z_i| \leqslant |z_{i+1}|$.

Algoritims. CBCRT employs the Tartaglia procedure and QTCRT employs the Ferrari procedure.

Programming. QTCRT calls the subroutines CBCRT and AORD, and CBCRT calls the subroutine QDCRT and function CBRT. The routines were written by A. H. Morris. The function SPMPAR is also used.

DOUBLE PRECISION ROOTS OF A REAL POLYNOMIAL

Given a polynomial $a_0 + a_1 z + \cdots + a_n z^n$ of degree $n \geqslant 1$ with real coefficients in double precision. The subroutine RPOLY computes the roots $z_1, \ldots, z_n$ of the polynomial in double precision.

CALL RPOLY (A, N, ZR, ZI, FAIL)

A is a double precision array of dimension n + 1, N is an integer variable, and ZR and ZI are double precision arrays of dimension n. It is assumed that $A(j) = a_{n-j+1}$ for $j = 1, \ldots, n+1$ and that $N = n$ where $1 \leqslant n \leqslant 100$. When RPOLY is called, if all the roots $z_1, \ldots, z_n$ are found then $N = n$ on output. Also, the real parts of the roots are stored in ZR(1),...,ZR(n) and the imaginary parts are stored in ZI(1),...,ZI(n). The roots are unordered except that complex conjugate pairs of roots appear consecutively with the positive imaginary part being first.

Error Return. FAIL is a logical variable. If n roots are found then FAIL is set to .FALSE. Otherwise, if the leading coefficient a_n is 0 or RPOLY cannot find all the roots, then FAIL is set to .TRUE. If $a_n = 0$ then N is reset to 0. Otherwise, if $a_n \neq 0$ and RPOLY finds m roots $z_1, \ldots, z_m$, then N is reset to m. In this case, if $m > 0$ then the m roots are stored in the ZR and ZI arrays.

Programming. RPOLY employs the subroutines FXSHFR, QUADIT, REALIT, CALCSC, NEXTK, NEWEST, QUADSD, and QUADPL. These routines exchange information in a labeled common block named GLOBAL. The routines were written by M. A. Jenkins (Queen's University, Kingston, Ontario). The functions SPMPAR, DPMPAR, and I1MACH are also used.

References

(1) Jenkins, M. A., "Zeros of a Real Polynomial," ***ACM Trans. Math Software 1*** (1975), pp. 178-189.

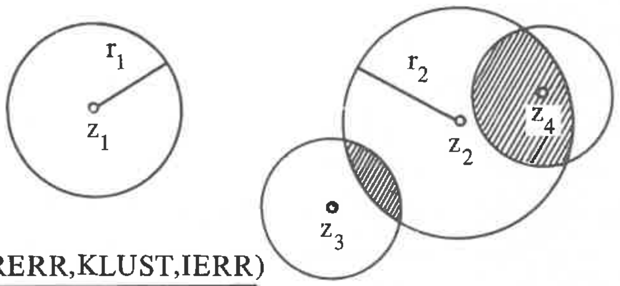
(2) Jenkins, M. A. and Traub, J. F., "A Three-Stage Algorithm for Real Polynomials using Quadratic Iterations," ***SIAM J. Numerical Analysis 7*** (1970), pp. 545-566.

ACCURACY OF THE ROOTS OF A REAL POLYNOMIAL

Given a polynomial $p(z) = a_0 + a_1 z + \cdots + a_n z^n$ of degree $n \geq 1$ with real coefficients. Let $z_1,\ldots,z_n$ be approximations for the roots of $p(z)$. Then for each z_i, the subroutine RBND obtains the radius r_i of a disk $D_i = \{z : |z - z_i| \leq r_i\}$ centered at z_i which contains a true zero of the polynomial. The radius r_i is an upper bound on the absolute error of the approximation z_i.

For each z_i, RBND also computes the number k_i of disks D_j (including the disk D_i) which overlap with D_i. The value k_i is the number of roots of $p(z)$ that are clustered near z_i. If $k_i = 1$ then z_i approximates a simple root.

Example. In the figure $k_1 = 1, k_2 = 3, k_3 = 2,$ and $k_4 = 2.$



CALL RBND(n,A,Z,RADIUS,RERR,KLUST,IERR)

A is a real array containing the coefficients $a_0,a_1,\ldots,a_n$ and Z a complex array containing the approximate roots $z_1,\ldots,z_n$. It is assumed that $n \geq 1$ and $A(i) = a_{i-1}$ for $i = 1,\ldots,n+1$.

IERR is an integer variable, RADIUS a real array of dimension n or larger, and KLUST an integer array of dimension n or larger. When RBND is called, if no input errors are detected then IERR is assigned the value 0, the radii $r_1,\ldots,r_n$ are computed and stored in RADIUS, and the values $k_1,\ldots,k_n$ are computed and stored in KLUST.

RERR is a real array of dimension n or larger. If $z_i = 0$ then RERR(i) is set to -1 by the routine. Otherwise, if $z_i \neq 0$ then RERR(i) = the estimated maximum relative error of z_i.

Error Return. IERR = 1 when $n < 1$ and IERR = 2 when $a_n = 0$. In these cases no computation is performed.

Programming. RBND employs the functions CPABS and SPMPAR. RBND was written by Carl B. Bailey and modified by William R. Gavin (Sandia Laboratories). The format of the routine was modified by A. H. Morris.

COPYING VECTORS

A copy of a vector $X = (x_1,...,x_n)$ can be made and stored in the array Y by the following subroutines:

CALL SCOPY (n,X,kx,Y,ky)
CALL DCOPY (n,X,kx,Y,ky)
CALL CCOPY (n,X,kx,Y,ky)

The input argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geqslant 0$ then it is assumed that x_i is stored in $X(1 + (i-1) * kx)$ for $i = 1,...,n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1 + (n-i) * |kx|)$. Similarly, the input argument ky specifies the spacing of the components of Y.

SCOPY is used if X and Y are single precision real arrays, DCOPY is used if X and Y are double precision real arrays, and CCOPY is used if X and Y are complex arrays. When any of these routines is called, if $n \leqslant 0$ then the routine immediately terminates. Otherwise, if $n \geqslant 1$ then the components x_i of X are stored in Y according to the spacing specified by the ky parameter.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The copy routines were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage***. Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

INTERCHANGING VECTORS

The components of two vectors $X = (x_1,...,x_n)$ and $Y = (y_1,...,y_n)$ can be interchanged by the following subroutines:

CALL SSWAP (n,X,kx,Y,ky)
CALL DSWAP (n,X,kx,Y,ky)
CALL CSWAP (n,X,kx,Y,ky)

The input argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geqslant 0$ then it is assumed that x_i is stored in $X(1 + (i - 1) * kx)$ for $i = 1, ..., n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1 + (n - i) * |kx|)$. Similarly, the input argument ky specifies the spacing of the components of Y.

SSWAP is used if X and Y are single precision real arrays, DSWAP is used if X and Y are double precision real arrays, and CSWAP is used if X and Y are complex arrays. When any of these routines is called, if $n \leqslant 0$ then the routine immediately terminates. Otherwise, if $n \geqslant 1$ then the components x_i and y_i are interchanged for $i = 1,...,n$. Thus, when the routine terminates $X = (y_1,...,y_n)$ and $Y = (x_1,...,x_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The interchange routines were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage.*** Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

PLANAR ROTATION OF VECTORS

Let $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$ be vectors and c and s be real numbers such that $c^2 + s^2 = 1$. X and Y can be replaced with $cX + sY$ and $-sX + cY$ by the following subroutines:

<u>CALL SROT (n,X,kx,Y,ky,c,s)</u>
<u>CALL DROT (n,X,kx,Y,ky,c,s)</u>
<u>CALL CSROT (n,X,kx,Y,ky,c,s)</u>

The input argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geqslant 0$ then it is assumed that x_i is stored in $X(1 + (i - 1) * kx)$ for i = 1, ..., n. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1 + (n - i) * |kx|)$. Similarly, the input argument ky specifies the spacing of the components of Y.

SROT is used if X and Y are single precision real arrays, DROT is used if X and Y are double precision real arrays, and CSROT is used if X and Y are complex arrays. The arguments c and s are single precision real numbers when SROT and CSROT are used, and are double precision real numbers when DROT is used. When any of these routines is called, if $n \leqslant 0$ then the routine immediately terminates. Otherwise, if $n \geqslant 1$ then the components x_i and y_i are replaced with $cx_i + sy_i$ and $-sx_i + cy_i$ for i = 1, ..., n.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The rotation routines were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krough, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage.*** Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

DOT PRODUCTS OF VECTORS

The following functions are available for computing the sums $\sum_i x_i y_i$ and $\sum_i \bar{x}_i y_i$ where $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ are real or complex vectors.

SDOT (n,X,kx,Y,ky)
DDOT (n,X,kx,Y,ky)
CDOTC (n,X,kx,Y,ky)
CDOTU (n,X,kx,Y,ky)

The input argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geqslant 0$ then it is assumed that x_i is stored in $X(1 + (i - 1) * kx)$ for $i = 1, \ldots, n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1 + (n - i) * |kx|)$. Similarly, the input argument ky specifies the spacing of the components of Y.

SDOT is used if X and Y are single precision real arrays, and DDOT is used if X and Y are double precision real arrays. SDOT is a single precision real function and DDOT is a double precision real function. When either of these two functions is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then the function is assigned the value $\sum_{i=1}^{n} x_i y_i$.

CDOTC and CDOTU are used if X and Y are complex arrays. CDOTC and CDOTU are complex functions. When either of these two functions is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then CDOTC (n,X,kx,Y,ky) is assigned the value $\sum_{i=1}^{n} \bar{x}_i y_i$ and CDOTU (n,X,kx,Y,ky) is assigned the value $\sum_{i=1}^{n} x_i y_i$.

Remarks

(1) It should be noted that the mapping $(X,Y) \rightarrow \Sigma_i\, x_i y_i$ computed by CDOTU is not an inner product. However, the mapping is a symmetric bilinear form.

(2) In SDOT, CDOTC, CDOTU all calculations are performed in single precision.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The dot product functions were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage***. Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

SCALING VECTORS

If a is a real or complex number and $X = (x_1,...,x_n)$ a vector, then the vector X can be replaced with the vector aX by the following subroutines:

CALL SSCAL (n,a,X,kx)
CALL DSCAL (n,a,X,kx)
CALL CSCAL (n,a,X,kx)
CALL CSSCAL (n,a,X,kx)

The input argument kx is a positive integer. It is assumed that the component x_i is stored in $X(1 + (i-1) * kx)$ for $i = 1,...,n$.

SSCAL is used if a is a single precision real number and X is a single precision real array, DSCAL is used if a is a double precision real number and X is a double precision real array, CSCAL is used if a is a complex number and X is a complex array, and CSSCAL is used if a is a single precision real number and X is a complex array. When any of these routines is called, if $n \leqslant 0$ then the routine immediately terminates. Otherwise, if $n \geqslant 1$ then each x_i is replaced with ax_i. Thus when the routine terminates $X = (ax_1,...,ax_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The scaling routines were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage.*** Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

VECTOR ADDITION

If a is a real or complex number and $X = (x_1,...,x_n)$ a vector, then any vector $Y = (y_1,...,y_n)$ can be replaced with the vector $aX + Y$ by the following subroutines:

```
CALL SAXPY (n,a,X,kx,Y,ky)
CALL DAXPY (n,a,X,kx,Y,ky)
CALL CAXPY (n,a,X,kx,Y,ky)
```

The input argument kx is an integer which specifies the interval between successive components x_i of the vector X. If $kx \geqslant 0$ then it is assumed that x_i is stored in $X(1 + (i - 1) * kx)$ for $i = 1,...,n$. Otherwise, if $kx < 0$ then it is assumed that x_i is stored in $X(1 + (n - i) * |kx|)$. Similarly, the argument ky specifies the spacing of the components of the vector Y.

SAXPY is used if a is a single precision real number and X, Y are single precision real arrays, DAXPY is used if a is a double precision real number and X, Y are double precision real arrays, and CAXPY is used if a is a complex number and X, Y are complex arrays. When any of these routines is called, if $n \leqslant 0$ or $a = 0$ then the routine immediately terminates. Otherwise, if $n \geqslant 1$ then y_i is replaced with $ax_i + y_i$ for $i = 1,...,n$. Thus when the routine terminates $Y = (ax_1 + y_1,...,ax_n + y_n)$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The addition routines were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage.*** Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

L_1 NORM OF A VECTOR

The following functions are available for computing the L_1 norm of a real vector or a modified L_1 norm of a complex vector.

SASUM (n,X,kx)
DASUM (n,X,kx)
SCASUM (n,X,kx)

$X = (x_1,...,x_n)$ is a vector and the input argument kx is a positive integer. It is assumed that x_i is stored in $X(1 + (i-1) * kx)$ for $i = 1,...,n$.

SASUM is used if X is a single precision real array and DASUM is used if X is a double precision real array. SASUM is a single precision real function and DASUM is a double precision real function. When either of these two functions is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then the function is assigned the value $\sum_{i=1}^{n} |x_i|$.

SCASUM is used if X is a complex array. SCASUM is a single precision real function. When SCASUM is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then SCASUM (n,X,kx) is assigned the value $\sum_{i=1}^{n} [|Re(x_i)| + |IM(x_i)|]$.

Note. SCASUM (n,X,kx) is the norm of the complex vector $X = (x_1,...,x_n)$ when X is regarded as a real vector of dimension 2n. This norm is frequently preferred over the standard L_1 norm $\sum_{i=1}^{n} |x_i|$ since it takes less time to compute.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The L_1 Norm functions were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage***. Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

L_2 NORM OF A VECTOR

The following functions are available for computing the L_2 norm of a real or complex vector.

<u>SNRM2 (n,X,kx)</u>
<u>DNRM2 (n,X,kx)</u>
<u>SCNRM2 (n,X,kx)</u>

$X = (x_1,\dots,x_n)$ is a vector and kx is a positive integer. It is assumed that the component x_i is stored in $X(1 + (i-1) * kx)$ for $i = 1,\dots,n$.

SNRM2 is used if X is a single precision real array, DNRM2 is used if X is a double precision real array, and SCNRM2 is used if X is a complex array. SNRM2 and SCNRM2 are single precision real functions, and DNRM2 is a double precision real function. When any of these functions is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then the function is assigned the value $\left[\sum_{i=1}^{n} |x_i|^2\right]^{1/2}$.

Programming. These routines are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The L_2 norm functions were coded by Charles Lawson (Jet Propulsion Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage.*** Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

L_∞ NORM OF A VECTOR

The following functions are available for finding the largest component x_i of a vector $X = (x_1,...,x_n)$.

ISAMAX (n,X,kx)
IDAMAX (n,X,kx)
ICAMAX (n,X,kx)

The input argument kx is a positive integer. It is assumed that the component x_i is stored in $X(1 + (i - 1) * kx)$ for $i = 1,...,n$.

ISAMAX is used if X is a single precision real array and IDAMAX is used if X is a double precision real array. ISAMAX and IDAMAX are integer functions. When either of these two functions is called, if $n \leqslant 0$ then the function is assigned the value 0. Otherwise, if $n \geqslant 1$ then the function is assigned the value i where i is the smallest index such that $|x_i| = \max\{|x_j| : j = 1,...,n\}$.

ICAMAX is also an integer function. It is used when X is a complex array. If $n \leqslant 0$ then ICAMAX(n,X,kx) is assigned the value 0. Otherwise, if $n \geqslant 1$ then the function is assigned the value i where i is the smallest index such that $|\mathrm{Re}(x_i)|+|\mathrm{Im}(x_i)| = \max\{|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| : j = 1,...,n\}$.

Note. The mapping $X \to \max\{|\mathrm{Re}(x_j)|+|\mathrm{Im}(x_j)| : j = 1,...,n\}$ is the L_∞ norm of the complex vector $X = (x_1,...,x_n)$ when X is regarded as a real $n \times 2$ matrix. This norm is frequently preferred over the standard L_∞ norm $\max\{|x_j| : j = 1,...,n\}$ since it takes less time to compute.

Programming. These functions are part of the BLAS package of basic linear algebra subroutines designed by C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. The L_∞ Norm functions were coded by Jack Dongarra (Argonne National Laboratory).

Reference. Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., ***Basic Linear Algebra Subprograms for FORTRAN Usage***. Report SAND 77-0898, Sandia Laboratories, Albuquerque, New Mexico, 1977.

PACKING AND UNPACKING SYMMETRIC MATRICES

An n X n symmetric matrix $A = (a_{ij})$ can be represented by either its lower triangular elements

$$\left(\begin{array}{cccc} a_{11} & a_{12} & a_{13} & \cdots\cdot \\ \cline{1-1} a_{21} & a_{22} & a_{23} & \cdots\cdot \\ \cline{1-2} a_{31} & a_{32} & a_{33} & \cdots\cdot \\ \cline{1-3} \vdots & \vdots & \vdots & \end{array}\right)$$

or its upper triangular elements. If the lower triangular elements are used, then the packed form for the matrix is an array of dimension $n(n + 1)/2$ containing the elements $a_{11}a_{21}a_{22}a_{31}a_{32}a_{33}$ .... Similarly, if the upper triangular elements are used then the packed form for the matrix is an array containing $a_{11}a_{12} \cdots a_{1n}a_{21}a_{22} \cdots a_{2n} \cdots$ ***Currently the lower triangular elements are used for packing symmetric matrices.*** The following subroutines are available for packing and unpacking real symmetric matrices.

```
CALL MCVFS (A,ka,n,B)
CALL DMCVFS (A,ka,n,B)
```

A is an n X n symmetric matrix and B an array whose dimension is equal to or greater than $n(n + 1)/2$. The routines store the packed form of A in B. MCVFS is used if A and B are single precision real arrays and DMCVFS is used if A and B are double precision real arrays. The input argument ka has the following value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that $ka \geqslant n$.

Note. A and B may begin in the same location.

Programmer. A. H. Morris

```
CALL MCVSF (A,ka,n,B)
CALL DMCVSF (A,ka,n,B)
```

B is an array containing the elements of a packed n X n symmetric matrix and A is an array of dimension ka X n where $ka \geqslant n$. The routines unpack B and store the results in A. MCVSF is used if A and B are single precision real arrays and DMCVSF is used if A and B are double precision real arrays.

Note. The output matrix A may begin in the same location as the array B.

Programmer. A. H. Morris

CONVERSION OF REAL MATRICES TO AND FROM DOUBLE PRECISION FORM

The subroutines MCVRD and MCVDR are available for converting real matrices to and from double precision form.

CALL MCVRD(m,n,A,ka,B,kb)

A is a real single precision $m \times n$ matrix and B a double precision array. MCVRD stores the matrix in double precision form in the array B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant m$.

Programmer. A. H. Morris

CALL MCVDR(m,n,A,ka,B,kb)

A is a double precision $m \times n$ matrix and B a real single precision array. MCVDR stores the matrix in single precision form in the array B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant m$.

Programmer. A. H. Morris

THE REAL AND IMAGINARY PORTIONS OF A COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\mathrm{Re}(A) = (\mathrm{Re}(a_{ij}))$ and $\mathrm{Im}(A) = (\mathrm{Im}(a_{ij}))$ denote the real and imaginary portions of A. The following subroutines are available for obtaining Re(A) and Im(A).

CALL CMREAL (m,n,A,ka,B,kb)

A is a complex $m \times n$ matrix and B a real array. CMREAL obtains Re(A) and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant m$.

Programmer. A. H. Morris

CALL CMIMAG (m,n,A,ka,B,kb)

A is a complex $m \times n$ matrix and B a real array. CMIMAG obtains Im(A) and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant m$.

Programmer. A. H. Morris

COPYING MATRICES

A copy of the matrix A can be made and inserted into B by the following subroutines:

CALL MCOPY(m,n,A,ka,B,kb)

A is a real m $\times$ n matrix and B a real array. MCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
It is assumed that ka $\geqslant$ m and kb $\geqslant$ m.

Programmer. A. H. Morris

CALL SMCOPY (n,A,B)

A is a real packed n $\times$ n symmetric matrix and B is a real array whose dimension is equal to or greater than n(n + 1)/2. SMCOPY makes a copy of the packed symmetric matrix A and stores it in B.

Programmer. A. H. Morris

CALL DMCOPY(m,n,A,ka,B,kb)

A is a double precision m $\times$ n matrix and B a double precision array. DMCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
It is assumed that ka $\geqslant$ m and kb $\geqslant$ m.

Programmer. A. H. Morris

CALL CMCOPY(m,n,A,ka,B,kb)

A is a complex m $\times$ n matrix and B a complex array. CMCOPY makes a copy of the matrix A and stores it in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
It is assumed that ka $\geqslant$ m and kb $\geqslant$ m.

Programmer. A. H. Morris

COMPUTATION OF THE CONJUGATE OF A COMPLEX MATRIX

If $A = (a_{ij})$ is a complex matrix then let $\overline{A} = (\overline{a}_{ij})$ denote the conjugate of A. The following subroutine is available for computing the conjugate matrix $\overline{A}$.

CALL CMCONJ (m,n,A,ka,B,kb)

A is a complex $m \times n$ matrix and B is a complex array. CMCONJ computes $\overline{A}$ and stores the results in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant m$.

Programmer. A. H. Morris

TRANSPOSING MATRICES

The subroutines TPOSE, DTPOSE, CTPOSE and TIP, DTIP, CTIP are available for transposing a matrix A. TPOSE, DTPOSE, and CTPOSE are used if the results are to be stored in a separate storage area B. TIP, DTIP, and CTIP are used if the results are to be stored in A (i.e., if the transposition is to be done in place).

CALL TPOSE(m,n,A,ka,B,kb)
CALL DTPOSE(m,n,A,ka,B,kb)
CALL CTPOSE(m,n,A,ka,B,kb)

TPOSE is called if A is a single precision real matrix and B a single precision real array, DTPOSE is called if A is a double precision real matrix and B a double precision real array, and CTPOSE is called if A is a complex matrix and B a complex array.

A is a matrix having m rows and n columns. The routine transposes A and stores the results in B. The input arguments ka and kb have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
It is assumed that $ka \geqslant m$ and $kb \geqslant n$.

Programmer. A. H. Morris

CALL TIP(A,m,n,MOVE,k,MDIM)
CALL DTIP(A,m,n,MOVE,k,MDIM)
CALL CTIP(A,m,n,MOVE,k,MDIM)

TIP is called if A is a single precision real array, DTIP is called if A is a double precision real array, and CTIP is called if A is a complex array.

A is an array of dimension mn which contains an $m \times n$ matrix to be transposed. The routine transposes the matrix and stores the results in A. If $m = n$ then the arguments MOVE, k, MDIM are ignored.

If $m \neq n$ then k may be any integer. If $k \leqslant 0$ then MOVE is ignored. Otherwise, if $k \geqslant 1$ then MOVE is assumed to be an array of dimension k. MOVE is a storage area for saving information that may help speed up the transposition process. If no information is saved then TIP, DTIP, and CTIP may run 2-10 times slower than TPOSE, DTPOSE, and CTPOSE. However, the use of a storage area may or may not actually speed up the transposition process. This depends entirely on the values of m and n. If $m \neq n$ then

MDIM is a variable that is set by the routine. After the routine terminates, MDIM will be the estimated optimum setting for k for the current values of m and n.

Programming. The routines TIP, DTIP, and CTIP employ the subroutine INFCTR. The routines were written by Norman Brenner (Dept. of Earth and Planetary Sciences, Massachusetts Institute of Technology), and modified by A. H. Morris.

Reference. Brenner, Norman, "Algorithm 467. Matrix Transposition in Place," ***Communications ACM*** *16* (November, 1973), pp. 692–694.

COMPUTING ADJOINTS OF COMPLEX MATRICES

If $A = (a_{ij})$ then let $A^* = (\bar{a}_{ji})$ denote the adjoint of A. The following subroutines are available for computing A*.

<u>CALL CMADJ (m,n,A,ka,B,kb)</u>

A is a complex $m \times n$ matrix and B is a complex array. CMADJ computes A* and stores the results in B. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is assumed that $ka \geqslant m$ and $kb \geqslant n$.

Remarks. CMADJ combines the following operations:

CALL CTPOSE (m,n,A,ka,B,kb)
CALL CMCONJ (n,m,B,kb,B,kb)

It is assumed that A and B are different storage areas.

Programmer. A. H. Morris

<u>CALL CTRANS(ka,n,A)</u>

A is a complex array of dimension $ka \times n$ which contains an $n \times n$ matrix. CTRANS computes the adjoint of the matrix and stores the results in A. It is assumed that $ka \geqslant n$.

Programmer. George J. Davis (Georgia State University)

MATRIX ADDITION

The matrix sum C = A + B can be computed by the following subroutines:

CALL MADD(m,n,A,ka,B,kb,C,kc)

A and B are real m × n matrices and C is a real array. MADD computes A + B and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is assumed that ka $\geqslant$ m, kb $\geqslant$ m, kc $\geqslant$ m.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that kc $\geqslant$ m).

Programmer. A. H. Morris

CALL SMADD(n,A,B,C)

A and B are real packed n × n symmetric matrices and C is a real array whose dimension is equal to or greater than n(n + 1)/2. SMADD computes A + B, which is also a symmetric matrix, and stores the results in packed form in C.

The array C may begin in the same location as A or B. If C begins in the same location as A then the result C will overwrite the input data A. Similarly, B will be overwritten if C begins in the same location as B. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris

CALL DMADD(m,n,A,ka,B,kb,C,kc)

A and B are double precision m × n matrices and C is a double precision array. DMADD computes A + B and stores the results in C. The arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that $ka \geqslant m$, $kb \geqslant m$, $kc \geqslant m$.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that $kc \geqslant m$).

Programmer. A. H. Morris

CALL CMADD(m,n,A,ka,B,kb,C,kC)

A and B are complex $m \times n$ matrices and C is a complex array. CMADD computes A + B and stores the results in C. The arguments ka, kb, kc have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that $ka \geqslant m$, $kb \geqslant m$, $kc \geqslant m$.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restiction on kc (other than the customary restriction that $kc \geqslant m$).

Programmer. A. H. Morris

MATRIX SUBTRACTION

The matrix difference C = A – B can be computed by the following subroutines:

CALL MSUBT(m,n,A,ka,B,kb,C,kc)

A and B are real m × n matrices and C is a real array. MSUBT computes A – B and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that ka $\geqslant$ m, kb $\geqslant$ m, kc $\geqslant$ m.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that kc $\geqslant$ m).

Programmer. A. H. Morris

CALL SMSUBT (n,A,B,C)

A and B are real packed n × n symmetric matrices and C is a real array whose dimension is equal to or greater than n(n + 1)/2. SMSUBT computes A–B, which is also a symmetric matrix, and stores the results in packed form in C.

The array C may begin in the same location as A or B. If C begins in the same location as A then the result C will overwrite the input data A. Similarly, B will be overwritten if C begins in the same location as B. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris

CALL DMSUBT(m,n,A,ka,B,kb,C,kc)

A and B are double precision m × n matrices and C is a double precision array. DMSUBT computes A–B and stores the results in C. The arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that ka $\geqslant$ m, kb $\geqslant$ m, kc $\geqslant$ m.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that kc $\geqslant$ m).

Programmer. A. H. Morris

CALL CMSUBT(m,n,A,ka,B,kb,C,kc)

A and B are complex m $\times$ n matrices and C is a complex array. CMSUBT computes A$-$B and stores the results in C. The input arguments ka, kb, kc have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is assumed that ka $\geqslant$ m, kb $\geqslant$ m, kc $\geqslant$ m.

The array C may begin in the same location as A or B. If C begins in the same location as A then it is assumed that kc = ka. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restiction on kc (other than the customary restriction that kc $\geqslant$ m).

Programmer. A. H. Morris

MATRIX MULTIPLICATION

The matrix product C = AB can be computed by the following subroutines:

CALL MPROD(m,n,ℓ,A,ka,B,kb,C,kc,WK)
CALL DMPROD(m,n,ℓ,A,ka,B,kb,C,kc,WK)
CALL CMPROD(m,n,ℓ,A,ka,B,kb,C,kc,WK)

MPROD is called if A and B are single precision real matrices and C and WK are single precision real arrays, DMPROD is called if A and B are double precision real matrices and C and WK are double precision real arrays, and CMPROD is called if A and B are complex matrices and C and WK are complex arrays.

A is a matrix having m rows and n columns, and B is a matrix having n rows and ℓ columns. The routine computes the product AB and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is assumed that ka $\geqslant$ m, kb $\geqslant$ n, kc $\geqslant$ m.

WK is an array that serves as a temporary storage area. The matrix C may begin in the same location as A or B. If C begins in the same location as A, then it is assumed that kc = ka and that the dimension of WK is equal to or greater than ℓ. In this case, the result C will overwrite the input data A. Similarly, if C begins in the same location as B then it is assumed that kc = kb and that the dimension of WK is equal to or greater than m. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case, the array WK is not referenced.

Notes.

(1) If C begins in the same location as A or B, then it is assumed that the storage areas for A and B do not overlap.

(2) All inner products $\Sigma_k\ a_{ik} b_{kj}$ are computed by MPROD in double precision.

Programming. MPROD employs the subroutines RLOC and YCHG, DMPROD employs the subroutines DLOC and DYCHG, and CMPROD employs the subroutines CLOC and CYCHG. The routines were written by A. H. Morris.

PRODUCT OF A PACKED SYMMETRIC MATRIX AND A VECTOR

Let A denote a packed symmetric matrix of order n and x a column vector of dimension n where $n \geqslant 1$. Then the following subroutines are available for computing the product Ax.

```
CALL SVPRD(A,n,x,y)
CALL DSVPRD(A,n,x,y)
```

The argument y is an array of dimension n. SVPRD is called when A,x,y are single precision real arrays, and DSVPRD is called when A,x,y are double precision real arrays. When either of these routines is called, Ax is computed and stored in y.

Programmer. A. H. Morris

TRANSPOSE MATRIX PRODUCTS

If A^t denotes the transpose of A, then the matrix product $C = A^tB$ can be computed by the following subroutine:

CALL TMPROD(m,n,ℓ,A,ka,B,kb,C,kc)

A is a real matrix having m rows and n columns, B is a real matrix having m rows and ℓ columns, and C is a real array. TMPROD computes A^tB and stores the results in C. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

Here it is assumed that $ka \geqslant m$, $kb \geqslant m$, $kc \geqslant n$. Also it is assumed that the storage area for C does not overlap with the storage areas for A and B.

Note. All inner products $\sum_k a_{ki}b_{kj}$ are computed in double precision and the results stored in single precision.

Programmer. A. H. Morris

SYMMETRIC MATRIX PRODUCTS

If A^t denotes the transpose of A, then the matrix product A^tA can be computed and its packed form inserted into the array B by the following subroutine:

CALL SMPROD(m,n,A,ka,B)

A is a real m $\times$ n matrix and B a real array whose dimension is equal to or greater than $n(n+1)/2$. SMPROD computes A^tA and stores its packed form in B. The input argument ka has the following value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that ka $\geqslant$ m.

Note. All inner products $\sum_k a_{ki}a_{kj}$ are computed in double precision and the results stored in single precision.

Programmer. A. H. Morris

KRONECKER PRODUCT OF MATRICES

If A is an $m \times n$ matrix and B a $k \times \ell$ matrix, then the ***Kronecker product $A \otimes B$*** is defined by

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} .$$

From this definition we obtain:

(1) (Transpose Equality) $(A \otimes B)^t = A^t \otimes B^t$.

(2) $(A \otimes B) \otimes E = A \otimes (B \otimes E)$ for any matrix E.

(3) $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ if C is a matrix having n rows and D a matrix having ℓ rows.

If A and B are complex square matrices of orders m and k respectively, then from the Jordan canonical forms of A and B the determinant equality $\det(A \otimes B) = (\det A)^k (\det B)^m$ can be verified. Also, if A and B are nonsingular then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ from (3).

The following subroutines are available for computing $C = A \otimes B$.

```
CALL KPROD(A,ka,m,n,B,kb,k,ℓ,C,kc)
CALL DKPROD(A,ka,m,n,B,kb,k,ℓ,C,kc)
CALL CKPROD(A,ka,m,n,B,kb,k,ℓ,C,kc)
```

It is assumed that A is an $m \times n$ matrix, B a $k \times \ell$ matrix, and C a 2-dimensional array. KPROD is used if A, B, C are single precision real arrays, DKPROD is used if A, B, C are double precision arrays, and CKPROD is used if A, B, C are complex arrays. When the routine is called, $A \otimes B$ is computed and stored in C.

The arguments ka, kb, and kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is assumed that $ka \geqslant m$, $kb \geqslant k$, $kc \geqslant mk$, and that C contains at least $n\ell$ columns.

Remark. It is assumed that the array C does not overlap with A or B.

Programmer. A. H. Morris

INVERTING GENERAL REAL MATRICES AND SOLVING GENERAL SYSTEMS OF REAL LINEAR EQUATIONS

The subroutines CROUT, KROUT, MSLV, NPIVOT, and DMSLV are available for inverting real matrices A and solving systems of real linear equations $AX = B$. CROUT, KROUT, MSLV, and NPIVOT solve single precision problems, and DMSLV solves double precision problems.

All the routines except NPIVOT are general-purpose, employing partial pivot Gauss elimination. NPIVOT can only occasionally be used since it uses Gauss-Jordan elimination with no pivot search. Normally, CROUT and KROUT produce the same results, which will be of equal or greater accuracy than the results produced by MSLV. However, since many of the calculations are performed in double precision in CROUT and KROUT, whereas only single precision is used in MSLV, MSLV may run 2-3 times faster than CROUT and KROUT.

CALL CROUT(MO,n,m,A,ka,B,kb,D,INDEX,TEMP)

A is a single precision real matrix of order n where $n \geqslant 1$. If $MO = 0$ then the inverse of A is computed and stored in A. If $MO \neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a single precision real matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then the argument kb is ignored.

D is a single precision real variable. When CROUT is called, D is assigned the value det(A) where det(A) is the determinant of A. If D is found to have the value 0 then the routine immediately terminates.

INDEX is an array of dimension $n - 1$ or larger that is used by the routine for keeping track of the row interchanges that are made. If $MO \neq 0$ then this array is not needed.

TEMP is a single precision real array of dimension n or larger that is a work space for the routine. If $MO \neq 0$ then this array is not needed.

Remarks

(1) KROUT should be used instead of CROUT when the determinant D is not needed. Underflow or overflow in the calculation of D may cause CROUT to terminate prematurely.

(2) If MO $\neq 0$ then one may write:

    CALL CROUT(MO,n,m,A,ka,B,kb,D)

In this case, even though the inverse of A is not computed, the matrix A is destroyed.

Algorithm. The Crout procedure is used. All inner products are computed in double precision and the results returned in single precision. Partial pivoting is performed.

Programming. CROUT calls the subroutine CROUT0. These routines were written by A. H. Morris.

CALL KROUT(MO,n,m,A,ka,B,kb,IERR,INDEX,TEMP)

A is a single precision real matrix of order n where $n \geqslant 1$. If MO = 0 then the inverse of A is computed and stored in A. If MO $\neq 0$ then the inverse is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a single precision real matrix having n rows and m columns. In this case the matrix equation AX = B is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then the argument kb is ignored.

INDEX is an array of dimension $n - 1$ or larger that is used by the routine for keeping track of the row interchanges that are made. If MO $\neq 0$ then this array is not needed.

TEMP is a single precision real array of dimension n or larger that is a work space for the routine. If MO $\neq 0$ then this array is not needed.

Error Return. IERR is a variable that reports the status of the results. When the routine terminates IERR will have one of the following values:

IERR = 0	The requested results were successfully obtained.
IERR = −1	Either n, ka, or kb is incorrect. In this case A and B have not been modified.
IERR = k	The kth column of A has been reduced to a column containing only zeros.

When an error is detected, the routine immediately terminates.

Remark. If MO $\neq 0$ then one may write:

CALL KROUT(MO,n,m,A,ka,B,kb,IERR)

In this case, even though the inverse of A is not computed, the matrix A is destroyed.

Algorithm. The CROUT procedure is used. All inner products are computed in double precision and the results returned in single precision. Partial pivoting is performed.

Programming. KROUT calls the subroutine KROUT0. These routines were written by A. H. Morris.

CALL NPIVOT(n,m,A,ka,B,kb,D,IERR)

A is a single precision real matrix of order n where $n \geqslant 1$. When NPIVOT is called the inverse of A is computed and stored in A.

The argument m is an integer. If $m \geqslant 1$ then B is a single precision real matrix having n rows and m columns. In this case the matrix equation $AX = B$ is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then kb is ignored.

D is a single precision real variable. On input D must be assigned a value by the user. If the input value is τ, then when NPIVOT terminates $D = \tau d$ where d is the determinant of A.

Error Return. IERR is an integer variable. If inversion is successful then IERR is assigned the value 0. Otherwise, IERR = 1 if NPIVOT cannot invert the matrix.

Algorithm. The Gauss-Jordan procedure is used. However, no pivot searching is done. NPIVOT terminates (with IERR set to 1) whenever a zero pivot element is encountered.

Remarks. Since pivot search is frequently needed to invert a matrix, and since pivot search is normally required to obtain accurate results, ***NPIVOT should not be used except on those occasions when pivot search is known to be superfluous.***

Programmer. A. H. Morris

CALL MSLV(MO,n,m,A,ka,B,kb,D,RCOND,IERR,IPVT,WK)
CALL DMSLV(MO,n,m,A,ka,B,kb,D,RCOND,IERR,IPVT,WK)

A is matrix of order n where $n \geqslant 1$. If MO = 0 then the inverse of A is computed and stored in A. If MO $\neq$ 0 then the inverse is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a matrix having n rows and m columns. In this case the matrix equation AX = B is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then the argument kb is ignored.

D is an array of dimension 2. When either routine is called the determinant det(A) of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leqslant |d| < 10$ and k an integer, then d is stored in D(1) and the exponent k is stored in floating point form in D(2).

RCOND is a variable. When either routine is called, the routine makes an estimate c of the condition number of the matrix A (relative to the L_1 norm). RCOND is assigned the value $1/c$.

IPVT is an integer array of dimension n or larger that is used by the routines for keeping track of the row interchanges that are made. WK is an array of dimension n or larger that is used as a work space.

Remarks

(1) If MSLV is called then it is assumed that A and B are single precision real matrices, D and WK are single precision real arrays, and RCOND is a single precision real variable. Otherwise, if DMSLV is called then it is assumed that A and B are double precision real matrices, D and WK are double precision real arrays, and RCOND is a double precision real variable.

(2) RCOND satisifies $0 \leqslant \text{RCOND} \leqslant 1$. If $\text{RCOND} \approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if MSLV is used to invert a matrix in the 14 digit CDC single precision arithmetic and RCOND = .4E−3, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If $\text{RCOND} \approx 1$ then one should expect the results to be almost as accurate as the original data A. However, if $\text{RCOND} \approx 0$ then one should expect the results to be nonsense.

(3) The matrix A is always destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that $1 + \text{RCOND} > 1$, then IERR is set to 0 and the problem is solved. Otherwise, if $1 + \text{RCOND} = 1$ then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant will not have been computed.

Algorithm. The partial pivot Gauss elimination procedure is used.

Programming. MSLV and DMSLV are driver routines for the LINPACK subroutines SGECO, SGEFA, SGESL, SGEDI and DGECO, DGEFA, DGESL, DGEDI. The subroutines were coded by Cleve Moler (University of New Mexico). The subroutines employ the vector routines SSWAP, SDOT, SSCAL, SAXPY, SASUM, ISAMAX and DSWAP, DDOT, DSCAL, DAXPY, DASUM, IDAMAX.

References

(1) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

(2) Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Journal of Numerical Analysis 16*** (1979), pp. 368-375.

SOLUTION OF REAL EQUATIONS WITH ITERATIVE IMPROVEMENT

Given a real n $\times$ n matrix A and a real column vector b. The following routine is available for solving the equation Ax = b. Iterative improvement is performed to compute the solution x to machine accuracy.

CALL SLVMP(MO,n,A,ka,b,X,WK,IWK,IERR)

MO is an input argument which specifies if SLVMP is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional real array of dimension ka $\times$ n containing the matrix A, b a real vector of dimension n, and X a real array of dimension n. When SLVMP is called, Ax = b is solved and the solution stored in X. A and b are not modified by the routine.

WK is a real array of dimension $n^2 + n$ or larger, and IWK an integer array of dimension n or larger. These arrays are for internal use by the routine. On an initial call to SLVMP, an LU decomposition is obtained for A and stored in WK and IWK. Then the equation Ax = b is solved.

IERR is an integer variable that reports the status of the results. On an initial call to SLVMP. when the routine terminates IERR has one of the following values:

IERR = 0	The solution X was obtained to machine accuracy.
IERR = 1	X was obtained, but not to machine accuracy.
IERR = –k	The kth column of A was reduced to a column containing only zeros.

When IERR = –k, no solution is obtained.

After an initial call to SLVMP, if IERR = 0 or 1 on output, then the routine may be called to solve a new set of equations Ax = $\tilde{b}$ without having to redecompose the matrix A. In this case, the input argument MO may be set to any nonzero value. When MO $\neq$ 0 it is assumed that only b has been modified. The routine employs the LU decomposition obtained on the initial call to SLVMP to solve the new set of equations Ax = b. On output X will contain the solution to the new set of equations. As before, A and b are not modified by the routine.

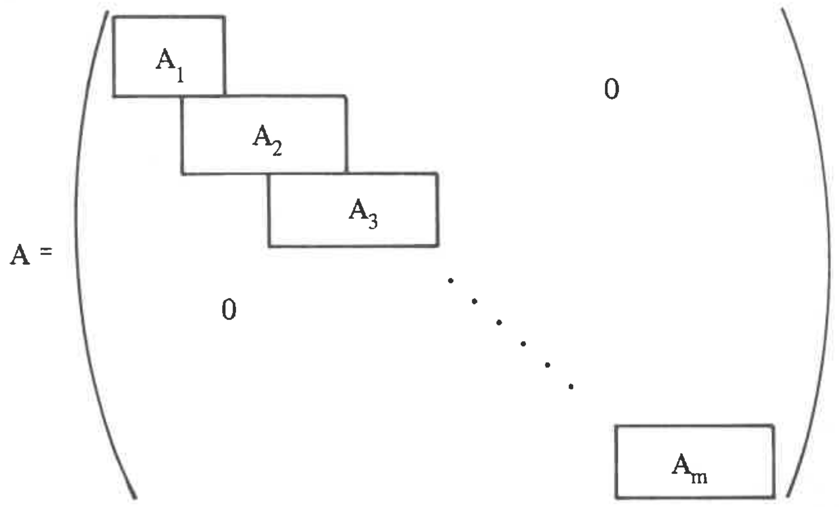
If SLVMP is recalled with MO $\neq$ 0, then when the routine terminates IERR will have one of the following values:

IERR = 0	The solution X was obtained to machine accuracy.
IERR = 1	X was obtained, but not to machine accuracy.

Programming. SLVMP calls the subroutine LUIMP. These routines were written by A. H. Morris. The subroutines MCOPY, SGEFA, SGESL, SSCAL, SAXPY and functions SPMPAR, SDOT, ISAMAX are also employed.

SOLUTION OF ALMOST BLOCK DIAGONAL SYSTEMS OF LINEAR EQUATIONS

Consider a system $Ax = b$ of linear equations where A is an $n \times n$ matrix having the block structure



Here it is assumed that A_i is an $r_i \times c_i$ matrix for $i = 1,\dots,m$, and that A_i and A_{i+1} may have $\delta_i \geqslant 0$ columns in common for $i < m$. Thus $\sum_{i=1}^{m} r_i = n$ and block A_i begins in column $\sum_{k=1}^{i-1} (c_k - \delta_k) + 1$ for $i \geqslant 2$. It is also assumed that three successive blocks A_{i-1}, A_i, A_{i+1} do not have columns in common. Thus $\delta_{i-1} + \delta_i \leqslant c_i$ for $i = 2,\dots,m-1$. If $m \geqslant 2$ then the following subroutines are available for solving $Ax = b$.

<u>CALL ARCECO(n,S,MTR,m,IWK,b,X,IERR)</u>

S is an array of dimension $\sum_{i=1}^{m} r_i c_i$ or larger. On input S contains the blocks $A_1,\dots,A_m$ of the matrix A. The blocks are stored in sequence. A_1 is stored in the first $r_1 c_1$ locations of S, A_2 is stored in the next $r_2 c_2$ locations, etc. For each A_i, the columns of A_i are stored in sequence in S.

Example.

$$\text{If}\quad A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & a_{42} & a_{43} & a_{44} & 0 \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix}$$

then $n = 5$, $m = 3$, $\delta_1 = 2$, and $\delta_2 = 0$. Also, S is an array containing the data a_{11}, a_{21}, a_{12}, a_{22}, a_{13}, a_{23}, a_{32}, a_{42}, a_{33}, a_{43}, a_{34}, a_{44}, a_{55}.

MTR is an integer matrix of dimension 3 X m containing the block information of the matrix A:

MTR(1,i) = r_i (i = 1,...,m)
MTR(2,i) = c_i (i = 1,...,m)
MTR(3,i) = δ_i (i = 1,...,m − 1)

For convenience, the routine sets MTR(3,m) = 0.

X is an array of dimension n or larger. When ARCECO is called, A is decomposed and then the equations Ax = b are solved. The decomposition of A is stored in S, overwriting the initial input data A, and the solution x is stored in X. The vector b is destroyed by the routine.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the decomposition of A are stored in IWK.

IERR is a variable that reports the status of the results. When ARCECO terminates, IERR has one of the following values:

IERR = 0	The system of equations was solved.
IERR = 1	(Input Error) Either n, m, or MTR is incorrect, or three successive blocks A_{i-1}, A_i, A_{i+1} of A have columns in common.
IERR = − 1	A is a singular matrix. The equations cannot be solved.

Usage. After a call to ARCECO, if IERR = 0 on output then the subroutine ARCESL (see below) may be called to solve a new set of equations Ax = $\tilde{b}$ without having to redecompose the matrix A. ARCESL employs the decomposition of A obtained by ARCECO.

Algorithm. A modification of the alternate row and column elimination procedure by Varah is used.

Programming. ARCECO employs the subroutines ARCEDC, ARCEPR, ARCEPC, ARCESL, ARCEFS, ARCEFM, ARCEFE, ARCEBS, ARCEBM, and ARCEBE. These routines were developed by J. C. Diaz (Mobil Research and Development Corp., Farmers Branch, Texas), G. Fairweather(University of Kentucky), and P. Keast (University of Toronto).

Reference. Diaz, J. C., Fairweather, G. and Keast, P., "FORTRAN Packages for Solving Certain Almost Block Diagonal Linear Systems by Modified Alternate Row and Column Elimination," *ACM Trans. Math Software 9* (1983), pp. 358-375.

CALL ARCESL(S,MTR,m,IWK,b,X)

The argument m is the number of blocks $A_1,...,A_m$ in the matrix A. ARCESL may be used only when IERR = 0 on output from ARCECO. In this case, S contains the decomposition of A obtained by ARCECO and IWK contains the pivot indices of the decomposition. The argument b is a vector of dimension n, and X an array of dimension n or larger. When ARCESL is called, the equations Ax = b are solved and the solution stored in X. The vector b is destroyed by the routine.

Programming. ARCESL calls the subroutines ARCEFS, ARCEFM, ARCEFE, ARCEBS, ARCEBM, and ARCEBE. These routines were developed by J. C. Diaz (Mobil Research and Development Corp., Farmers Branch, Texas), G. Fairweather (University of Kentucky), and P. Keast (University of Toronto).

SOLUTION OF ALMOST BLOCK TRIDIAGONAL SYSTEMS OF LINEAR EQUATIONS

Consider a system Tx = b of linear equations where T is a square matrix having the block structure

$$T = \begin{pmatrix} A_1 & B_1 & C_1 & & & \\ C_2 & A_2 & B_2 & & 0 & \\ & C_3 & A_3 & B_3 & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & & & \\ & & C_{n-1} & A_{n-1} & B_{n-1} & \\ & & B_n & C_n & A_n & \end{pmatrix}$$

Here it is assumed that A_k, B_k, C_k $(k=1,\ldots,n)$ are $m \times m$ matrices, and that b is a column vector of dimension mn. If $n \geqslant 4$ then the following subroutine is available for solving Tx = b.

<u>CALL BTSLV(MO,m,n,A,B,C,X,IP,IERR)</u>

MO is an input argument which specifies if BTSLV is being called for the first time, or if it is being recalled to solve another set of equations Tx = b where T is the same coefficient matrix but b has been modified. On an initial call to the routine, MO = 0 and we have the following setup:

A, B, C are 3-dimensional arrays of dimension $m \times m \times n$ where the (i,j)-th elements of the matrices A_k, B_k, C_k are stored in A(i,j,k), B(i,j,k), C(i,j,k) for k = 1,...,n. A, B, C are modified by the routine.

X is an array of dimension mn or larger. On input, the vector b is stored in X. When BTSLV is called, if a solution x is obtained for Tx = b then the solution is stored in X.

IP is an array of dimension mn or larger that is used by the routine for listing the row interchanges that are made.

On an initial call to the routine, a block LU decomposition is performed on T, the results of which are stored in A, B, C. This decomposition involves row interchanges only within the diagonal blocks A_k; i.e., no row interchanges are performed between rows of different blocks A_k and A_ℓ. Thus it may occur that the decomposition of a nonsingular matrix T cannot be completed. IERR is a variable that reports the status of the results. When BTSLV terminates, then IERR will have one of the following values:

IERR = 0	T was successfully decomposed and the equations Tx = b solved.
IERR = −1	(Input Error) Either $m < 1$ or $n < 4$.
IERR = k	The decomposition process failed in the kth diagonal block. The routine cannot solve the equations in their present form.

After an initial call to BTSLV, if IERR = 0 then the routine may be recalled with MO $\neq$ 0 and a new b. When MO $\neq$ 0, then it is assumed that A, B, C, and IP have not been modified and that X contains the new b. The routine retrieves from A, B, C, and IP the block decomposition that was obtained on the initial call to BTSLV, and solves the new system of equations Tx = b. The solution is stored in X. In this case, IERR is not referenced by the routine.

Programming. BTSLV employs the subroutines DECBT, SOLBT, DEC, and SOL. These subroutines were written by Alan C. Hindmarsh (Lawrence Livermore Laboratory).

Reference. Hindmarsh, A. C., ***Solution of Block-Tridiagonal Systems of Linear Algebraic Equations***, Report UCID-30150, Lawrence Livermore Laboratory, 1977.

INVERTING SYMMETRIC REAL MATRICES AND SOLVING SYMMETRIC SYSTEMS OF REAL LINEAR EQUATIONS

The subroutines SMSLV and DSMSLV are available for inverting symmetric real matrices A and solving systems of real linear equations AX = B. SMSLV handles single precision problems and DSMSLV handles double precision problems. It is assumed that the matrix A is in packed form. If the inverse of A is computed, then the inverse is a symmetric matrix which will be stored in packed form.

Note. All eigenvalues of a real symmetric matrix A are real. The ***inertia*** of A is the ordered triple (π, ν, ζ) where π is the number of positive eigenvalues of A, ν the number of negative eigenvalues of A, and ζ the number of zero eigenvalues of A. Thus, if n is the order of A then $\pi + \nu + \zeta = n$. Also A is positive definite (positive semi-definite, negative definite, negative semi-definite) if $\pi = n$ ($\nu = 0$, $\nu = n$, $\pi = 0$).

CALL SMSLV (MO,n,m,A,B,kb,D,RCOND,INERT,IERR,IPVT,WK)
CALL DSMSLV (MO,n,m,A,B,kb,D,RCOND,INERT,IERR,IPVT,WK)

A is an array of dimension $n(n+1)/2$ containing the elements of a packed $n \times n$ symmetric matrix where $n \geqslant 1$. If MO = 0 then the inverse of A is computed and stored in A in packed form. If MO $\neq$ 0 then the inverse of A is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a matrix having n rows and m columns. In this case AX = B is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case B is ignored.

The argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then kb is ignored.

D is an array of dimension 2. When either routine is called the determinant det(A) of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leqslant |d| < 10$ and k an integer, then d is stored in D(1) and the exponent k is stored in floating point form in D(2).

RCOND is a variable. When either routine is called, the routine makes an estimate c of the condition number of the matrix A (relative to the L_1 norm). RCOND is assigned the value 1/c.

INERT is an integer array of dimension 3. When either routine is called the inertia of the matrix A is computed. INERT(1) is set to the number of positive eigenvalues of A, INERT(2) is set to the number of negative eigenvalues, and INERT(3) is set to the number of zero eigenvalues.

IPVT is an integer array of dimension n or larger that is used by the routines for keeping track of the row and column interchanges that are made. WK is an array of dimension n or larger that is used as a work space.

Remarks

(1) If SMSLV is called then it is assumed that A and B are single precision real matrices, D and WK are single precision real arrays, and RCOND is a single precision real variable. Otherwise, if DSMSLV is called then it is assumed that A and B are double precision real matrices, D and WK are double precision real arrays, and RCOND is a double precision real variable.

(2) RCOND satisifies $0 \leqslant \text{RCOND} \leqslant 1$. If $\text{RCOND} \approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if SMSLV is used to invert a matrix in the 14 digit CDC single precision arithemetic and RCOND = .4E−3, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If $\text{RCOND} \approx 1$ then one should expect the results to be almost as accurate as the original data A. However, if $\text{RCOND} \approx 0$ then one should expect the results to be nonsense.

(3) The data in A is always destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that $1 + \text{RCOND} > 1$, then IERR is set to 0 and the problem is solved. Otherwise, if $1 + \text{RCOND} = 1$ then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant and inertia will not have been computed.

Algorithm. The diagonal pivoting factorization procedure is used. Partial pivoting is employed.

Precision. SMSLV and the more general routine MSLV have approximately the same accuracy, and DSMSLV and DMSLV have approximately the same accuracy.

Efficiency. Even though SMSLV performs approximately half the number of multiplications and divisions as MSLV, normally one can expect SMSLV to take about 70-80% of the time required by MSLV. However, for sparse matrices SMSLV may be 20-30% slower than MSLV. Similar comments hold for DSMSLV and DMSLV.

Programming. SMSLV and DSMSLV are driver routines for the LINPACK subroutines SSPCO, SSPFA, SSPSL, SSPDI and DSPCO, DSPFA, DSPSL, DSPDI. SSPCO and DSPCO were written by Cleve Moler (University of New Mexico). The remaining LINPACK subroutines were written by James Bunch (University of California, San Diego). The subroutines employ the vector routines SCOPY, SSWAP, SDOT, SSCAL, SAXPY, SASUM, ISAMAX and DSWAP, DDOT, DSCAL, DAXPY, DASUM, IDAMAX.

References

(1) Bunch, J. R. and Parlett, B. N., "Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations," ***SIAM J. Numerical Analysis*** *8* (1971), pp. 639-655.

(2) Bunch, J. R., "Analysis of the Diagonal Pivoting Method," ***SIAM J. Numerical Analysis*** *8* (1971), pp. 656-680.

(3) Bunch, J. R., Kaufman, L., and Parlett, B. N., "Decomposition of a Symmetric Matrix," ***Numerische Mathematik*** *27* (1976), pp. 95-109.

(4) Bunch, J., and Kaufman, L., "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems," ***Math. Comp.*** *31* (1977), pp. 163-179.

(5) Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Numerical Analysis*** *16* (1979), pp. 368-375.

(6) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

INVERTING POSITIVE DEFINITE SYMMETRIC MATRICES AND SOLVING POSITIVE DEFINITE SYMMETRIC SYSTEMS OF LINEAR EQUATIONS

The subroutines PCHOL and DPCHOL are available for inverting positive definite symmetric real matrices A and solving systems of real linear equations AX = B. PCHOL handles single precision problems and DPCHOL handles double precision problems. It is assumed that the matrix A is in packed form. If the inverse of A is computed then the inverse is a symmetric matrix which will be stored in packed form.

```
CALL PCHOL(MO,n,m,A,B,kb,IERR)
CALL DPCHOL(MO,n,m,A,B,kb,IERR)
```

A is an array of dimension $n(n + 1)/2$ or larger containing the elements of a packed $n \times n$ positive definite symmetric matrix where $n \geqslant 1$. If MO = 0 then the inverse of A is computed and stored in A in packed form. If MO $\neq$ 0 then the inverse is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a matrix having n rows and m columns. In this case AX = B is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case B is ignored.

The argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then kb is ignored.

Remarks

(1) If PCHOL is called then it is assumed that A and B are single precision real arrays, and if DPCHOL is called then it is assumed that A and B are double precision arrays.
(2) The data in A is destroyed.

Error Return. IERR is an integer variable. If A is positive definite then IERR is set to 0 and the problem is solved. Otherwise, IERR = k if the leading $k \times k$ submatrix of A is not positive definite.

Algorithm. The Cholesky procedure is used.

Precision. The results obtained by PCHOL and DPCHOL are occasionally less accurate (up to 1 significant digit) than the results obtained by SMSLV and DSMSLV.

Programming. PCHOL and DPCHOL are driver routines for the LINPACK subroutines SPPFA, SPPSL, SPPDI and DPPFA, DPPSL, DPPDI. These subroutines were written by

Cleve Moler (University of New Mexico). The functions SDOT, DDOT and subroutines SAXPY, SSCAL, DAXPY, DSCAL are also used.

Reference. Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

INVERTING GENERAL COMPLEX MATRICES AND SOLVING GENERAL SYSTEMS OF COMPLEX LINEAR EQUATIONS

The following subroutine is available for inverting arbitrary complex matrices A and solving arbitrary systems of complex linear equations AX = B.

CALL CMSLV(MO,n,m,A,ka,B,kb,D,RCOND,IERR,IPVT,WK)

A is a complex matrix of order n where $n \geqslant 1$. If MO = 0 then the inverse of A is computed and stored in A. If MO $\neq$ 0 then the inverse is not computed.

The argument m is an integer. If $m \geqslant 1$ then B is a complex matrix having n rows and m columns. In this case the matrix equation AX = B is solved and the solution X is stored in B. If $m \leqslant 0$ then there are no equations to be solved. In this case the argument B is ignored.

The argument ka is the number of rows in the dimension statement for A in the calling program, and the argument kb is the number of rows in the dimension statement for B in the calling program. If $m \leqslant 0$ then the argument kb is ignored.

D is a complex array of dimension 2. When CMSLV is called the determinant det(A) of the matrix A is computed. If $\det(A) = d \cdot 10^k$ where $1 \leqslant |\mathrm{Re}(d)| + |\mathrm{Im}(d)| < 10$ and k an integer, then d is stored in D(1) and the exponent k is stored as a complex number in D(2).

RCOND is a single precision real variable. When CMSLV is called, the routine makes an estimate c of the condition number of the matrix A (relative to the modified L_1 norm where each absolute value $|z|$ is replaced with $|\mathrm{Re}(z)| + |\mathrm{Im}(z)|$). RCOND is assigned the value 1/c.

IPVT is an integer array of dimension n or larger that is used by the routine for keeping track of the row interchanges that are made. WK is a complex array of dimension n or larger that is used as a work space.

Remarks

(1) RCOND satisfies $0 \leqslant \mathrm{RCOND} \leqslant 1$. If $\mathrm{RCOND} \approx 10^{-k}$ then one can expect the results to have approximately k fewer significant digits of accuracy than the elements of A. For example, if CMSLV is used to invert a matrix in the 14 digit CDC single precision arithmetic and RCOND = .4E-3, then the computed coefficients of the inverse matrix should normally be accurate to about 11 digits. In general, RCOND characterizes how well or poorly conditioned the problem is. If $\mathrm{RCOND} \approx 1$ then one should expect the results to be almost as accurate as the original data A. However, if $\mathrm{RCOND} \approx 0$ then one should expect the results to be nonsense.

(2) The matrix A is always destroyed.

Error Return. IERR is an integer variable. If RCOND is sufficiently large so that $1 + \text{RCOND} > 1$, then IERR is set to 0 and the problem is solved. Otherwise, if $1 + \text{RCOND} = 1$ then IERR is set to 1 and the routine terminates. In this case, A will have been destroyed but B will not have been modified. Also the determinant will not have been computed.

Algorithm. The partial pivot Gauss elimination procedure is used. The pivots a_{kj} are selected so that $|\text{Re}(a_{kj})| + |\text{Im}(a_{kj})| = \max \{|\text{Re}(a_{ij})| + |\text{Im}(a_{ij})|: i = j,\dots,n\}$ rather than $|a_{kj}| = \max \{|a_{ij}|: i = j,\dots,n\}$.

Programming. CMSLV is a driver routine for the LINPACK subroutines CGECO, CGEFA, CGESL, CGEDI. The subroutines were coded by Cleve Moler (University of New Mexico). The subroutines employ the vector routines CSWAP, CDOTC, CSCAL, CSSCAL, CAXPY, SCASUM, ICAMAX.

References

(1) Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

(2) Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., "An Estimate for the Condition Number of a Matrix," ***SIAM Journal of Numerical Analysis 16*** (1979), pp. 368-375.

SOLUTION OF COMPLEX EQUATIONS WITH ITERATIVE IMPROVEMENT

Given a complex $n \times n$ matrix A and a complex column vector b. The following routine is available for solving the equation $Ax = b$. Iterative improvement is performed to compute the solution x to machine accuracy.

<u>CALL CSLVMP(MO,n,A,ka,b,X,WK,IWK,IERR)</u>

MO is an input argument which specifies if CSLVMP is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional complex array of dimension ka $\times$ n containing the matrix A, b a complex vector of dimension n, and X a complex array of dimension n. When CSLVMP is called, $Ax = b$ is solved and the solution stored in X. A and b are not modified by the routine.

WK is a complex array of dimension $n^2 + n$ or larger, and IWK an integer array of dimension n or larger. These arrays are for internal use by the routine. On an initial call to CSLVMP, an LU decomposition is obtained for A and stored in WK and IWK. Then the equation $Ax = b$ is solved.

IERR is an integer variable that reports the status of the results. On an initial call to CSLVMP, when the routine terminates IERR will have one of the following values:

IERR = 0	The solution X was obtained to machine accuracy.
IERR = 1	X was obtained, but not to machine accuracy.
IERR = –k	The k^{th} column of A was reduced to a column containing only zeros.

When IERR = –k, no solution is obtained.

After an initial call to CSLVMP, if IERR = 0 or 1 on output, then the routine may be called to solve a new set of equations $Ax = \tilde{b}$ without having to redecompose the matrix A. In this case, the input argument MO may be set to any nonzero value. When MO $\neq$ 0 it is assumed that only b has been modified. The routine employs the LU decomposition obtained on the initial call to CSLVMP to solve the new set of equations $Ax = b$. On output X will contain the solution to the new set of equations. As before, A and b are not modified by the routine.

If CSLVMP is recalled with MO $\neq$ 0, then when the routine terminates IERR will have one of the following values:

IERR = 0	The solution X was obtained to machine accuracy.
IERR = 1	X was obtained, but not to machine accuracy.

Programming. CSLVMP calls the subroutine CLUIMP. These routines were written by A. H. Morris. The subroutines CMCOPY, CGEFA, CGESL, CSCAL, CAXPY and functions SPMPAR, CDOTC, ICAMAX are also employed.

SINGULAR VALUE DECOMPOSITION OF A MATRIX

If A is a complex m $\times$ n matrix then there exists an m $\times$ m unitary matrix U and an n $\times$ n unitary matrix V such that D = U*AV is a diagonal matrix[1]. Let $d_1,\ldots,d_k$ be the diagonal elements of D where k = min $\{m,n\}$. Then U and V can be selected so that the diagonal elements are real numbers and $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_k \geqslant 0$. The nonnegative diagonal elements d_i are unique. However, a variety of selections can be made for U and V. In particular, if A is a real matrix then U and V can be chosen to be real orthogonal matrices. The decomposition D = U*AV is called the ***singular value decomposition of A.*** The elements $d_1,\ldots,d_k$ are the ***singular values*** of A, the columns of U are ***left singular vectors***, and the columns of V are ***right singular vectors.***

Remark. For m > n, $D = \begin{pmatrix} D_1 \\ 0 \end{pmatrix}$ where $D_1 = \text{diag}(d_1,\ldots,d_n)$. Consequently, if U is partitioned into $U = (U_1\ U_2)$ where U_1 has n columns, then it follows that $A = UDV^* = U_1 D_1 V^*$. The decomposition $A = U_1 D_1 V^*$ is frequently also called the singular value decomposition, and in many applications it suffices.

The following subroutines are available for finding the singular value decomposition D = U*AV of a matrix A.

CALL SSVDC(A,ka,m,n,D,E,U,ku,V,kv,WORK,JOB,INFO)
CALL DSVDC(A,ka,m,n,D,E,U,ku,V,kv,WORK,JOB,INFO)
CALL CSVDC(A,ka,m,n,D,E,U,ku,V,kv,WORK,JOB,INFO)

A is a 2-dimensional array of dimension ka $\times$ n containing the m $\times$ n matrix whose singular value decomposition is to be computed. D is an array of dimension min $\{m + 1, n\}$. When any of the routines is called, the singular values of A are computed and stored in descending order of magnitude in D(1),...,D(k) where k = min $\{m, n\}$.

JOB is an integer that controls the computation of the singular vectors. It is assumed that JOB = I·10 + J when I, J = 0, 1,...,9. I and J have the following meaning.

I = 0	Do not compute the left singular vectors.
I = 1	Compute all m left singular vectors.
I > 1	Compute the first min $\{m, n\}$ left singular vectors. (Here we compute the decomposition $A = U_1 D_1 V^*$.)
J = 0	Do not compute the right singular vectors.
J > 0	Compute the right singular vectors.

U is a 2-dimensional array which contains the left singular vectors that are requested, and ku is the number of rows in the dimension statement for U in the calling program. It is

[1] U* denotes the adjoint matrix of U.

assumed that ku $\geqslant$ m. If no left singular vectors are requested (i.e., if JOB $<$ 10) then U is ignored by the routines. Otherwise, U must be of dimension ku $\times$ m if all m left singular vectors are requested, and U must be of dimension ku $\times$ min$\{m, n\}$ if the first min$\{m, n\}$ left singular vectors are requested.

V is a 2-dimensional array which contains the right singular vectors that are requested, and kv is the number of rows in the dimension statement for V in the calling program. It is assumed that kv $\geqslant$ n. If no right singular vectors are requested then V is ignored by the routines. Otherwise, V must be of dimension kv $\times$ n if the right singular vectors are requested.

E is an array of dimension n or larger, and WORK is an array of dimension m or larger. E and WORK are storage areas for the routines.

Remarks

(1) If SSVDC is called then it is assumed that the arrays A,D,E,U,V,WORK are single precision real arrays. If DSVDC is called then it is assumed that the arrays are double precision real arrays, and if CSVDC is called then it is assumed that the arrays are complex arrays.

(2) The contents of A are destroyed by the routines. If left singular vectors are requested and there is sufficient storage in A to hold the vectors (there will be sufficient storage if m $\leqslant$ n or JOB $\geqslant$ 20), then one may set U = A. Similarly, if right singular vectors are requested and m $\geqslant$ n then one may set V = A. However, only one of the two arrays U and V may be identified with A.

Error Return. INFO is an integer variable. If all the singular values are found then INFO will be set to 0 and the array E will contain zeros. However, if the j^{th} singular value cannot be found then INFO is set to j. In this case, if $j < k$ where $k = \min\{m, n\}$ then the singular values $d_{j+1},\ldots,d_k$ will have been computed and stored in D. A will have been reduced to an upper bidiagonal matrix B with D as its diagonal and E its super diagonal. If U and V have been requested then $B = U^*AV$ will be satisfied.

Programming. SSVDC, DSVDC, and CSVDC are part of the LINPACK package of matrix subroutines released by Argonne National Laboratory. The routines were coded by G. W. Stewart (University of Maryland). The routines employ the vector subroutines SSWAP, SROT, SDOT, SSCAL, SAXPY, SNRM2, DSWAP, DROT, DDOT, DSCAL, DAXPY, DNRM2, and CSWAP, CSROT, CDOTC, CSCAL, CAXPY, SCNRM2. Also the subroutines SROTG and DROTG are called.

Reference. Dongarra, J. J., Bunch, J.R., Moler, C.B., and Stewart, G. W., ***LINPACK Users' Guide***, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

EVALUATION OF THE CHARACTERISTIC POLYNOMIAL OF A MATRIX

The following functions are available for computing the determinant of A − xI where A is an n × n matrix, x a number, and I the n × n identity matrix.

DET(A,ka,n,x)
DPDET(A,ka,n,x)
CDET(A,ka,n,x)

DET is a real function that is used when A is a single precision real matrix and x is a single precision real number, DPDET is a double precision function that is used when A is a double precision real matrix and x is a double precision real number, and CDET is a complex function that is used when A is a complex matrix and x is a complex number.

The value of the appropriate function is the determinant of the matrix A − xI. The argument ka has the value:

ka = the numoer of rows in the dimension statement for A in the calling program

It is assumed that ka $\geqslant$ n $\geqslant$ 1.

Note. A is destroyed during computation.

Algorithm. Gauss partial pivoting is performed to reduce A − xI to upper triangular form. In CDET the pivots a_{kj} are selected so that $|\mathrm{Re}(a_{kj})| + |\mathrm{Im}(a_{kj})| = \mathrm{Max}\{|\mathrm{Re}(a_{ij})| + |\mathrm{Im}(a_{ij})| : i = j,\ldots,n\}$ rather than $|a_{kj}| = \max\{|a_{ij}| : i = j,\ldots,n\}$.

Programmer. A. H. Morris

SOLUTION OF THE MATRIX EQUATION AX + XB = C

Given an m $\times$ m matrix A, n $\times$ n matrix B, and m $\times$ n matrix C. The subroutines ABSLV and DABSLV are available for obtaining the m $\times$ n matrix X which solves the equation AX + XB = C. ABSLV yields single precision results and DABSLV yields double precision results.

CALL ABSLV(MO,m,n,A,ka,B,kb,C,kc,WK,IERR)
CALL DABSLV(MO,m,n,A,ka,B,kb,C,kc,WK,IERR)

If ABSLV is called then it is assumed that A, B, C, and WK are single precision real arrays. Otherwise, if DABSLV is called then it is assumed that A, B, C, and WK are double precision arrays.

It is assumed that $m \geqslant 1$ and $n \geqslant 1$. The input arguments ka, kb, kc have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program
kc = the number of rows in the dimension statement for C in the calling program

It is required that $ka \geqslant m$, $kb \geqslant n$, $kc \geqslant m$.

WK is an array of dimension $m^2 + n^2 + 2k$ or larger where $k = \max\{m,n\}$. WK is a general storage area for the routine.

MO is an input argument which specifies if the routine is being called for the first time. On an initial call MO = 0. In this case, A is reduced to lower real Schur form, B is reduced to upper real Schur form, and then the transformed system of equations is solved.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0	The solution was obtained and stored in C.
IERR = 1	The equations are inconsistent for A and B.
IERR = −1	A could not be reduced to lower Schur form.
IERR = −2	B could not be reduced to upper Schur form.

If IERR $\neq$ 0 then no solution is obtained.

When IERR = 0, A contains the lower Schur form of the matrix A, B contains the upper Schur form of the matrix B, and WK contains the orthogonal matrices involved in the decompositions of A and B. This information can be reused to solve a new set of equations. The following options are available:

MO = 1	New matrices A and C are given. The data for B is reused in solving the new set of equations.
MO = 2	New matrices B and C are given. The data for A is reused in solving the new set of equations.
MO ≠ 0, 1, 2	A new matrix C is given. The data for A and B is reused in solving the new set of equations.

When the routine is recalled, it is assumed that m, n, and WK have not been modified.

Programming. ABSLV employs the subroutines ABSLV1, ORTHES, ORTRN1, SCHUR, SHRSLV, SLV, and DABSLV employs the subroutines DABSV1, DORTH, DRTRN1, DSCHUR, DSHSLV, DPSLV. ABSLV and DABSLV are adaptations by A. H. Morris of the subroutine AXPXB written by R. H. Bartels and G. W. Stewart (University of Texas at Austin).

Reference. Bartels, R. H. and Stewart, G. W., "Algorithm 432, Solution of the Matrix Equation AX + XB = C," ***Comm. ACM 15*** (1972), pp. 820 - 826.

SOLUTION OF THE MATRIX EQUATION $A^tX + XA = C$ WHEN C IS SYMMETRIC

Given matrices A and C of order n where C is symmetric. Then the subroutines TASLV and DTASLV are available for obtaining the symmetric matrix X which solves the equation $A^tX + XA = C$. TASLV yields single precision results and DTASLV yields double precision results.

```
CALL TASLV(MO,n,A,ka,C,kc,WK,IERR)
CALL DTASLV(MO,n,A,ka,C,kc,WK,IERR)
```

If TASLV is called then it is assumed that A, C, and WK are single precision real arrays. Otherwise, if DTASLV is called then it is assumed that A, C, and WK are double precision arrays.

It is assumed that $n \geqslant 1$. The input arguments ka and kc have the following values:
ka = the number of rows in the dimension statement for A in the calling program
kc = the number of rows in the dimension statement for C in the calling program
It is required that $ka \geqslant n$ and $kc \geqslant n$.

WK is an array of dimension $n^2 + 2n$ or larger that is a general storage area for the routine.

MO is an input argument which specifies if the routine is being called for the first time. On an initial call MO = 0. In this case, A is reduced to upper real Schur form and then the transformed system of equations is solved.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0	The solution was obtained and stored in C.
IERR = 1	The equations are inconsistent.
IERR = −1	A could not be reduced to upper Schur form.

If IERR $\neq$ 0 then no solution is obtained.

When IERR = 0, A contains the upper Schur form of the matrix A and WK contains the orthogonal matrix involved in the decomposition of A. This data can be reused to solve a new set of equations $A^tX + XA = \tilde{C}$. In this case, MO can be set to any nonzero value. When MO $\neq$ 0 it is assumed that only C has been modified. When the routine terminates, the solution for the new set of equations is stored in C.

Programming. TASLV employs the subroutines TASLV1, ORTHES, ORTRN1, SCHUR, SYMSLV, SLV and DTASLV employs the subroutines DTASV1, DORTH, DRTRN1, DSCHUR, DSYMSV, DPSLV. TASLV and DTASLV are adaptations by A. H. Morris of the subroutine ATXPXA written by R. H. Bartels and G. W. Stewart (University of Texas at Austin).

Reference. Bartels, R. H. and Stewart, G. W., "Algorithm 432, Solution of the Matrix Equation AX + XB = C," *Comm. ACM 15* (1972), pp. 820 - 826.

SOLUTION OF THE MATRIX EQUATION $AX^2 + BX + C = 0$

Given complex $n \times n$ matrices A, B, and C. The following subroutine is available for obtaining a complex $n \times n$ matrix X which solves the equation $AX^2 + BX + C = 0$.

CALL SQUINT(m,n,A,B,C,IND,X,WK,ℓ,τ,MAX,IERR)

It is assumed that A, B, C, and X are 2-dimensional complex arrays of dimension $m \times n$ where $m \geqslant n$. When SQUINT is called, the $n \times n$ complex matrix solution obtained for $AX^2 + BX + C = 0$ is stored in X. A, B, and C are modified by the routine.

IND is an integer variable. On input, if IND $\neq 0$ then it is assumed that an initial approximation for the desired solution is provided in X by the user. Otherwise, if IND = 0 then the routine provides its own initial approximation. Then Newton iteration is performed. On output, IERR = the number of iterations that were performed to compute X.

WK is a complex array of dimension ℓ that is a work space for the routine. It is assumed that $\ell \geqslant 7n^2 + n$. When SQUINT terminates, WK(1) is a complex number whose real part is the norm $\|AX^2 + BX + C\|_\infty$.

The argument τ is a real number. If $\tau \leqslant 0$ then X is computed to machine precision. Otherwise, if $\tau > 0$ then iteration terminates when $\|AX^2 + BX + C\| < \tau$.

MAX is a variable. If MAX > 0, then MAX is the maximum number of iterations that may be performed. If MAX $\leqslant 0$ then it is reset by the routine to 30, the default maximum number of iterations.

Error Return. IERR is an integer variable that is set by the routine. If the desired solution X is obtained, then IERR is assigned the value 0. Otherwise, IERR is set to one of the following values:

IERR = 1	MAX iterations were performed. More iterations are needed.
IERR = 2,3	Factorization of the equations could not be completed. X could not be computed.
IERR = 10+n	Newton iteration failed on iteration n. Possibly too much accuracy was requested. X could not be computed.
IERR = 999	(Input Error) Either $m \geqslant n \geqslant 1$ is not satisfied or $\ell < 7n^2 + n$.

When IERR = 1 occurs, X contains the most recent value obtained for the solution and WK(1) is a complex number whose real part is the latest value obtained for the norm $\|AX^2 + BX + C\|_\infty$.

Programming. SQUINT employs the subroutines SQUIN2, CQZHES, CQZIT, TRISLV, and CTRANS. These routines were designed by George J. Davis (Georgia State University, Atlanta, Georgia). CQZHES and CQZIT are modifications of the EISPACK subroutines QZHES and QZIT, developed at Argonne National Laboratory. The function SPMPAR is also used.

References

(1) Davis, G. J., "Algorithm 598. An Algorithm to Compute Solvents of the Matrix Equation $AX^2 + BX + C = 0$," ***ACM Trans. Math Software 9*** (1983), pp. 246-254.

(2) Garbow, B. S., et al., ***Matrix Eigensystems Routines – EISPACK Guide Extension***, Springer–Verlag, 1977.

EXPONENTIAL OF A REAL MATRIX

Let A be a real matrix of order $n \geqslant 1$. Then the subroutines MEXP and DMEXP are available for computing the exponential matrix $e^A = \sum_{i=0}^{\infty} A^i/i!$. MEXP yields single precision results and DMEXP yields double precision results.

CALL MEXP(A,ka,n,Z,kz,WK,IERR)

A is a real matrix of order $n \geqslant 1$ and Z a real 2-dimensional array. MEXP computes e^A and stores the results in Z. The arguments ka and kz have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kz = the number of rows in the dimension statement for Z in the calling program

It is assumed that $ka \geqslant n$, $kz \geqslant n$, and that A and Z are different storage areas. If $n > 1$ then A is destroyed.

WK is a real array of dimension $n(n + 8)$ or larger that is a work space for the routine.

IERR is a variable that reports the status of the results. When MEXP terminates, IERR is assigned one of the following values:

IERR = 0	The exponential was computed.
IERR = 1	The norm $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ is too large. e^A cannot be computed.
IERR = 2	The Pade denominator matrix is singular. (This should never occur.)

Algorithm. A is balanced, yielding a matrix $B = D^{-1}P^tAPD$ where D is a diagonal matrix, P a permutation matrix, and $\|B\|_1 \leqslant \|A\|_1$. Next m is set to the smallest nonnegative integer such that $\min\{\|B\|_1, \|B\|_\infty\} \leqslant 2^m$, and the 8th diagonal Pade approximation for e^x is used to compute $\exp(B/2^m)$. Then $e^B = [\exp(B/2^m)]^{2^m}$ is obtained by m squarings, and $e^A = PDe^BD^{-1}P^t$ is applied.

Programming. MEXP calls the subroutines BALANC, BALINV, and SLV. The function I1MACH is also used. MEXP was written by A. H. Morris.

Reference. Ward, Robert, C., "Numerical Computation of the Matrix Exponential with Accuracy Estimate," ***SIAM J. Numerical Analysis 14*** (1977), pp. 600 - 610.

CALL DMEXP(A,ka,n,Z,kz,WK,IERR)

A is a double precision matrix of order $n \geqslant 1$ and Z a double precision 2-dimensional array. DMEXP computes e^A and stores the results in Z. The arguments ka and kz have the following values:

ka = the number of rows in the dimension statement for A in the calling program

kz = the number of rows in the dimension statement for Z in the calling program

It is assumed that $ka \geqslant n$, $kz \geqslant n$, and that A and Z are different storage areas. If $n > 1$ then A is destroyed.

WK is a double precision array of dimension $n(n + 12)$ or larger that is a work space for the routine.

IERR is a variable that reports the status of the results. When DMEXP terminates, IERR is assigned one of the following values:

IERR = 0	The exponential was computed.
IERR = 1	The norm $\|A\|_1$ is too large. e^A cannot be computed.
IERR = 2	The Pade denominator matrix is singular. (This should never occur.)

Programming. DMEXP calls the subroutines DBAL, DBALNV, and DPSLV. The function I1MACH is also used. DMEXP was written by A. H. Morris.

Reference. Ward, Robert C., "Numerical Computation of the Matrix Exponential with Accuracy Estimate," ***SIAM J. Numerical Analysis 14*** (1977), pp. 600 - 610.

SOLVING SYSTEMS OF 200-400 LINEAR EQUATIONS

For $n \geqslant 1$, let A denote an $n \times n$ matrix and b a column vector of dimension n. Then the subroutines LE, DPLE, and CLE are available for solving the equations $Ax = b$ where A is not stored in-core. For large n, these routines require a work space of dimension $\approx n^2/4$. This permits the solution of systems of equations of double the order permitted by the standard solution procedures.

CALL LE(ROWK,n,b,X,WK,IWK,IERR)
CALL DPLE(ROWK,n,b,X,WK,IWK,IERR)
CALL CLE(ROWK,n,b,X,WK,IWK,IERR)

X is an array of dimension n and IERR an integer variable. When the equations are solved, then IERR is set to 0 and the solution is stored in X.

ROWK is the name of a user defined subroutine that has the format:
CALL ROWK(n,k,R)
R is an array of dimension n and $k = 1,\dots,n$. When ROWK is called, the kth row of the matrix A is stored in R. ROWK must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension $[n^2/4] + n + 3$ or larger,[1] and IWK is an integer array of dimension $\max\{1, n-1\}$ or larger. WK and IWK are work spaces for the routines.

Error Return. IERR = k when the first k rows of A are found to be linearly dependent.

Remarks

(1) When LE is called then it is assumed that b, X, WK and the array R in ROWK are single precision real arrays. When DPLE is called then it is assumed that these arrays are double precision real arrays, and when CLE is called then it is assumed that the arrays are complex.

(2) When the equations are solved, ROWK is called to attach the first row of A, then the second row, etc. Each row of A is attached only once.

(3) The array b is not modified by the routines.

Example. Consider a system of $n = 300$ real linear equations $Ax = b$ where the rows of A are stored, one row per logical record, in sequence in an unformatted file (say file 4). Then the following code can be used to solve the equations:

[1] Here $[n^2/4]$ denotes the largest integer $\leqslant n^2/4$.

```
REAL B(300), X(300), WK(22803)
INTEGER IWK(300)
EXTERNAL GETROW
DATA N/300/
      ⋮
REWIND 4
CALL LE(GETROW,N,B,X,WK,IWK,IERR)
```

Here GETROW may be defined by:

```
SUBROUTINE GETROW(N,I,R)
REAL R(N)
READ (4) (R(J),J=1,N)
RETURN
END
```

Algorithm. The partial pivot Henderson-Wassyng procedure is used.

Programming. LE, DPLE, and CLE are modified versions (by A. H. Morris) of the subroutine TE, written by A. Wassyng (University of the Witwatersrand, Johannesburg, South Africa).

Reference. Wassyng, A., "Solving Ax = b: A Method with Reduced Storage Requirements," *SIAM J. Numerical Analysis 19* (1982), pp. 197-204.

BAND MATRIX STORAGE

For an $m \times n$ matrix $A = (a_{ij})$, let m_ℓ be the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. Then m_ℓ and m_u are called the ***lower*** and ***upper band widths*** of A, and $m_\ell + m_u + 1$ the ***total band width*** of A. It is clear that $0 \leqslant m_\ell < m$ and $0 \leqslant m_u < n$, and that $a_{ij} \neq 0$ only when $i-m_\ell \leqslant j \leqslant i + m_u$. If the band width $m_\ell + m_u + 1$ is sufficiently small, then it is also clear that a considerable savings in storage can occur by storing only the nonzero diagonals of A. The band storage scheme adopted by the NSWC library is to store A as an $m \times (m_\ell + m_u + 1)$ matrix $B = (b_{ik})$. The columns of B are the nonzero diagonals of A. Specifically, for each nonzero a_{ij}, $b_{ik} = a_{ij}$ where $k = j - i + m_\ell + 1$. The remaining b_{ik}'s are zeros.

Example. Consider the matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & a_{46} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} & a_{57} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & 0 & a_{76} & a_{77} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{87} \end{pmatrix}$$

where $m_\ell = 1$ and $m_u = 2$. Then A will be stored in band form as follows:

$$B = \begin{pmatrix} 0 & a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} & a_{35} \\ a_{43} & a_{44} & a_{45} & a_{46} \\ a_{54} & a_{55} & a_{56} & a_{57} \\ a_{65} & a_{66} & a_{67} & 0 \\ a_{76} & a_{77} & 0 & 0 \\ a_{87} & 0 & 0 & 0 \end{pmatrix}$$

Remark. The first m_ℓ columns of B contain the nonzero diagonals of A below the main diagonal, the $(m_\ell + 1)^{st}$ column of B contains the main diagonal, and the last m_u columns of B contain the nonzero diagonals of A above the main diagonal.

CONVERSION OF BANDED MATRICES TO AND FROM THE STANDARD FORMAT

The following subroutines permit one to convert banded matrices to and from the standard format.

CALL CVBR(A,ka,m,n,m_ℓ,m_u,B,kb)
CALL CVBC(A,ka,m,n,m_ℓ,m_u,B,kb)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

B is a 2-dimensional array of dimension $kb \times n$ where $kb \geqslant m$. CVBR is used if A and B are real arrays, and CVBC is used if A and B are complex arrays. When the routine is called, the matrix A is stored in the array B in the standard format.

Remark. B may begin in the same location as A. If B begins in the same location then it is assumed that kb = ka. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris

CALL CVRB(A,ka,m,n,m_ℓ,m_u,B,kb)
CALL CVCB(A,ka,m,n,m_ℓ,m_u,B,kb)

A is an $m \times n$ matrix stored in the standard format, and m_ℓ and m_u are integers such that $0 \leqslant m_\ell < m$ and $0 \leqslant m_u < n$. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $ka \geqslant m$.

B is a 2-dimensional array of dimension $kb \times \ell$ where $kb \geqslant m$ and $\ell \geqslant m_\ell + m_u + 1$. CVRB is used if A and B are real arrays, and CVCB is used if A and B are complex arrays. When the routine is called, the m_ℓ diagonals of A immediately below the main diagonal, the main diagonal, and the m_u diagonals immediately above the main diagonal are stored in band form in B.

Remarks

(1) Given a matrix $A = (a_{ij})$, then these routines may be used to convert A to band form when the lower and upper bandwidths m_0 and m_u of A are known. If m_ℓ and m_u are not known, then the subroutines CVRB1 and CVCB1 described below can be used to convert A to band form.

(2) B may begin in the same location as A. If B begins in the same location then it is assumed that kb = ka. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris

CALL CVRB1(A,ka,m,n,m_ℓ,m_u,B,kb,ℓ,IERR)

CALL CVCB1(A,ka,m,n,m_ℓ,m_u,B,kb,ℓ,IERR)

A is an m $\times$ n matrix stored in the standard format. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that A is to be stored in band form in B. B is a 2-dimensional array of dimension kb $\times$ ℓ where kb $\geqslant$ m. The argument ℓ is an estimate of the maximum number of diagonals of A that will have to be stored.

CVRB1 is used if A and B are real arrays, and CVCB1 is used if A and B are complex arrays. IERR, m_ℓ, and m_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for B then A is stored in band form in B. Also IERR is assigned the value 0, m_ℓ = the number of diagonals of A below the main diagonal containing nonzero elements, and m_u = the number of diagonals above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value $m_\ell + m_u + 1$. Reset $\ell \geqslant$ IERR.

Remarks. B may begin in the same location as A. If B begins in the same location then it is assumed that kb = ka. In this case, the result B will overwrite the input data A. Otherwise, if B does not begin in the same location as A, then it is assumed that the storage areas A and B do not overlap.

Programming. CVRB1 calls the subroutine CVRB, and CVCB1 calls the subroutine CVCB. These routines were written by A. H. Morris.

CONVERSION OF BANDED MATRICES TO AND FROM SPARSE FORM

The following subroutines permit one to convert banded matrices to and from sparse form.

<u>CALL MCVBS(A,ka,m,n,m_ℓ,m_u,B,IB,JB,NUM,IERR)</u>
<u>CALL CMCVBS(A,ka,m,n,m_ℓ,m_u,B,IB,JB,NUM,IERR)</u>

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m.

It is assumed that A is to be stored in sparse form in the arrays B, IB, JB. NUM is the estimated maximum number of elements that will appear in B and JB. It is assumed that B and JB are of dimension max $\{1,\text{NUM}\}$ and that IB is of dimension m + 1.

MCVBS is used if A and B are real arrays, and CMCVBS is used if A and B are complex arrays. IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for B and JB, then IERR is assigned the value 0 and A is stored in sparse form in B, IB, JB.

Error Return. If there is not sufficient storage in B and JB for the i^{th} row of A, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A will have been stored in B and JB. Also IB(1),...,IB(i) will contain the appropriate row locations.

Programmer. A. H. Morris

<u>CALL MCVSB(A,IA,JA,m,n,B,kb,ℓ,m_ℓ,m_u,IERR)</u>
<u>CALL CMCVSB(A,IA,JA,m,n,B,kb,ℓ,m_ℓ,m_u,IERR)</u>

A is an m $\times$ n sparse matrix stored in the arrays A, IA, JA. It is assumed that A is to be stored in band form in B. B is a 2-dimensional array of dimension kb $\times$ ℓ where kb $\geqslant$ m. The argument ℓ is an estimate of the maximum number of diagonals of A that will have to be stored.

MCVSB is used if A and B are real arrays, and CMCVSB is used if A and B are complex arrays. IERR, m_ℓ, and m_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for B then A is stored in band form in B. Also IERR is assigned the value 0, m_ℓ = the number of diagonals of A below the main diagonal containing nonzero elements, and m_u = the number of diagonals above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for B, then IERR is assigned the value $m_\ell + m_u + 1$. Reset $\ell \geqslant$ IERR.

Programmer. A. H. Morris

TRANSPOSING BANDED MATRICES

The following subroutines are available for transposing banded matrices.

<u>CALL BPOSE(A,ka,m,n,m_ℓ,m_u,B,kb)</u>
<u>CALL CBPOSE(A,ka,m,n,m_ℓ,m_u,B,kb)</u>

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

B is a 2-dimensional array of dimension $kb \times \ell$ where $kb \geqslant n$ and $\ell \geqslant m_\ell + m_u + 1$. BPOSE is used if A and B are real arrays, and CPOSE is used if A and B are complex arrays. When the routine is called, the transpose A^t of A is stored in band form in B.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris

ADDITION OF BANDED MATRICES

Let A and B be m $\times$ n matrices stored in band form. The following subroutines are available for computing the sum C = A + B.

CALL BADD($m,n,A,ka,m_\ell,m_u,B,kb,n_\ell,n_u,C,kc,\ell,\nu_\ell,\nu_u$,IERR)

CALL CBADD($m,n,A,ka,m_\ell,m_u,B,kb,n_\ell,n_u,C,kc,\ell,\nu_\ell,\nu_u$,IERR)

A and B are m $\times$ n matrices stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, m_u the number of diagonals of A above the main diagonal containing nonzero elements, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program.

It is assumed that A + B is to be stored in band form in C. C is a 2-dimensional array of dimension kc $\times$ ℓ where kc$\geqslant$m. The input argument ℓ is an estimate of the maximum number of diagonals of A + B which will have to be stored ($\ell \leqslant \max\{m_\ell,n_\ell\} + \max\{m_u,n_u\} + 1$). BADD is used if A and B are real arrays, and CBADD is used if A and B are complex arrays. IERR, ν_ℓ, and ν_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for C then A + B is computed and stored in band form in C. Also IERR is assigned the value 0, ν_ℓ = the number of diagonals of A + B below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of A + B above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for C, then IERR is assigned the value ν where ν is an estimate of the number of columns needed for C. Reset $\ell \geqslant \nu$.

Remarks. If $m_\ell \geqslant n_\ell$ then C may begin in the same location as A. If C begins in the same location as A, then it is assumed that kc = ka and that the arrays A and B do not overlap. In this case, the result C will overwrite the input data A. Similarly, if $m_\ell \leqslant n_\ell$ then C may begin in the same location as B when kc = kb and A and B do not overlap. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that kc $\geqslant$ m).

Example. If B = –A then ℓ may be assigned any value $\geqslant$ 1. In this case, C will contain only the main diagonal of the zero matrix A + B, and $\nu_\ell = \nu_u = 0$.

Programmer. A. H. Morris

SUBTRACTION OF BANDED MATRICES

Let A and B be m × n matrices stored in band form. The following subroutines are available for computing the difference C = A − B.

CALL BSUBT(m,n,A,ka,m_ℓ,m_u,B,kb,n_ℓ,n_u,C,kc,ℓ,ν_ℓ,ν_u,IERR)

CALL CBSUBT(m,n,A,ka,m_ℓ,m_u,B,kb,n_ℓ,n_u,C,kc,ℓ,ν_ℓ,ν_u,IERR)

A and B are m × n matrices stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, m_u the number of diagonals of A above the main diagonal containing nonzero elements, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program.

It is assumed that A − B is to be stored in band form in C. C is a 2-dimensional array of dimension kc × ℓ where kc ≥ m. The input argument ℓ is an estimate of the maximum number of diagonals of A−B which will have to be stored ($\ell \leq \max\{m_\ell,n_\ell\} + \max\{m_u,n_u\} + 1$). BSUBT is used if A and B are real arrays, and CBSUBT is used if A and B are complex arrays. IERR, ν_ℓ and ν_u are integer variables. When the routine is called, if ℓ specifies sufficient storage for C then A − B is computed and stored in band form in C. Also IERR is assigned the value 0, ν_ℓ = the number of diagonals of A − B below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of A − B above the main diagonal containing nonzero elements.

Error Return. If ℓ does not specify sufficient storage for C, the IERR is assigned the value ν where ν is an estimate of the number of columns needed for C. Reset $\ell \geq \nu$.

Remarks. If $m_\ell \geq n_\ell$ then C may begin in the same location as A. If C begins in the same location as A, then it is assumed that kc = ka and that the arrays A and B do not overlap. In this case, the result C will overwrite the input data A. Similarly, if $m_\ell \leq n_\ell$ then C may begin in the same location as B when kc = kb and A and B do not overlap. Otherwise, if C does not begin in the same location as A or B, then it is assumed that the storage area for C does not overlap with the storage areas for A and B. In this case there is no restriction on kc (other than the customary restriction that kc ≥ m).

Example. If B = A then ℓ may be assigned any value ≥ 1. In this case, C will contain only the main diagonal of the zero matrix A − B, and $\nu_\ell = \nu_u = 0$.

Programmer. A. H. Morris

MULTIPLICATION OF BANDED MATRICES

Let A and B be matrices stored in band form. The following subroutines are available for computing the product C = AB.

CALL BPROD(m,n,ℓ,A,ka,m_ℓ,m_u,B,kb,n_ℓ,n_u,C,kc,nc,ν_ℓ,ν_u,IERR)

CALL CBPROD(m,n,ℓ,A,ka,m_ℓ,m_u,B,kb,n_ℓ,n_u,C,kc,nc,ν_ℓ,ν_u,IERR)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals of A below the main diagonal containing nonzero elements, and m_u the number of diagonals of A above the main diagonal containing nonzero elements. B is an $n \times \ell$ matrix stored in band form, n_ℓ the number of diagonals of B below the main diagonal containing nonzero elements, and n_u the number of diagonals of B above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program, and kb the number of rows in the dimension statement for B in the calling program. It is assumed that ka $\geqslant$ m and kb $\geqslant$ n.

It is assumed that AB is to be stored in band form in C. C is a 2-dimensional array of dimension kc $\times$ nc where kc $\geqslant$ m. The input argument nc is an estimate of the maximum number of diagonals of AB which will have to be stored (nc $\leqslant \min\{n-1, m_\ell + n_\ell\} + \min\{\ell - 1, m_u + n_u\} + 1$). BPROD is used if A, B, and C are real arrays, and CBPROD is used if A, B, and C are complex arrays. IERR, ν_ℓ, and ν_u are integer variables. When the routine is called, if nc specifies sufficient storage for C then AB is computed and stored in band form in C. Also, IERR is assigned the value 0, ν_ℓ = the number of diagonals of AB below the main diagonal containing nonzero elements, and ν_u = the number of diagonals of AB above the main diagonal containing nonzero elements.

Error Return. If nc does not specify sufficient storage for C, then IERR is assigned the value ν where ν is an estimate of the number of columns needed for C. Reset nc $\geqslant \nu$.

Note. It is assumed that the storage area for C does not overlap with the storage areas for A and B.

Programmer. A. H. Morris

PRODUCT OF A REAL BANDED MATRIX AND VECTOR

Let A be a real m $\times$ n matrix stored in band form. Then the following subroutines are available for multiplying A with a real vector.

CALL BVPRD(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m.

The argument x is a column vector of dimension n and y an array of dimension m. When BVPRD is called, the product Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL BVPRD1(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m.

The arguments x and y are column vectors of dimension n and m respectively. When BVPRD1 is called, Ax + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL BTPRD(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m.

The argument x is a row vector of dimension m and y an array of dimension n. When BTPRD is called, the product xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL BTPRD1$(m,n,A,ka,m_\ell,m_u,x,y)$

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m.

The arguments x and y are row vectors of dimension m and n respectively. When BTPRD1 is called, xA + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

PRODUCT OF A COMPLEX BANDED MATRIX AND VECTOR

Let A be a complex m × n matrix stored in band form. Then the following subroutines are available for multiplying A with a complex vector.

CALL CBVPD(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m × n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are complex arrays. When CBVPD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL CBVPD1(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m × n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

The arguments x and y are column vectors of dimension n and m respectively. A, x, y are complex arrays. When CBVPD1 is called, Ax + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL CBTPD(m,n,A,ka,m_ℓ,m_u,x,y)

A is an m × n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are complex arrays. When CBTPD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL CBTPD1(m,n,A,ka,m_ℓ,m_u,x,y)

A is an $m \times n$ matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and $ka \geqslant m$.

The arguments x and y are row vectors of dimension m and n respectively. A, x, y are complex arrays. When CBTPD1 is called, $xA + y$ is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

SOLUTION OF BANDED SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular n $\times$ n real matrix stored in band form and b a real column vector of dimension n. The subroutine BSLV is available for solving the system of equations $Ax = b$, and the subroutine BSLV1 is available for solving the transposed system of equations $A^t x = b$. On an initial call to either routine, partial pivot Gauss elimination is first employed to obtain an LU decomposition of A, and then the equations are solved. BSLV and BSLV1 always generate the same LU decomposition of A. After the decomposition is obtained on an initial call to BSLV or BSLV1, either routine may be called to solve a new system of equations $Ax = r$ or $A^t x = r$ without having to redecompose the matrix A.

CALL BSLV(MO,A,ka,n,m_ℓ,m_u,X,IWK,IERR)
CALL BSLV1(MO,A,ka,n,m_ℓ,m_u,X,IWK,IERR)

BSLV is called for solving $Ax = b$ and BSLV1 is called for solving $A^t x = b$. The argument m_ℓ is the number of diagonals below the main diagonal of A containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. It is assumed that $n \geqslant 1$, $0 \leqslant m_\ell < n$, and $0 \leqslant m_u < n$. MO is an input argument which specifies if BSLV or BSLV1 is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional array of dimension ka $\times$ m where $ka \geqslant n$ and $m \geqslant 2m_\ell + m_u + 1$. On input, the first $m_\ell + m_u + 1$ columns of the array contain the matrix A in band form. When the routine terminates, the array A will contain the upper triangular matrix U of the LU decomposition and the multipliers which were used to obtain it.

X is an array of dimension n or larger. On input, X contains the vector b. On output, X will contain the solution of the system of equations.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the LU decomposition are stored in IWK.

On an initial call to BSLV or BSLV1, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The system of equations was solved.
IERR = −1	Either $n \leqslant 0$ or $ka < n$.
IERR = −2	Either $m_\ell < 0$ or $m_\ell \geqslant n$.
IERR = −3	Either $m_u < 0$ or $m_u \geqslant n$.
IERR = k	Column k of A has been reduced to a column containing only zeros.

After an initial call to BSLV or BSLV1, if IERR = 0 on output then either routine may be called with MO $\neq$ 0. When MO $\neq$ 0 it is assumed that only b may have been modified. BSLV is called for solving the new set of equations Ax = b, and BSLV1 is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to BSLV or BSLV1 to solve the new set of equations. On input, X contains the new vector b. On output, X will contain the solution to the new set of equations. In this case, IERR is not referenced by the routine.

Remark. Since the array A must contain at least $2m_\ell + m_u + 1$ columns in order that it can hold the upper triangular matrix U and associated multipliers, if $m_\ell \neq m_u$ then it is clear that these routines will be more efficient when $m_\ell < m_u$.

Programming. BSLV and BSLV1 employ the subroutines SNBFA, SNBSL, SAXPY, SSCAL, SSWAP and the functions ISAMAX and SDOT. SNBFA and SNBSL were written by E. A. Voorhees (Los Alamos Scientific Laboratory) and modified by A. H. Morris. SNBFA and SNBSL are distributed by the SLATEC library.

SOLUTION OF BANDED SYSTEMS OF COMPLEX LINEAR EQUATIONS

Let A be a nonsingular $n \times n$ complex matrix stored in band form and b a complex column vector of dimension n. The subroutine CBSLV is available for solving the system of equations $Ax = b$, and the subroutine CBSLV1 is available for solving the transposed system of equations $A^t x = b$. On an initial call to either routine, partial pivot Gauss elimination is first employed to obtain an LU decomposition of A, and then the equations are solved. CBSLV and CBSLV1 always generate the same LU decomposition of A. After the decomposition is obtained on an initial call to CBSLV or CBSLV1, either routine may be called to solve a new system of equations $Ax = r$ or $A^t x = r$ without having to redecompose the matrix A.

CALL CBSLV(MO,A,ka,n,m_ℓ,m_u,X,IWK,IERR)

CALL CBSLV1(MO,A,ka,n,m_ℓ,m_u,X,IWK,IERR)

CBSLV is called for solving $Ax = b$ and CBSLV1 is called for solving $A^t x = b$. The argument m_ℓ is the number of diagonals below the main diagonal of A containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. It is assumed that $n \geqslant 1$, $0 \leqslant m_\ell < n$, and $0 \leqslant m_u < n$. MO is an input argument which specifies if CBSLV or CBSLV1 is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A is a 2-dimensional complex array of dimension ka $\times$ m where $ka \geqslant n$ and $m \geqslant 2m_\ell + m_u + 1$. On input, the first $m_\ell + m_u + 1$ columns of the array contain the matrix A in band form. When the routine terminates, the array A will contain the upper triangular matrix U of the LU decomposition and the multipliers which were used to obtain it.

X is a complex array of dimension n or larger. On input, X contains the vector b. On output, X will contain the solution of the system of equations.

IWK is an array of dimension n or larger for internal use by the routine. The pivot indices involved in the LU decomposition are stored in IWK.

On an initial call to CBSLV or CBSLV1, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The system of equations was solved.
IERR = −1	Either $n \leqslant 0$ or $ka < n$.
IERR = −2	Either $m_\ell < 0$ or $m_\ell \geqslant n$.
IERR = −3	Either $m_u < 0$ or $m_u \geqslant n$.
IERR = k	Column k of A has been reduced to a column containing only zeros.

After an initial call to CBSLV or CBSLV1, if IERR = 0 on output then either routine may be called with MO ≠ 0. When MO ≠ 0 it is assumed that only b may have been modified. CBSLV is called for solving the new set of equations $Ax = b$, and CBSLV1 is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to CBSLV or CBSLV1 to solve the new set of equations. On input, X contains the new vector b. On output, X will contain the solution to the new set of equations. In this case, IERR is not referenced by the routine.

Remark. Since the array A must contain at least $2m_\ell + m_u + 1$ columns in order that it can hold the upper triangular matrix U and associated multipliers, if $m_\ell \neq m_u$ then it is clear that these routines will be more efficient when $m_\ell < m_u$.

Programming. CBSLV and CBSLV1 employ the subroutines CBFA, CBSL, CAXPY, CSCAL, CSWAP and the functions ICAMAX and CDOTU. CBFA and CBSL are adaptations by A. H. Morris of the subroutines SNBFA and SNBSL, written by E. A. Voorhees (Los Alamos Scientific Laboratory). SNBFA and SNBSL are distributed by the SLATEC library.

STORAGE OF SPARSE MATRICES

A matrix is said to be ***sparse*** if it contains sufficiently many zero elements for it to be worthwhile to use special techniques that avoid storing and operating with the zeros. The scheme adopted by the NSWC library for storing a sparse $m \times n$ matrix (a_{ij}) requires three 1-dimensional arrays A, IA, JA. The array A contains the non-zero elements of the matrix, stored row by row. The array JA contains the column numbers of the corresponding elements of the A array; i.e, if A(k) contains a_{ij} then JA(k) = j. The elements of a row of the matrix may be given in any order in A.

IA is an array containing m + 1 integers which specify where the rows of the matrix are stored in A. For $i \leqslant m$, IA(i) is the index of the location in A where the ith row information begins. It is assumed that the rows are stored sequentially; i.e., that $IA(1) \leqslant \cdots \leqslant IA(m)$. IA(m + 1) is set so that IA(m + 1) − IA(1) = the number of elements stored in A. For $i \leqslant m$, if IA(i) < IA(i + 1) then A(IA(i)) is the first entry of the ith row of the matrix in A. Otherwise, if IA(i) = IA(i + 1) then no entries for the ith row of the matrix are stored in A. This can occur only if the ith row of the matrix consists entirely of zeros. If this occurs then the ith row is called a ***null row*** of A. For any $i \leqslant m$, IA(i + 1) - IA(i) is the number of entries for the ith row of the matrix that are stored in A. For convenience, IA(i + 1) − IA(i) is called the ***length*** of the ith row.

Example. The matrix

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 & a_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{37} & a_{38} \\ 0 & 0 & a_{43} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

can be stored as follows:

A:	a_{11}	a_{18}	a_{12}	a_{37}	a_{38}	a_{43}
JA:	1	8	2	7	8	3
IA:	1	4	4	6	7	

The storage of the elements a_{11} a_{12} a_{18} in the order a_{11} a_{18} a_{12} is perfectly satisfactory. The elements of a row of the matrix may be given in any order desired.

Note. It is not required that each a_{ij} in A be non-zero.

CONVERSION OF SPARSE MATRICES TO AND FROM THE STANDARD FORMAT

The following subroutines permit one to convert sparse matrices to and from the standard format.

CALL CVRS(A,ka,m,n,B,IB,JB,NUM,IERR)
CALL CVCS(A,ka,m,n,B,IB,JB,NUM,IERR)

A is an m $\times$ n matrix stored in the standard format. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that A is to be stored in sparse form in the arrays B, IB, JB. CVRS is used if A is a real matrix and B a real array, and CVCS is used if A is a complex matrix and B a complex array.

The input argument NUM is the estimated maximum number of elements that will appear in B and JB. It is assumed that B and JB are of dimension max $\{1,\text{NUM}\}$ and that IB is of dimension m + 1. IERR is an integer variable. When the routine is called, if NUM specifies sufficient storage for B and JB, then A is stored in B, IB, JB and IERR is assigned the value 0.

Error Return. If it is found that there is not sufficient storage in B and JB for the i^{th} row of A, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A will have been stored in B and JB, and IB(1),...,IB(i) will contain the appropriate row locations.

Remark. No zero elements of A are stored in B, and the elements of each row of B are ordered so that the column indices of the elements of the row are in ascending order.

Example. If A is the m $\times$ n zero matrix then NUM can be set to 0. In this case the result will be IB(1) = $\cdots$ = IB(m + 1) = 1.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris

CALL CVSR(A,IA,JA,B,kb,m,n)
CALL CVSC(A,IA,JA,B,kb,m,n)

A is an m $\times$ n sparse matrix stored in the arrays A, IA, JA, and B is a 2-dimensional array of dimension kb $\times$ n where kb $\geqslant$ m. CVSR is used if A and B are real arrays, and CVSC is used if A and B are complex arrays. When the routine is called, the matrix A is stored in the array B in the standard format.

Note. It is assumed that the storage areas A and B do not overlap.

Programmer. A. H. Morris

COMPUTING CONJUGATES OF SPARSE COMPLEX MATRICES

If $A = (a_{ij})$ is a complex matrix stored in sparse form, then the following subroutine is available for computing the conjugate matrix $\bar{A} = (\bar{a}_{ij})$.

CALL CSCONJ(A,IA,JA,B,IB,JB,m)

It is assumed that the sparse complex matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k. A and B are complex arrays. It is assumed that the matrix A has $m \geqslant 1$ rows and that IB is an array of dimension m + 1. When the routine is called, the conjugate matrix $\bar{A}$ is stored in B, IB, JB.

Remark. The user may let B = A, IB = IA, and JB = JA.

Programmer. A. H. Morris

TRANSPOSING SPARSE REAL MATRICES

The subroutines RPOSE and RPOSE1 are available for transposing a sparse $m \times n$ real matrix A. RPOSE1 is more general than RPOSE. For any permutation $\pi = \{i_1,...,i_m\}$ of $\{1,...,m\}$ let P denote the corresponding $m \times m$ permutation matrix. Then RPOSE1 computes the matrix $(PA)^t$.

CALL RPOSE(A,IA,JA,B,IB,JB,m,n)

It is assumed that the sparse matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n + 1$. When RPOSE is called, the transpose A^t is stored in B, IB, JB.

Remarks. RPOSE orders the elements of each row of A^t so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software 4*** (1978), 3, pp. 250–269.

CALL RPOSE1(π,A,IA,JA,B,IB,JB,m,n)

It is assumed that π is an integer array of dimension m containing the data $\{i_1,...,i_m\}$, and that the sparse matrix A is stored in the arrays A, IA, JA. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension $n + 1$. When RPOSE1 is called, $(PA)^t$ is computed and the results stored in B, IB, JB.

Remarks. RPOSE1 orders the elements of each row of $(PA)^t$ so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A.H. Morris

Reference. Gustavson, F.G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software 4*** (1978), 3, pp. 250–269.

TRANSPOSING SPARSE COMPLEX MATRICES

The subroutines CPOSE and CPOSE1 are available for transposing a sparse $m \times n$ complex matrix A. CPOSE1 is more general than CPOSE. For any permutation $\pi = \{i_1, \ldots, i_m\}$ of $\{1, \ldots, m\}$ let P denote the corresponding $m \times m$ permutation matrix. Then CPOSE1 computes the matrix $(PA)^t$.

<u>CALL CPOSE(A,IA,JA,B,IB,JB,m,n)</u>

It is assumed that the sparse matrix A is stored in the arrays A, IA, JA. A and B are complex arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension n + 1. When CPOSE is called, the transpose A^t is stored in B, IB, JB.

Remarks. CPOSE orders the elements of each row of A^t so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," *ACM Trans. Math Software 4* (1978), 3, pp. 250-269.

<u>CALL CPOSE1(π,A,IA,JA,B,IB,JB,m,n)</u>

It is assumed that π is an integer array of dimension m containing the data $\{i_1, \ldots, i_m\}$, and that the sparse matrix A is stored in the arrays A, IA, JA. A and B are complex arrays. If A and JA contain k elements, then it is also assumed that B and JB are arrays of dimension k and that IB is an array of dimension n + 1. When CPOSE1 is called, $(PA)^t$ is computed and the results stored in B, IB, JB.

Remarks. CPOSE1 orders the elements of each row of $(PA)^t$ so that the column indices of the elements of the row are in ascending order. However, it does no checking for zero elements in A. If zero elements appear in the array A, then the zero elements will also appear in B.

Restriction. It is assumed that the storage areas B, IB, JB do not overlap with the storage areas A, IA, JA.

Programmer. A. H. Morris

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software 4*** (1978), 3, pp. 250-269.

ADDITION OF SPARSE MATRICES

Let A and B be sparse $m \times n$ matrices stored in the arrays A, IA, JA and B, IB, JB. The sum C = A + B can be computed by the following subroutines.

CALL RADD(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)
CALL RADD1(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)

It is assumed that A and B are real matrices, and that A + B is to be stored in the arrays C, IC, JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension max $\{1,\text{NUM}\}$ and that IC is of dimension $m + 1$.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines.

IERR is an integer variable. When RADD or RADD1 is called, if NUM specifies sufficient storage for C and JC, then A + B is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of A + B, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A + B will have been computed and stored in C and JC. Also IC(1),...,IC(i) will contain the appropriate row locations.

Remark. RADD and RADD1 differ in their storage operations. RADD makes certain that no zero elements of A + B appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. RADD1 does neither of these things, and hence takes less time. However, RADD1 may require more storage.

Example. If A is a sparse $m \times m$ matrix and $B = -A$ where A and B contain k elements, then when RADD1 is called, NUM must be assigned a value $\geqslant k$ and the resulting sparse matrix C will consist entirely of zeros. However, if RADD is called, then NUM can be set to 0 and the result will be $IC(1) = \cdots = IC(m+1) = 1$ (the zero matrix).

Note. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JB.

Programmer. A. H. Morris

CALL CADD(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)
CALL CADD1(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)

It is assumed that A and B are complex matrices, and that A + B is to be stored in the arrays C, IC, JC. A, B, and C are complex arrays and NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension $\max\{1,\text{NUM}\}$ and that IC is of dimension m + 1.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines. X is a complex array.

IERR is an integer variable. When CADD or CADD1 is called, if NUM specifies sufficient storage for C and JC, then A + B is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of A + B, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A + B will have been computed and stored in C and JC. Also IC(1),...,IC(i) will contain the appropriate row locations.

Remark. CADD and CADD1 differ in their storage operations. CADD makes certain that no zero elements of A + B appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. CADD1 does neither of these things, and hence takes less time. However, CADD1 may require more storage.

Example. If A is a sparse $m \times m$ matrix and $B = -A$ where A and B contain k elements, then when CADD1 is called, NUM must be assigned a value $\geqslant k$ and the resulting sparse matrix C will consist entirely of zeros. However, if CADD is called then NUM can be set to 0 and the result will be $IC(1) = \cdots = IC(m + 1) = 1$ (the zero matrix).

Note. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JB.

Programmer. A. H. Morris

SUBTRACTION OF SPARSE MATRICES

Let A and B be sparse m $\times$ n matrices stored in the arrays A, IA, JA and B, IB, JB. The difference C = A − B can be computed by the following subroutines.

CALL RSUB(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)
CALL RSUB1(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)

It is assumed that A and B are real matrices, and that A − B is to be stored in the arrays C, IC, JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension max $\{1,\text{NUM}\}$ and that IC is of dimension m + 1.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines.

IERR is an integer variable. When RSUB or RSUB1 is called, if NUM specifies sufficient storage for C and JC, then A − B is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of A − B, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A − B will have been computed and stored in C and JC. Also IC(1),...,C(i) will contain the appropriate row locations.

Remark. RSUB and RSUB1 differ in their storage operations. RSUB makes certain that no zero elements of A − B appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. RSUB1 does neither of these things, and hence takes less time. However, RSUB1 may require more storage.

Example. If A is a sparse m $\times$ n matrix containing k elements and B = A, then when RSUB1 is called, NUM must be assigned a value $\geqslant$k and the resulting sparse matrix C will consist entirely of zeros. However, if RSUB is called then NUM can be set to 0 and the result will be IC(1) = $\cdots$ = IC(m + 1) = 1 (the zero matrix).

Note. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JB.

Programmer. A. H. Morris

CALL CSUB(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)
CALL CSUB1(A,IA,JA,B,IB,JB,C,IC,JC,m,n,X,IX,NUM,IERR)

It is assumed that A and B are complex matrices, and that A − B is to be stored in the arrays C, IC, JC. A, B, and C are complex arrays and NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension max $\{1,\text{NUM}\}$ and that IC is of dimension m + 1.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines. X is a complex array.

IERR is an integer variable. When CSUB or CSUB1 is called, if NUM specifies sufficient storage for C and JC, then A − B is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of A − B, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of A − B will have been computed and stored in C and JC. Also IC(1),...,IC(i) will contain the appropriate row locations.

Remark. CSUB and CSUB1 differ in their storage operations. CSUB makes certain that no zero elements of A − B appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. CSUB1 does neither of these things, and hence takes less time. However, CSUB1 may require more storage.

Example. If A is a sparse m × n matrix containing k elements and B = A, then when CSUB1 is called, NUM must be assigned a value $\geqslant k$ and the resulting sparse matrix C will consist entirely of zeros. However, if CSUB is called then NUM can be set to 0 and the result will be IC(1) = ··· = IC(m + 1) = 1 (the zero matrix).

Note. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JB.

Programmer. A. H. Morris

MULTIPLICATION OF SPARSE MATRICES

Let A be a sparse $\ell \times m$ matrix stored in the arrays A, IA, JA and B a sparse $m \times n$ matrix stored in the arrays B, IB, JB. The product C = AB can be computed by the following subroutines.

CALL RMLT(A,IA,JA,B,IB,JB,C,IC,JC,ℓ,m,n,X,IX,NUM,IERR)
CALL RMLT1(A,IA,JA,B,IB,JB,C,IC,JC,ℓ,m,n,X,IX,NUM,IERR)

It is assumed that A and B are real matrices, and that the product AB is to be stored in the arrays C, IC, JC. NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension max $\{1,\text{NUM}\}$ and that IC is of dimension $\ell + 1$.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines.

IERR is an integer variable. When RMLT or RMLT1 is called, if NUM specifies sufficient storage for C and JC, then AB is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of AB, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of AB will have been computed and stored in C and JC. Also IC(1),...,IC(i) will contain the appropriate row locations.

Remark. RMLT and RMLT1 differ in their storage operations. RMLT makes certain that no zero elements of AB appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. RMLT1 does neither of these things, and hence takes less time. However, RMLT1 may require more storage.

Restriction. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JC.

Programmer. A. H. Morris

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software 4*** (1978), 3, pp. 250–269.

CALL CMLT(A,IA,JA,B,IB,JB,C,IC,JC,ℓ,m,n,X,IX,NUM,IERR)
CALL CMLT1(A,IA,JA,B,IB,JB,C,IC,JC,ℓ,m,n,X,IX,NUM,IERR)

It is assumed that A and B are complex matrices, and that the product AB is to be stored in the arrays C, IC, JC. A, B, and C are complex arrays and NUM is the estimated maximum number of elements that will appear in C and JC. It is assumed that C and JC are of dimension max $\{1,\text{NUM}\}$ and that IC is dimension $\ell + 1$.

X and IX are arrays of dimension n or larger. These arrays are work spaces for the routines. X is a complex array.

IERR is an integer variable. When CMLT or CMLT1 is called, if NUM specifies sufficient storage for C and JC, then AB is computed and the results stored in C, IC, JC. Also IERR is assigned the value 0.

Error Return. If there is not sufficient storage in C and JC for the i^{th} row of AB, then IERR is set to i and the routine terminates. In this case, if $i > 1$ then the first $i - 1$ rows of AB will have been computed and stored in C and JC. Also IC(1),...,IC(i) will contain the appropriate row locations.

Remark. CMLT and CMLT1 differ in their storage operations. CMLT makes certain that no zero elements of AB appear in C, and it orders the elements of each row of C so that the column indices of the elements of the row are in ascending order. CMLT1 does neither of these things, and hence takes less time. However, CMLT1 may require more storage.

Restriction. It is assumed that the storage areas C, IC, JC do not overlap with the storage areas A, IA, JA and B, IB, JC.

Programmer. A. H. Morris

Reference. Gustavson, F. G., "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," ***ACM Trans. Math Software 4*** (1978), 3, pp. 250-269.

PRODUCT OF A REAL SPARSE MATRIX AND VECTOR

Let A be a real m × n sparse matrix stored in the arrays A, IA, JA. Then the following subroutines are available for multiplying A with a real vector.

CALL MVPRD(m,n,A,IA,JA,x,y)

The argument x is a column vector of dimension n and y an array of dimension m. When MVPRD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL MVPRD1(m,n,A,IA,JA,x,y)

The arguments x and y are column vectors of dimension n and m respectively. When MVPRD1 is called, Ax + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL MTPRD(m,n,A,IA,JA,x,y)

The argument x is a row vector of dimension m and y an array of dimension n. When MTPRD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

CALL MTPRD1(m,n,A,IA,JA,x,y)

The arguments x and y are row vectors of dimension m and n, respectively. When MTPRD1 is called, xA + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

PRODUCT OF A COMPLEX SPARSE MATRIX AND VECTOR

Let A be a complex m × n sparse matrix stored in the arrays A, IA, JA. Then the following subroutines are available for multiplying A with a complex vector.

<u>CALL CVPRD(m,n,A,IA,JA,x,y)</u>

The argument x is a column vector of dimension n and y an array of dimension m. A, x, y are complex arrays. When CVPRD is called, Ax is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

<u>CALL CVPRD1(m,n,A,IA,JA,x,y)</u>

The arguments x and y are column vectors of dimension n and m, respectively. A, x, y are complex arrays. When CVPRD1 is called, Ax + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

<u>CALL CTPRD(m,n,A,IA,JA,x,y)</u>

The argument x is a row vector of dimension m and y an array of dimension n. A, x, y are complex arrays. When CTPRD is called, xA is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

<u>CALL CTPRD1(m,n,A,IA,JA,x,y)</u>

The arguments x and y are row vectors of dimension m and n, respectively. A, x, y are complex arrays. When CTPRD1 is called, xA + y is computed and stored in y.

Remark. It is assumed that the arrays A, x, y do not overlap.

Programmer. A. H. Morris

ORDERING THE ROWS OF A SPARSE MATRIX BY INCREASING LENGTH

Let A be a sparse m $\times$ n matrix stored in the arrays A, IA, JA. The following subroutine is available for ordering the rows of the matrix by increasing length.

CALL SPORD(m,n,IA,R,IWK)

R is an integer array of dimension m. When SPORD is called, the rows of the matrix are ordered by increasing length. The row ordering is given in R.

IWK is an integer array of dimension m + n + 1 or larger that is used for a work space.

Remark. If rows $i_1,\dots,i_k$ are the rows of length ℓ, then the indices $i_1,\dots,i_k$ are listed in R in increasing sequence.

Programmer. A. H. Morris

REORDERING SPARSE MATRICES INTO BLOCK TRIANGULAR FORM

Let A be a sparse $n \times n$ matrix stored in the arrays A, IA, JA. Then the subroutine BLKORD is available for reordering the rows and columns of A so that one has a lower block triangular matrix

$$(*) \qquad \begin{pmatrix} A_{11} & & \mathbf{0} \\ A_{21} & A_{22} & \\ \vdots & \vdots & \ddots \\ A_{k1} & A_{k2} \cdots & A_{kk} \end{pmatrix}$$

where the blocks A_{ii} are square and cannot themselves be reordered into lower block triangular form.

<u>CALL BLKORD(n,IA,JA,R,C,IB,k,IWK,IERR)</u>

R and C are integer arrays of dimension n, and IERR is an integer variable. When BLKORD is called, the rows of the matrix are first ordered so that the main diagonal contains a maximum number of nonzeros. After this is done then IERR = the number of zeros that appear on the diagonal. If IERR = 0 then the rows and columns of the matrix are ordered into block triangular form (*). The row ordering is given in R and the column ordering is given in C.

IB is an integer array of dimension n and k is an integer variable. When the matrix has been ordered into block triangular form (*) then k = the number of blocks A_{ii}. Also IB(i) = the row number in the block triangular matrix of the beginning of block A_{ii} $(i = 1,\dots,k)$.

IWK is an integer array of dimension 5n or larger that is used for a work space by the routine.

Error Return. If IERR $\neq 0$ then the routine terminates. In this case, R contains the row ordering that gives the main diagonal with the maximum number of nonzeros.

Remark. IA, JA, and n are not modified by the routine.

Programming. BLKORD employs the subroutines MC21A, MC21B, MC13D, MC13E designed by I. S. Duff and J. K. Reid (AERE Harwell, England).

References

(1) Duff, I. S., "On Algorithms for Obtaining a Maximum Transversal," ***ACM Trans. Math Software 7*** (1981), pp. 315-330.

(2) Duff, I. S. and Reid, J. K., "An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix," ***ACM Trans. Math Software 4*** (1978), pp. 137-147.

SOLUTION OF SPARSE SYSTEMS OF REAL LINEAR EQUATIONS

Let A be a nonsingular n $\times$ n sparse real matrix stored in the arrays A, IA, JA and let b be a real column vector of dimension n. The subroutines SPSLV and RSLV are available for solving the system of equations Ax = b, and the subroutine TSLV is available for solving the transposed system of equations $A^t x = b$. These routines employ partial pivot gauss elimination with column interchanges to first obtain an LU decomposition of A. If SPSLV is called then only the off-diagonal nonzero elements of U are stored, and then the equations are solved. However, if RSLV or TSLV is called then the off-diagonal nonzero elements of both L and U are stored. Thus RSLV and TSLV will frequently require at least double the amount of storage needed by SPSLV, but they can be recalled to solve other systems of equations Ax = r and $A^t x = r$ without having to redecompose the matrix A. Moreover, since RSLV and TSLV will always generate the same LU decomposition of A, RSLV can be called to decompose A and solve a system of equations Ax = b, and then TSLV can be called to solve a transposed system of equations $A^t x = r$ using the decomposition obtained by RSLV.

CALL SPSLV(n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)

It is assumed that $n \geqslant 1$ and that X is an array of dimension n. The solution of the system of equations Ax = b is computed and stored in X. A, IA, JA and b are not modified by the routine.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1,...,i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of a subroutine such as SPSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1,...,j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routine, and MAX is an input argument. When SPSLV is called, an LU decomposition of A is first obtained where U is a unit upper triangular matrix. The off-diagonal portion of U is stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of U that might be nonzero and have to be stored ($MAX \leqslant n(n-1)/2$). IWK is an integer array of dimension $3n + MAX + 2$ or larger, and WK is a real array of dimension $n + MAX$ or larger.

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

$IERR > 0$	$Ax = b$ was solved. $IERR = \max\{1,m\}$ where m is the total number of off-diagonal nonzero elements of U.
$IERR = 0$	The argument n is nonpositive.
$IERR = -k$	Row R(k) of A is null.
$IERR = -n - k$	Row R(k) of A has a duplicate entry.
$IERR = -2n - k$	Row R(k) of A has been reduced to a row containing only zeros.
$IERR = -3n - k$	Row k of the upper triangular matrix exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

Remarks. The amount of storage MAX depends critically on the row ordering given in R. If it is suspected that the rows and columns of A can be reordered so that one has a lower block triangular matrix

$$\begin{pmatrix} A_{11} & & \mathbf{0} \\ A_{21} & A_{22} & \\ \vdots & \vdots & \ddots \\ A_{k1} & A_{k2} \cdots & A_{kk} \end{pmatrix}$$

then the subroutine BLKORD should first be tried. This subroutine will specify an ordering for lower block triangular form if one exists. However, if such an ordering does not exist and one is uncertain how to order the rows, then the row ordering given by the subroutine SPORD frequently yields good results. In any case, the selection of an initial column ordering C is never bothersome since partial pivoting with column interchanges is performed. The initial ordering $C(i) = i$ $(i = 1,\ldots,n)$ always suffices.

Programming. SPSLV is a modification by A. H. Morris of the subroutine NSPIV. SPSLV employs the subroutine NSPIV1. NSPIV and NSPIV1 were written by Andrew H. Sherman (University of Texas at Austin).

Reference. Sherman, Andrew H., "Algorithms for Sparse Gaussian Elimination with Partial Pivoting," ***ACM Trans. Math Software 4*** (1978), pp. 330-338.

CALL RSLV(MO,n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)

CALL TSLV(MO,n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)

RSLV is called for solving $Ax = b$, and TSLV is called for solving $A^t x = b$. MO is an input argument which specifies if RSLV or TSLV is being called for the first time. On an initial call, MO = 0 and we have the following setup:

It is assumed that $n \geqslant 1$ and that X is an array of dimension n. The solution of the system of equations is stored in X. A, IA, JA are not modified by the routines. X and b may share the same storage area. If X is a separate storage area then b is not modified by the routines.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1,\ldots,i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of subroutines such as RSLV and TSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1,\ldots,j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routines, and MAX is an input argument. On an initial call to RSLV or TSLV, an LU decomposition of A is first obtained where L is a lower triangular matrix and U a unit upper triangular matrix. The off-diagonal portions of L and U are stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of L and U that might be nonzero and have to be stored ($MAX \leqslant n(n-1)$). IWK is an integer array of dimension $4n + MAX + 2$ or larger, and WK is a real array of dimension $2n + MAX$ or larger.

On an initial call to RSLV or TSLV, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR > 0	The system of equations was solved. IERR = max $\{1,m\}$ where m is the total number of off-diagonal nonzero elements of L and U.
IERR = 0	The argument n is nonpositive.
IERR = −k	Row R(k) of A is null.

IERR = $-n - k$	Row R(k) of A has a duplicate entry.
IERR = $-2n - k$	Row R(k) of A has been reduced to a row containing only zeros.
IERR = $-3n - k$	Row k of U or L exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

After an initial call to RSLV or TSLV, if IERR > 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. RSLV is called for solving the new set of equations $Ax = b$, and TSLV is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to RSLV or TSLV to solve the new system of equations. The solution is stored in X. As before, X and b may share the same storage area. If MO $\neq 0$ then only n, R, C, IWK, and WK are used. A, IA, JA, MAX, and IERR are not referenced by the routine.

Note. The remarks concerning the ordering of the rows and columns of A when SPSLV is used hold also for RSLV and TSLV.

Programming. RSLV calls the subroutines RSLV1 and SPLU, and TSLV calls the subroutines TSLV1 and SPLU. These routines were written by A. H. Morris.

SOLUTION OF SPARSE SYSTEMS OF COMPLEX LINEAR EQUATIONS

Let A be a nonsingular n $\times$ n sparse complex matrix stored in the arrays A, IA, JA and let b be a complex column vector of dimension n. The subroutines CSPSLV and CSLV are available for solving the system of equations Ax = b, and the subroutine CTSLV is available for solving the transposed system of equations $A^t x = b$. These routines employ partial pivot Gauss elimination with column interchanges to first obtain an LU decomposition of A. If CSPSLV is called then only the off-diagonal nonzero elements of U are stored, and then the equations are solved. However, if CSLV or CTSLV is called then the off-diagonal nonzero elements of both L and U are stored. Thus CSLV and CTSLV will frequently require at least double the amount of storage needed by CSPSLV, but they can be recalled to solve other systems of equations Ax = r and $A^t x = r$ without having to redecompose the matrix A. Moreover, since CSLV and CTSLV will always generate the same LU decomposition of A, CSLV can be called to decompose A and solve a system of equations Ax = b, and then CTSLV can be called to solve a transposed system of equations $A^t x = r$ using the decomposition obtained by CSLV.

CALL CSPSLV(n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)

A, b, and X are complex arrays. It is assumed that $n \geqslant 1$ and that X is an array of dimension n. The solution of the system of equations Ax = b is computed and stored in X. A, IA, JA and b are not modified by the routine.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1,\dots,i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of a subroutine such as CSPSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1,\dots,j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routine, and MAX is an input argument. When CSPSLV is called, an LU decomposition of A is first obtained where U is a unit upper triangular matrix. The off-diagonal portion of U is stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of U that might be nonzero and have to be stored ($MAX \leqslant n(n-1)/2$). IWK is an integer array of dimension $3n + MAX + 2$ or larger, and WK is a complex array of dimension $n + MAX$ or larger.

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR > 0	$Ax = b$ was solved. IERR = max $\{1,m\}$ where m is the total number of off-diagonal nonzero elements of U.
IERR = 0	The argument n is nonpositive.
IERR = $-k$	Row R(k) of A is null.
IERR = $-n-k$	Row R(k) of A has a duplicate entry.
IERR = $-2n-k$	Row R(k) of A has been reduced to a row containing only zeros.
IERR = $-3n-k$	Row k of the upper triangular matrix exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

Remarks. The amount of storage MAX depends critically on the row ordering given in R. If it is suspected that the rows and columns of A can be reordered so that one has a lower block triangular matrix

$$\begin{pmatrix} A_{11} & & 0 \\ A_{21} & A_{22} & \\ \vdots & \vdots & \ddots \\ A_{k1} & A_{k2} \cdots & A_{kk} \end{pmatrix}$$

then the subroutine BLKORD should first be tried. This subroutine will specify an ordering for lower block triangular form if one exists. However, if such an ordering does not exist and one is uncertain how to order the rows, then the row ordering given by the subroutine SPORD frequently yields good results. In any case, the selection of an initial column ordering C is never bothersome since partial pivoting with column interchanges is performed. The initial ordering $C(i) = i$ ($i = 1,...,n$) always suffices.

Programming. CSPSLV is an adaptation by A. H. Morris of the subroutine NSPIV, written by Andrew H. Sherman (University of Texas at Austin). CSPSLV employs the subroutine CNSPIV.

Reference. Sherman, Andrew H., "Algorithms for Sparse Gaussian Elimination with Partial Pivoting," *ACM Trans. Math Software 4* (1978), pp. 330-338.

CALL CSLV(MO,n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)
CALL CTSLV(MO,n,A,IA,JA,b,R,C,MAX,X,IWK,WK,IERR)

CSLV is called for solving $Ax = b$, and CTSLV is called for solving $A^t x = b$. MO is an input argument which specifies if CSLV or CTSLV is being called for the first time. On an initial call, MO = 0 and we have the following setup:

A, b, and X are complex arrays. It is assumed that $n \geqslant 1$ and that X is an array of dimension n. The solution of the system of equations is stored in X. A, IA, JA are not modified by the routines. X and b may share the same storage area. If X is a separate storage area then b is not modified by the routines.

R is an integer array of n entries specifying the order in which the n rows of A are to be examined and processed. For example, if R contains the entries $i_1,\dots,i_n$ then the algorithm first performs operations on row i_1, next on row i_2, etc. It is well known that the order in which the rows of a sparse matrix are processed can have a significant impact on the overall performance of subroutines such as CSLV and CTSLV. Thus R must be chosen judiciously. R is not modified by the routine.

C is an integer array of n entries which plays a role similar to R. On input, C specifies a suggested order in which the n columns of A should be ordered for selection of the pivot elements. For example, if C contains the entries $j_1,\dots,j_n$ then it is suggested that the first pivot element may be from column j_1, the second pivot element from column j_2, etc. However, since partial pivoting with column interchange is performed, on output C may have been modified. On output, C will contain the actual order of the n columns from which the pivot elements were selected. This order will depend on A and R, and not on b.

IWK and WK are arrays for internal use by the routines, and MAX is an input argument. On an initial call to CSLV or CTSLV, an LU decomposition of A is first obtained where L is a lower triangular matrix and U a unit upper triangular matrix. The off-diagonal portions of L and U are stored in sparse form in IWK and WK. MAX is an estimate of the maximum number of off-diagonal elements of L and U that might be nonzero and have to be stored ($\text{MAX} \leqslant n(n-1)$). IWK is an integer array of dimension 4n + MAX + 2 or larger, and WK is a complex array of dimension 2n + MAX or larger.

On an initial call to CSLV or CTSLV, IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR > 0	The system of equations was solved. IERR = max $\{1,m\}$ where m is the total number of off-diagonal nonzero elements of L and U.
IERR = 0	The argument n is nonpositive.
IERR = −k	Row R(k) of A is null.

IERR = $-n-k$	Row R(k) of A has a duplicate entry.
IERR = $-2n-k$	Row R(k) of A has been reduced to a row containing only zeros.
IERR = $-3n-k$	Row k of U or L exceeds storage. MAX must be increased.

When an error is detected, the routine immediately terminates.

After an initial call to CSLV or CTSLV, if IERR > 0 on output then either routine may be called with MO $\neq 0$. When MO $\neq 0$ it is assumed that only b may have been modified. CSLV is called for solving the new set of equations $Ax = b$, and CTSLV is called for solving the new set of equations $A^t x = b$. The routine employs the LU decomposition obtained on the initial call to CSLV or CTSLV to solve the new system of equations. The solution is stored in X. As before, X and b may share the same storage area. If MO $\neq 0$ then only n, R, C, IWK, and WK are used. A, IA, JA, MAX, and IERR are not referenced by the routine.

Note. The remarks concerning the ordering of the rows and columns of A when CSPSLV is used hold also for CSLV and CTSLV.

Programming. CSLV calls the subroutines CSLV1 and CSPLU, and CTSLV calls the subroutines CTSLV1 and CSPLU. These routines were written by A. H. Morris.

COMPUTATION OF EIGENVALUES OF GENERAL REAL MATRICES

The subroutines EIG and EIG1 are available for computing the eigenvalues of real matrices. These routines frequently yield results accurate to 13-14 significant digits. Indeed, for symmetric matrices they may give 2 or more digits better accuracy than the routines designed specifically for symmetric matrices. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur when the matrix is not symmetric. In this case one should not expect more than 7-8 digit accuracy.

CALL EIG(IBAL,A,ka,n,WR,WI,IERR)
CALL EIG1(IBAL,A,ka,n,WR,WI,IERR)

A is a matrix of order n $\geqslant$ 1 and WR,WI are real arrays of dimension n or larger. When EIG or EIG1 is called then the eigenvalues $\lambda_1,\dots,\lambda_n$ of A are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

IBAL and ka are input arguments. The argument ka is the number of rows in the dimension statement for A in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the routines balance A before they compute the eigenvalues. Otherwise, if IBAL = 0 then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\dots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy by as much as 5 - 6 significant digits. Thus it is recommended that balancing be done.

(2) A is destroyed during computation. EIG and EIG1 reduce A to upper Hessenberg form and then apply the QR algorithm to obtain the eigenvalues. They differ only in the choice of transformations used to reduce A to upper Hessenberg form. EIG employs elementary similarity transformations and EIG1 employs orthogonal similarity

transformations. In theory the use of orthogonal transformations assures one of a tighter bound on the errors. However, since in practice matrices infrequently arise for which the orthogonal transformations actually generate more accurate results, and since the orthogonal transformations normally require more time than the elementary transformations, therefore EIG is the recommended routine.

Programming. EIG and EIG1 are driver routines for the EISPACK subroutines BALANC, ELMHS0, ORTHES, and HQR. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF GENERAL REAL MATRICES

The subroutines EIGV and EIGV1 are available for computing the eigenvalues and eigenvectors of real matrices. These routines are extensions of the respective eigenvalue routines EIG and EIG1. Thus all comments made concerning the accuracy of the eigenvalues produced by EIG and EIG1 apply also to EIGV and EIGV1. In particular, EIGV and EIGV1 can frequently yield high precision results for the eigenvalues if they are distinct. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2 - 3 units of the 14th significant digit, but the components of the corresponding eigenvector are only accurate to 9 - 10 significant digits. In the case of repeated eigenvalues the situation regarding the eigenvectors is totally unpredictable. The components of such an eigenvector may be correct to 6 - 7 significant digits, or the eigenvector may not even be an eigenvector! In this case the results should be checked.

```
CALL EIGV(IBAL,A,ka,n,WR,WI,ZR,ZI,IERR)
CALL EIGV1(IBAL,A,ka,n,WR,WI,ZR,ZI,IERR)
```

A is a matrix of order $n \geqslant 1$ and WR,WI are real arrays of dimension n or larger. When EIGV or EIGV1 is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding eigenvectors $z_1, \ldots, z_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

The input argument ka is the number of rows in the dimension statement for A in the calling program. ZR and ZI are real arrays of dimension ka $\times$ n. For $j = 1, \ldots, n$ the real parts of the components of the eigenvector z_j are stored in the jth column of ZR (in locations ZR(1, j),...,ZR(n, j)) and the imaginary parts are stored in the jth column of ZI. The eigenvectors $z_1, \ldots, z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the routines balance A before they compute the eigenvalues and eigenvectors. Otherwise, if IBAL = 0 then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the jth eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy of the eigenvalues by as much as 5-6 significant digits. When this occurs balancing will normally be needed to obtain the eigenvectors. In general, it is recommended that balancing be done.

(2) A is destroyed during computation. EIGV and EIGV1 both reduce A to upper Hessenberg form, apply the QR algorithm to the Hessenberg matrix to obtain the eigenvalues, and then backsubstitute to generate the eigenvectors. They differ only in the choice of transformations used to reduce A to upper Hessenberg form. EIGV employs elementary similarity transformations and EIGV1 employs orthogonal similarity transformations. In theory the use of orthogonal transformations assures one of a tighter bound on the errors. However, since in practice matrices infrequently arise for which the orthogonal transformations actually generate more accurate results, and since the orthogonal transformations normally require more time than the elementary transformations, therefore EIGV is the recommended routine.

Programming. EIGV and EIGV1 are driver routines for the EISPACK subroutines BALANC, ELMHS0, ORTHES, ELTRN0, ORTRAN, HQR2, and BALBAK. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES OF REAL MATRICES

The subroutine DEIG is available for the double precision computation of the eigenvalues of real matrices. This routine frequently yields results accurate to 26-28 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 13-14 digit accuracy.

CALL DEIG(IBAL,A,ka,n,WR,WI,IERR)

A is a double precision matrix of order $n \geqslant 1$ and WR, WI are double precision arrays of dimension n or larger. When DEIG is called then the eigenvalues $\lambda_1,\dots,\lambda_n$ of A are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

IBAL and ka are input arguments. The argument ka is the number of rows in the dimension statement for A in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the routine balances A before it computes the eigenvalues. Otherwise, if IBAL $= 0$ then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the jth eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\dots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks
(1) A is destroyed during computation.
(2) DEIG is a double precision version of the eigenvalue routine EIG1.

Programming. DEIG is a driver routine for the subroutines DBAL, DORTH, and DHQR. These subroutines are double precision versions of the EISPACK subroutines BALANC, ORTHES, and HQR, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF REAL MATRICES

The subroutine DEIGV is available for the double precision computation of the eigenvalues and eigenvectors of real matrices. This routine frequently yields values for the eigenvalues that are accurate to 26-28 significant digits. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. If the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 13-14 digit accuracy.

CALL DEIGV(IBAL,A,ka,n,WR,WI,ZR,ZI,IERR)

A is a double precision matrix of order $n \geqslant 1$ and WR, WI are double precision arrays of dimension n or larger. When DEIGV is called then the eigenvalues $\lambda_1,\dots,\lambda_n$ and corresponding eigenvectors $z_1,\dots,z_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered except that complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part being first.

The input argument ka is the number of rows in the dimension statement for A in the calling program. ZR and ZI are double precision arrays of dimension ka $\times$ n. For $j = 1,\dots,n$ the real parts of the components of the eigenvector z_j are stored in the jth column of ZR (in locations ZR(1,j),...,ZR(n,j)) and the imaginary parts are stored in the jth column of ZI. The eigenvectors $z_1,\dots,z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the routine balances A before it computes the eigenvalues and eigenvectors. Otherwise, if IBAL = 0 then A is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the jth eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\dots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks

(1) A is destroyed during computation.

(2) DEIGV is a double precision version of the eigenvalue/eigenvector routine EIGV1.

Programming. DEIGV is a driver routine for the subroutines DBAL, DORTH, DORTRN, DHQR2, and DBABK. These subroutines are double precision versions of the EISPACK routines BALANC, ORTHES, ORTRAN, HQR2, and BALBAK, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES OF SYMMETRIC REAL MATRICES

The subroutines SEIG and SEIG1 are available for computing the eigenvalues of symmetric real matrices. These routines frequently yield high precision results. SEIG is faster than SEIG1, but at times SEIG1 will produce better results when the symmetric matrix is tridiagonal. For arbitrary symmetric matrices it is not clear if there is any difference in the reliability of the routines.

CALL SEIG(A,ka,n,W,T,IERR)
CALL SEIG1(A,ka,n,W,T,IERR)

A is a symmetric matrix of order $n \geqslant 1$ and W an array of dimension n or larger. When SEIG or SEIG1 is called the eigenvalues $\lambda_1,...,\lambda_n$ are computed and stored in W(1),...,W(n). The eigenvalues are ordered so that $\lambda_1 \leqslant \cdots \leqslant \lambda_n$.

A may be packed or in standard form.[1] The input argument ka is a nonnegative integer. If ka = 0 then A is assumed to be packed. Otherwise, if ka $\neq$ 0 then A is assumed to be in the standard format. In this case ka has the value:

ka = the number of rows in the dimension statement for A in the calling program

It is assumed that ka $\geqslant$ n. However, it is not required that A(i,j) be defined for $i < j$. Only the lower triangular elements of A are used.

T is an array used for temporary storage. If SEIG is called then T must be of dimension 2n. However, if SEIG1 is called then T need only be of dimension n.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations of the QL algorithm are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1,...,\lambda_{j-1}$ will have been computed and stored in W. The eigenvalues are ordered so that $\lambda_1 \leqslant \cdots \leqslant \lambda_{j-1}$. However, they need not be the smallest eigenvalues of A.

Note. A is destroyed during computation.

Programming. SEIG and SEIG1 are driver routines for the EISPACK subroutines TRED1, TRED3, TQLRAT, and IMTQL1. These subroutines were developed at Argonne National Laboratory. The function SPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

[1] For details on the packed format see the section on packing and unpacking symmetric matrices.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF SYMMETRIC REAL MATRICES

The subroutines SEIGV and SEIGV1 are available for computing the eigenvalues and eigenvectors of symmetric real matrices. These routines frequently yield high precision results for the eigenvalues. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2–3 units of the 14th significant digit, but the components of the corresponding eigenvector are only accurate to 9–10 significant digits. SEIGV is faster than SEIGV1, but at times SEIGV1 will produce better results when the symmetric matrix is tridiagonal. For arbitrary symmetric matrices it is not clear if there is any difference in the reliability of the routines.

```
CALL SEIGV(A,ka,n,W,Z,T,IERR)
CALL SEIGV1(A,ka,n,W,Z,T,IERR)
```

A is a symmetric matrix of order $n \geqslant 1$ and W is an array of dimension n or larger. When SEIG or SEIGV1 is called the eigenvalues $\lambda_1,\ldots,\lambda_n$ and corresponding orthonormal eigenvectors $z_1,\ldots,z_n$ are computed. The eigenvalues are stored in W(1),...,W(n) and are ordered so that $\lambda_1 \leqslant \cdots \leqslant \lambda_n$.

A must be in the standard format, having the dimension ka $\times$ n. It is assumed that ka $\geqslant$ n. However, it is not required that A(i,j) be defined for $i < j$. Only the lower triangular elements of A are used.

Z is an array of dimension ka $\times$ n or larger. For $j = 1,\ldots,n$ the components of the eigenvector z_j are stored in the jth column of Z (in locations Z(1,j),...,Z(n,j)). To conserve memory one can let A and Z denote the same array.

T is an array of dimension n used for temporary storage.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations of the QL algorithm are required to compute the jth eigenvalue λ_j, then IERR is set to j. In this case, if $j > 1$ then the eigenvalues $\lambda_1,\ldots,\lambda_{j-1}$ and eigenvectors $z_1,\ldots,z_{j-1}$ will have been computed and stored in the W and Z arrays. However,the eigenvalues will be unordered.

Note. A is destroyed during computation.

Programming. SEIGV and SEIGV1 are driver routines for the EISPACK subroutines TRED2, TQL2, and IMTQL2. These subroutines were developed at Argonne National Laboratory. The function SPMPAR is also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES OF COMPLEX MATRICES

The subroutine CEIG is available for computing the eigenvalues of complex matrices. This routine frequently yields results accurate to 13-14 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 7-8 digit accuracy.

CALL CEIG(IBAL,AR,AI,ka,n,WR,WI,IERR)

AR and AI are real matrices of order $n \geqslant 1$, and WR and WI are real arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues are to be computed. When CEIG is called the eigenvalues $\lambda_1, \ldots, \lambda_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered.

IBAL and ka are input arguments. It is assumed that ka is the number of rows in the dimension statements for AR and AI in the calling program. IBAL may be any integer. If IBAL$\neq$0 then the complex matrix (represented by AR and AI) is balanced before the eigenvalues are computed. Otherwise, if IBAL = 0 then the complex matrix is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1}, \ldots, \lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy by as much as 5-6 significant digits. Thus it is recommended that balancing be done.

(2) AR and AI are destroyed during computation. CEIG reduces the complex matrix (represented by AR and AI) to upper Hessenberg form with unitary similarity transformations. Then the QR algorithm is used to obtain the eigenvalues.

Usage. If one has a complex matrix A, then AR and AI can be obtained using the matrix subroutines CMREAL and CMIMAG.

Programming. CEIG is a driver routine for the EISPACK subroutines CBAL, CORTH, and COMQR. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF COMPLEX MATRICES

The subroutine CEIGV is available for computing the eigenvalues and eigenvectors of complex matrices. This routine frequently yields values for the eigenvalues that are accurate to 13-14 significant digits. However, be aware that errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. It is not at all unusual to obtain an eigenvalue and eigenvector where the eigenvalue is correct to within 2-3 units of the 14th significant digit, but the components of the corresponding eigenvector are only accurate to 9-10 significant digits. If the eigenvalues of a matrix are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 7-8 digit accuracy, and the situation regarding the eigenvectors is totally unpredictable. The components of such an eigenvector may be correct to 6-7 significant digits, or the eigenvector may not even be an eigenvector! In this case the results should be checked.

CALL CEIGV (IBAL,AR,AI,ka,n,WR,WI,ZR,ZI,IERR,TEMP)

AR and AI are real matrices of order $n \geqslant 1$ and WR and WI are real arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues and eigenvectors are to be computed. When CEIGV is called the eigenvalues $\lambda_1,\ldots,\lambda_n$ and corresponding eigenvectors $z_1,\ldots,z_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered.

It is assumed that the input argument ka is the number of rows in the dimension statements for AR and AI in the calling program. ZR and ZI are real arrays of dimension ka $\times$ n. For j = 1,...,n the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1, j),...,ZR(n,j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1,\ldots,z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues and eigenvectors are computed. Otherwise, if IBAL = 0 then the complex matrix is not balanced.

TEMP is a real array used for temporary storage by the routine. If no balancing is to be done (i.e., if IBAL = 0) then TEMP must be of dimension 2n or larger. Otherwise, if balancing is to be performed then TEMP must be of dimension 3n or larger.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 30 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\ldots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks.

(1) Even though the balancing operation does not increase the theoretical bounds on the errors, nevertheless at times it may result in a slight loss of accuracy. On the other hand, balancing requires little additional time and in certain cases can improve the accuracy of the eigenvalues by as much as 5 - 6 significant digits. When this occurs balancing will normally be needed to obtain the eigenvectors. In general, it is recommended that balancing be done.

(2) AR and AI are destroyed during computation. CEIGV reduces the complex matrix (represented by AR and AI) to upper Hessenberg form with unitary similarity transformations. Then the QR algorithm is employed to obtain the eigenvalues, and backsubstitution is performed to generate the eigenvectors.

Usage. If one has a complex matrix A, then AR and AI can be obtained using the matrix subroutines CMREAL and CMIMAG.

Programming. CEIGV is a driver routine for the EISPACK subroutines CBAL, CORTH, COMQR2, and CBABK2. These subroutines were developed at Argonne National Laboratory. The functions SPMPAR and I1MACH are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES OF COMPLEX MATRICES

The subroutine DCEIG is available for the double precision computation of the eigenvalues of complex matrices. This routine frequently yields results accurate to 26-28 significant digits. However, if the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect more than 13-14 digit accuracy.

CALL DCEIG(IBAL,AR,AI,ka,n,WR,WI,IERR)

AR and AI are double precision matrices of order $n \geqslant 1$, and WR and WI are double precision arrays of dimension n or larger. AR and AI are the real and imaginary parts of the matrix whose eigenvalues are to be computed. When DCEIG is called the eigenvalues $\lambda_1,\ldots,\lambda_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered.

IBAL and ka are input arguments. It is assumed that ka is the number of rows in the dimension statements for AR and AI in the calling program. IBAL may be any integer. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues are computed. Otherwise, if IBAL = 0 then the complex matrix is not balanced.

Error Return. IERR is an integer variable. If all the eigenvalues are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\ldots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays.

Remarks.
(1) AR and AI are destroyed during computation.
(2) DCEIG is a double precision version of the eigenvalue routine CEIG.

Programming. DCEIG is a driver routine for the subroutines DCBAL, DCORTH, and DCOMQR. These subroutines are double precision versions of the EISPACK subroutines CBAL, CORTH, and COMQR, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DCPABS, DPMPAR, I1MACH and subroutine DCSQRT are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

DOUBLE PRECISION COMPUTATION OF EIGENVALUES AND EIGENVECTORS OF COMPLEX MATRICES

The subroutine DCEIGV is available for the double precision computation of the eigenvalues and eigenvectors of complex matrices. This routine frequently yields values for the eigenvalues that are accurate to 26-28 significant digits. However, be aware that the errors in the eigenvalues, no matter how seemingly insignificant, can be considerably magnified in the computation of the eigenvectors. If the eigenvalues are not distinct or if they are exceedingly tightly clustered, then a severe drop in accuracy can occur. In this case one should not expect the eigenvalues to have more than 13-14 digit accuracy.

CALL DCEIGV(IBAL,AR,AI,ka,n,WR,WI,ZR,ZI,IERR,TEMP)

AR and AI are double precision matrices of order $n \geqslant 1$ and WR and WI are double precision arrays of dimension n or larger. AR and AI are the real and imaginary portions of the complex matrix whose eigenvalues and eigenvectors are to be computed. When DCEIGV is called the eigenvalues $\lambda_1,\ldots,\lambda_n$ and corresponding eigenvectors $z_1,\ldots,z_n$ are computed. The real parts of the eigenvalues are stored in WR(1),...,WR(n) and the imaginary parts are stored in WI(1),...,WI(n). The eigenvalues are unordered.

It is assumed that the input argument ka is the number of rows in the dimension statements for AR and AI in the calling program. ZR and ZI are double precision arrays of dimension ka $\times$ n. For $j = 1,\ldots,n$ the real parts of the components of the eigenvector z_j are stored in the j^{th} column of ZR (in locations ZR(1,j),...,ZR(n,j)) and the imaginary parts are stored in the j^{th} column of ZI. The eigenvectors $z_1,\ldots,z_n$ are not normalized.

IBAL is an input argument that can be assigned any integer value. If IBAL $\neq 0$ then the complex matrix (represented by AR and AI) is balanced before the eigenvalues and eigenvectors are computed. Otherwise, if IBAL = 0 then the complex matrix is not balanced.

TEMP is a double precision array used for temporary storage by the routine. If no balancing is to be done (i.e., if IBAL = 0) then TEMP must be of dimension 2n or larger. Otherwise, if balancing is to be performed then TEMP must be of dimension 3n or larger.

Error Return. IERR is an integer variable. If all the eigenvalues and eigenvectors are found then IERR is set to 0. Otherwise, if more than 50 iterations are required to compute the j^{th} eigenvalue λ_j, then IERR is set to j and the routine terminates. In this case, if $j < n$ then the eigenvalues $\lambda_{j+1},\ldots,\lambda_n$ will have been computed and the results stored in the WR and WI arrays. However, none of the eigenvectors will have been computed. The eigenvectors are computed only after all the eigenvalues have been obtained.

Remarks.

(1) AR and AI are destroyed during computation.

(2) DCEIGV is a double precision version of the eigenvalue/eigenvector routine CEIGV.

Programming. DCEIGV is a driver routine for the subroutines DCBAL, DCORTH, DCMQR2, and DCBABK. These subroutines are double precision versions of the EISPACK subroutines CBAL, CORTH, COMQR2, and CBABK2, developed at Argonne National Laboratory. The double precision versions were prepared by A. H. Morris. The functions DCPABS, DPMPAR, I1MACH and subroutine DCSQRT are also used.

Reference. Smith, B. T., Boyle, J. M., et al., ***Matrix Eigensystem Routines – EISPACK Guide*** (Second Edition), Springer-Verlag, 1976.

ℓ_1 SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND INEQUALITY CONSTRAINTS

Let A be a $k \times n$ matrix, C an $\ell \times n$ matrix, and E an $m \times n$ matrix. Also let b, d, and f be column vectors of dimensions k, ℓ, and m respectively. The following subroutine is available for obtaining a column vector x of dimension n which minimizes $\|Ax - b\|_1 = \sum_{i=1}^{k} |A_i x - b_i|$ subject to the constraints

$$Cx = d$$
$$Ex \leqslant f.$$

Here A_i denotes the ith row of A, and $Ex \leqslant f$ means that every component of Ex is less than or equal to the corresponding component of f.

CALL CL1(k,ℓ,m,n,Q,kq,KODE,TOL,ITER,X,RES,RNORM,WK,IWK)

It is assumed that $k \geqslant 1$, $\ell \geqslant 0$, $m \geqslant 0$, and $n \geqslant 1$. Q is a 2-dimensional array with kq rows and at least n + 2 columns where $kq \geqslant k + \ell + m + 2$. The matrices A,C,E and vectors b,d,f are stored in the first $k + \ell + m$ rows and n + 1 columns of Q as follows:

$$Q = \begin{pmatrix} A & b \\ C & d \\ E & f \end{pmatrix}$$

Q is modified by the routine.

KODE is a variable used for input/output purposes, X an array of dimension n + 2 or larger, and RES an array of dimension $k + \ell + m$ or larger. On input KODE is normally set by the user to 0. This indicates that $\|Ax - b\|_1$ is to be minimized subject only to the constraints $Cx = d$ and $Ex \leqslant f$. However, if it is also desired that one or more variables x_j satisfy $x_j \leqslant 0$ or $x_j \geqslant 0$, or that one or more residuals $b_i - A_i x$ satisfy $b_i - A_i x \leqslant 0$ or $b_i - A_i x \geqslant 0$, then the user may set KODE to a nonzero value. If KODE $\neq 0$ on input, then the user must also set X(j) and RES(i) to the values

$$X(j) = \begin{cases} -1.0 & x_j \leqslant 0 \\ 0.0 & x_j \text{ is unrestricted} \\ 1.0 & x_j \geqslant 0 \end{cases}$$

$$RES(i) = \begin{cases} -1.0 & b_i - A_i x \leqslant 0 \\ 0.0 & b_i - A_i x \text{ is unrestricted} \\ 1.0 & b_i - A_i x \geqslant 0 \end{cases}$$

for j = 1,...,n and i = 1,...,k to indicate the additional constraints which are desired.

RNORM is a variable. When CL1 is called, if a vector x is found that minimizes $||Ax - b||_1$ subject to the desired constraints, then KODE = 0 on output and the solution x is stored in X. Also RNORM is assigned the value $||Ax - b||_1$, b − Ax is stored in the first k locations of RES, d − Cx is stored in the next ℓ locations, and f − Ex is stored in the last m locations.

The input argument TOL is a tolerance. In effect, the routine cannot distinguish between 0 and any quantity whose magnitude is less than TOL. Normally the setting TOL = $10^{-2\nu/3}$ suffices where ν is the number of decimal digits of accuracy available.

When CL1 is called, a modified form of the simplex algorithm is used to minimize $\|Ax - b\|_1$. Frequently the routine requires less than 5(k + ℓ + m) iterations to perform this task. ITER is a variable used for input/output purposes. On input the user must set ITER to the maximum number of iterations that will be permitted. When the routine terminates, ITER has for its value the number of iterations that were performed.

On output KODE reports the status of the results. The routine assigns KODE one of the following values:

KODE = 0	The problem was solved.
KODE = 1	The problem has no solution.
KODE = 2	Sufficient accuracy cannot be maintained to solve the problem using the current value of TOL.
KODE = 3	The maximum number of iterations were performed. More iterations are needed.

When KODE ⩾ 1 on output, X contains the last vector $\bar{x}$ which was obtained, RNORM $= ||A\bar{x} - b||_1$, and RES contains the vectors $b - A\bar{x}$, $d - C\bar{x}$, and $f - E\bar{x}$.

WK is an array of dimension 2(k + ℓ + m + n) or larger, and IWK is an array of dimension 3(k + ℓ + m) + 2n or larger. WK and IWK are work spaces for the routine.

Programming. CL1 calls the subroutine KL1. CL1 was written by I. Barrodale and F. D. K. Roberts (University of Victoria, British Columbia, Canada).

References

(1) Barrodale, I. and Roberts, F. D. K., "An Improved Algorithm for Discrete ℓ_1 Linear Approximation," ***SIAM J. Numer. Analysis 10*** (1973), pp. 839-848.

(2) ——————————, "An Efficient Algorithm for Discrete ℓ_1 Linear Approximation with Linear Constraints," ***SIAM J. Numer. Analysis 15*** (1978), pp. 603-611.

(3) ——————————, "Algorithm 552, Solution of the Constrained ℓ_1 Linear Approximation Problem," ***ACM Trans. Math Software 6*** (1980), pp. 231-235.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

Given an $m \times n$ matrix A and an $m \times \ell$ matrix B. The column vectors $b_1,\dots,b_\ell$ of B specify ℓ distinct linear least squares problems

$$Ax_j = b_j \quad (j = 1,\dots,\ell).$$

This set of problems can be written in the form $AX = B$ where X is the $n \times \ell$ matrix having the column vectors $x_1,\dots,x_\ell$. There always exists a unique minimum length least squares solution x_j for each $Ax_j = b_j$. The subroutines LLSQ, HFTI, and HFTI2 are available for obtaining the minimum length solution matrix X. HFTI and HFTI2 are more general than LLSQ, being able to solve arbitrary systems $AX = B$. LLSQ assumes that $m \geqslant n > 1$ and that the rank of A is n. The routines perform Householder triangularization. HFTI and HFTI2 require more time than LLSQ, but may be more accurate. In LLSQ all calculations are performed in single precision. In HFTI and HFTI2 most inner products are computed in double precision and the results stored in single precision.

CALL LLSQ(m,n,A,ka,B,kb,ℓ,WK,IWK,IERR)

It is assumed that $m \geqslant n > 1$ and that the rank of A is n. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is required that $ka \geqslant m$ and $kb \geqslant m$.

IERR is an integer variable. When LLSQ is called, if no input errors are detected then IERR is set to 0 and the solution matrix X stored in B. Also, if $m \neq n$ then the residual norm $\|Ax_j - b_j\|$ is computed and stored in $B(n + 1,j)$ for $j = 1,\dots,\ell$.[1]

WK and IWK are arrays of dimension n or larger that are work spaces for the routine.

Error Return. $IERR \neq 0$ when $m \geqslant n > 1$ is not satisfied ($IERR = 1$) or the rank of A is less than n ($IERR = 2$).

Note. A is destroyed during computation.

Programming. LLSQ is a driver for the subroutines ORTHO and ORSOL, written by Nai-Kuan Tsao and Paul J. Nikolai (Aerospace Research Laboratories, Wright-Patterson Air Force Base).

[1] Throughout this section $\|c\| = \sqrt{\Sigma c_i^2}$ for any vector $c = (c_1,\dots,c_m)$.

Reference. Tsao, N. K. and Nikolai, P. J., ***Procedures using Orthogonal Transformations for Linear Least Squares Problems.*** Report ARL TR 74-0124, Aerospace Research Laboratories, Wright-Patterson Air Force Base, 1974.

CALL HFTI(A,ka,m,n,B,kb,ℓ,τ,k,RNORM,H,G,IP)
CALL HFTI2(A,ka,m,n,B,kb,ℓ,D,τ,k,RNORM,H,G,IP,IERR)

There are no restrictions on m and n (other than they both be positive integers). The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is required that ka $\geqslant$ m. Also, if $\ell \geqslant 1$ then kb $\geqslant \max\{m,n\}$.

If $\ell \geqslant 1$ then RNORM is an array of dimension ℓ or larger. When HFTI or HFTI2 is called, the minimum length solution matrix X is computed and stored in B. Also the residual norm $\|Ax_j - b_j\|$ is computed and stored in RNORM(j) for j = 1,...,ℓ.

H, G, and IP are arrays of dimension n or larger that are work spaces for the routines.

The parameters τ, k, and D.

τ is a tolerance that is set by the user, k a variable, and D an array of dimension $\min\{m,n\}$ or larger. It is assumed that $\tau \geqslant 0$. Normally $\tau = 0$ is the setting that is used. D and k are set by the routines.

In order to understand the use of τ, k, and D one must be briefly acquainted with the processing of A. The routines first reduce A to a triangular matrix C where A = QCP. Q is an orthogonal matrix and P a permutation matrix. P is defined so that the diagonal elements c_{ii} of C satisfy $|c_{ii}| \geqslant |c_{i+1,i+1}|$ for each i. The variable k is set to the largest integer such that $|c_{kk}| > \tau$, and if HFTI2 is used then the diagonal elements c_{ii} are stored in D. C is now regarded as the partitioned matrix

$$C = \begin{pmatrix} C_1 & C_2 \\ 0 & C_3 \end{pmatrix}$$

where C_1 is a k $\times$ k matrix. Minimum length least squares solutions x_j are then computed for the problems $Ax_j = b_j$ using only the first k rows of C. This is equivalent to replacing A with

$$\widetilde{A} = Q \begin{pmatrix} C_1 & C_2 \\ 0 & 0 \end{pmatrix} P$$

and solving $\widetilde{A}x_j = b_j$ for j = 1,...,ℓ.

Since $|c_{11}| \geqslant \cdots \geqslant |c_{kk}| > \tau$ clearly k is the rank of $\widetilde{A}$. It is also true that the ratio $|c_{11}|/|c_{kk}|$ is a lower bound on the condition number of C_1 (relative to the spectral norm). Thus, if the ratio is extremely large (say $\geqslant 10^8$) then a severe loss of accuracy can be expected. A large ratio may be due all or in part to rank deficiency (or near rank deficiency) of the matrix A. Fortunately, rank deficiency is frequently not too difficult to detect and cure. When A is rank deficient then machine roundoff may assign c_{kk} a small value, say $.21 * 10^{-14}$, when it should be 0. The cure is to examine the diagonal elements c_{ii} which are stored in D, to reset τ so as to eliminate the unwanted $c_{ii}'s$, and then to rerun the problem. This will reduce the order of C_1, thereby lowering the rank of the replacement matrix $\widetilde{A}$. C_1 will now be better conditioned, but the value of the residual norms $\|Ax_j - b_j\|$ may be larger. If the norms do increase, then the solution obtained will be satisfactory only if the size of the increased norms fall within acceptable bounds.

Remarks

(1) The variable k is set to 0 if all $|c_{ii}| \leqslant \tau$. If $k = 0$ then the zero matrix is the solution for $AX = B$.

(2) If $\ell \leqslant 0$ then the decomposition $A = QCP$ is performed, the diagonal elements of C are stored in D, and k is computed. B and RNORM are ignored.

(3) The contents of A are destroyed by the routines.

(4) HFTI and HFTI2 yield the same results.

Error Return. IERR is a variable that is set by the routine. If no input errors are detected then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	if $m > ka$
IERR = 2	if $\ell > 1$ and $kb < \max\{m,n\}$

When an error is detected, the routine immediately terminates.

Programming. HFTI and HFTI2 call the subroutine H12. These routines were written by Charles L. Lawson and Richard J. Hanson (Jet Propulsion Laboratory), and modified by A. H. Morris.

Reference. Lawson, C. L., and Hanson, R. J., ***Solving Least Squares Problems***, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974.

LEAST SQUARES SOLUTION OF OVERDETERMINED SYSTEMS OF LINEAR EQUATIONS WITH ITERATIVE IMPROVEMENT

Given an $m \times n$ matrix A and an $m \times \ell$ matrix B. The column vectors $b_1,...,b_\ell$ of B specify ℓ distinct linear least squares problems

$$Ax_j = b_j \qquad (j = 1,...,\ell).$$

This set of problems can be written in the form AX = B where X is the $n \times \ell$ matrix having the column vectors $x_1,...,x_\ell$. Assume that $m \geqslant n > 1$ and that the rank of A is n. Then there exists a unique least squares solution x_j for each $Ax_j = b_j$. The subroutine LLSQMP is available for obtaining the solution matrix X. Iterative improvement is performed to compute X to machine accuracy.

CALL LLSQMP(m,n,A,ka,B,kb,ℓ,WK,IWK,IERR)

It is assumed that $m \geqslant n > 1$ and that the rank of A is n. The input arguments ka and kb have the following values:

ka = the number of rows in the dimension statement for A in the calling program
kb = the number of rows in the dimension statement for B in the calling program

It is required that $ka \geqslant m$ and $kb \geqslant m$.

When LLSQMP is called, the solution X is computed and stored in B. Also, if $m \neq n$ then the residual norm $\|Ax_j - b_j\|$ is computed and stored in B(n+1,j) for $j = 1,...,\ell$.[1] A is not modified by the routine.

WK is an array of dimension $mn + 2m + n$ or larger, and IWK an array of dimension n or larger. WK and IWK are work spaces for the routine.

IERR is a variable that is set by the routine. When LLSQMP terminates, IERR has one of the following values:

IERR = 0	The solution X was computed to machine accuracy.
IERR = 1	X was obtained, but not to machine accuracy.
IERR = 2	The restriction $m \geqslant n > 1$ is not satisfied.
IERR = 3	The rank of A is less than n.

Programming. LLSQMP is a driver for the subroutines ORTHO, ORSOL, and ORIMP. These subroutines were written by Nai-Kuan Tsao and Paul J. Nikolai (Aerospace Research Laboratories, Wright-Patterson Air Force Base). ORIMP was modified by A. H. Morris. The function SPMPAR and subroutine MCOPY are also used.

[1] Here $\|c\| = \sqrt{\sum_i c_i^2}$ for any vector $c = (c_1,...,c_m)$.

Reference. Tsao, N. K., and Nikolai, P. J., ***Procedures using Orthogonal Transformations for Linear Least Squares Problems,*** Report ARL TR 74-0124, Aerospace Research Laboratories, Wright-Patterson Air Force Base, 1974.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND INEQUALITY CONSTRAINTS

Let A be an $m_a \times n$ matrix. E an $m_e \times n$ matrix, G an $m_g \times n$ matrix, b a column vector of dimension m_a, f a column vector of dimension m_e, and h a column vector of dimension m_g. The subroutine LSEI is available for finding a column vector x of dimension n that minimizes $\|Ax - b\|$ subject to the constraints

$$\|Ex - f\| = r$$

$$Gx \geqslant h$$

where $r = \min_y \|Ey - f\|$.[1] The inequality $Gx \geqslant h$ means that every component of the product Gx must be equal to or greater than the corresponding component of h. If the equation $Ey = f$ can be solved, then $r = 0$ and the constraint $\|Ex - f\| = r$ becomes the equality constraints $Ex = f$. It is assumed that $m_a \geqslant 0$, $m_e \geqslant 0$, and $m_g \geqslant 0$. If $m_a = 0$ then LSEI solves $\|Ex - f\| = r$ subject to the constraints $Gx \geqslant h$.

<u>CALL LSEI (W,kw,m_e,m_a,m_g,n,OPT,x,RNORME,RNORMA,MODE,WK,IWK)</u>

If $m = m_e + m_a + m_g$ then W is the $m \times (n + 1)$ matrix:

$$W = \begin{pmatrix} E & f \\ A & b \\ G & h \end{pmatrix}$$

The input argument kw is assumed to have the value:

kw = the number of rows in the dimension statement for W in the calling program

Thus it is required that $kw \geqslant m$.

RNORME and RNORMA are real variables. When LSEI is called, if the constraints $\|Ex - f\| = r$ and $Gx \geqslant h$ are consistent then x is computed, RNORME is assigned the value r, and RNORMA is assigned the value $\|Ax - b\|$.[2]

OPT is an array, called the ***option vector***, which permits the user to take advantage of certain options that are supplied by the routine. If no options are desired then OPT may be declared to have dimension 1 and OPT(1) must be assigned the value 1. The details concerning the available options and how to specify them in OPT are given below.

IWK is an array of dimension $m_g + 2n + 2$ or larger, and WK is an array of dimension $2(m_e + n) + \max\{m_a + m_g, n\} + (m_g + 2)(n + 7)$ or larger. IWK and WK are work spaces

[1] Throughout this section $\|c\|$ denotes the norn $\sqrt{\Sigma_i c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.

[2] If $m_e = 0$ then RNORME = 0, and if $m_a = 0$ then RNORMA = 0.

When LSEI is called, using a solution for $\|Ex - f\| = r$, a reduced least squares problem with inequality constraints is obtained and solved. When the routine terminates IWK(1), IWK(2), IWK(3) contain the following information:

IWK(1) = the estimated rank of the matrix E
IWK(2) = the estimated rank of the reduced problem
IWK(3) = the amount of storage in the array WK that was actually needed

IWK(3) may be important since the amount of storage required by WK depends only on A, E, G and not on the data b, f, h. The above formula for the dimension of WK may at times overestimate the amount of storage that is actually needed. For subsequent runs involving the same A, E, G but possibly different b, f, h the dimension of WK may be set to the value of IWK(3).

Error Return. MODE is an integer variable that is set by the routine. If no input errors are detected and restrictions $\|Ex - f\| = r$ and $Gx \geqslant h$ are consistent, then the problem is solved and MODE is set to 0 or 1 depending on whether $r = 0$ or $r > 0$. Otherwise, if an input error is detected or the restrictions are found to be inconsistent, then MODE is assigned one of the following values:

MODE = 2	The restrictions are not consistent and $r = 0$.
MODE = 3	The restrictions are not consistent and $r > 0$.
MODE = 4	An input error was detected. Either $kw < m$, the covariance matrix is requested and $kw < n$, or the option vector OPT is not defined properly.

When a MODE 2, 3, or 4 situation is detected then the routine immediately terminates. In these three cases X, RNORME, and RNORMA are not defined.

Remarks

(1) W is modified by the routine.
(2) If $m \leqslant 0$ or $n \leqslant 0$ then MODE is set to 0 and the routine terminates.

The option vector OPT. If no options are desired then OPT may be declared to be of dimension 1 and OPT(1) must have the value 1. Otherwise, OPT is a linked list consisting of groups of data link_i, key_i, data_i ($i = 1,\dots,s$). Each link_i and key_i requires one word of storage. The number of words required by data_i depends on the value of key_i. The general layout of OPT is as follows:

OPT(1) = link_1 (index of the first entry of the next group)
OPT(2) = key_1 (key to the option)
OPT(3) = the first word of the data (data_1) for this option
•
•
•
OPT(link_1) = link_2 (index of the first entry of the next group)
OPT($\text{link}_1 + 1$) = key_2 (key to the option)

OPT($link_1$ + 2) = the first word of the data ($data_2$) for this option

•
•
•

OPT($link_s$) = 1.0 (There are no more options to be considered.)

The following options are permitted:

key = 1	It is assumed that kw $\geqslant$ n. Compute the n $\times$ n covariance matrix and store it in the first n rows and columns of W.[1] The data for this option is a single value. It must be nonzero for the covariance matrix to be computed.
key = 2	Scale the nonzero columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$ so that they have length 1. The data for this option is a single value. It must be nonzero for the scaling to be performed.
key = 3	Scale the columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$. The data for this option consists of n scaling factors, one for each matrix column.
key = 4	Change the internal tolerance τ which is used for determining the rank of E. The data for this option is the new tolerance. τ may be set to any value $\geqslant \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6700). If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.
key = 5	Change the internal tolerance τ which is used for rank determination in the reduced least squares problem.[1] The data for this option is the new tolerance. τ may be set to any value $\geqslant \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1$. If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.

The order of the options in the array OPT is arbitrary. If an option has an unrecognized key then the option is ignored. It is assumed that the dimension of OPT is no greater than 100000 and that the number of options is $\leqslant$ 1000. If either of these assumptions is violated then MODE will be set to 4 and the routine will terminate. It is also required that $link_i \neq link_j$ for $i \neq j$. If this restriction is not satisfied then the linked list OPT will be circular, MODE will be set to 4, and the routine will terminate.

Remarks. LSEI employs the least squares routine WNNLS, which also examines the option vector OPT. WNNLS recognizes options having keys 5,6,7,8. Keys 6 and 7 should never appear in an option vector for LSEI. If a key 5 tolerance is defined then the tolerance is

[1] See the background notes near the end of this section.

used in WNNLS. The only exception is when a key 8 tolerance is defined for WNNLS. The key 8 tolerance will override the key 5 tolerance only if the key 8 option follows the key 5 option in the option vector.

Example. Assume that we have an array D containing n scaling factors for the columns of the matrix $\begin{pmatrix} E \\ A \\ G \end{pmatrix}$, and that the tolerance TOL is always to be used for rank determination. Then OPT will have to be of dimension $\geqslant n + 9$ and OPT can be defined as follows:

```
         OPT(1) = N + 3             (Scaling option)
         OPT(2) = 3.0
         DO 10 I = 1,N
     10  OPT(I + 2) = D(I)
         OPT(N + 3) = N + 6         (Matrix E tolerance option)
         OPT(N + 4) = 4.0
         OPT(N + 5) = TOL
         OPT (N + 6) = N + 9        (Reduced problem tolerance option)
         OPT(N + 7) = 5.0
         OPT(N + 8) = TOL
         OPT(N + 9) = 1.0           (There are no more options.)
```

Background. If $m_e \neq 0$ then a change of variables $x = Qy$ is first made where Q is an orthogonal matrix. Next a column vector y is found that minimizes $\| EQy - f \|$. Let k denote the estimated rank of E (which is stored in IWK(1)). Then the vector y is of the form $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ where y_1 has k components and y_2 has $n - k$ components. The vector y_1 is found for which $\|EQy - f\|$ is minimized. This minimization does not depend on y_2; i.e., for any y_2 the vector $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ satisfies $r = \min \| EQy - f \|$.

After the y_1 portion of y is found, then the problem is reduced to finding y_2 for which $\|AQy - b\|$ is minimized subject to $GQy \geqslant h$. The matrices AQ and GQ are partitioned into

$$AQ = (A_1 \ A_2)$$
$$GQ = (G_1 \ G_2)$$

where A_1 and G_1 contain k columns and A_2 and G_2 contain $n - k$ columns. Then the reduced problem is that of finding the vector y_2 which mimimizes $\|A_2 y_2 - (b - A_1 y_1)\|$ subject to the constraint $G_2 y_2 \geqslant h - G_1 y_1$. The internal tolerance used for rank determination in computing y_2 may be modified by the key = 5 option.

The covariant matrix C obtained by the key = 1 option may be defined as follows: Let ℓ denote the estimated rank of A_2 (which is stored in IWK(2)), and let $\bar{C}$ denote the pseudo-inverse of the matrix $A_2^t A_2$. Also let $\sigma^2 = 1$ when $m_a \leqslant n - k$ and let $\sigma^2 = \|Ax - b\|^2 / (m_a - \ell)$ when $m_a > n - k$. Then

$$C = \sigma^2 Q \begin{pmatrix} 0 & 0 \\ 0 & \bar{C} \end{pmatrix} Q^t .$$

This definition may not be meaningful when there are inequality constraints $Gx \geqslant h$.

***Programming*.** LSEI employs the subroutines LSI, LPDP, WNNLS, WNLSM, and WNLIT. These routines were written by Karen H. Haskell and Richard J. Hanson (Sandia Laboratories) and modified by A. H. Morris. The subroutines HFTI, H12, SROTM, SROTMG, SCOPY, SSWAP, SSCAL, SAXPY and functions SPMPAR, SDOT, SASUM, SNRM2, ISAMAX are also used.

***Reference*.** Haskell, K. H. and Hanson, R. J., ***Selected Algorithms for the Linearly Constrained Least Squares Problem – A User's Guide*.** Report SAND 78-1290, Sandia Laboratories, Albuquerque, New Mexico, 1979.

LEAST SQUARES SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY AND NONNEGATIVITY CONSTRAINTS

Let A be an $m_a \times n$ matrix, E an $m_e \times n$ matrix, b a column vector of dimension m_a, and f a column vector of dimension m_e. The subroutine WNNLS is available for finding a column vector $x = (x_1,\dots,x_n)^t$ that minimizes $\|Ax - b\|$ subject to the constraints:[1]

$$Ex = f$$
$$x_i \geqslant 0 \quad \text{for} \quad i > \ell$$

It is assumed that $m_a \geqslant 0$, $m_e \geqslant 0$, and $0 \leqslant \ell \leqslant n$. If $m_a = 0$ then WNNLS solves $Ex = f$ subject to the constraints $x_i \geqslant 0 \ (i > \ell)$.

CALL WNNLS(W,kw,m_e,m_a,n,ℓ,OPT,x,RNORM,MODE,IWK,WK)

If $m = m_e + m_a$ then W is the $m \times (n+1)$ matrix:

$$W = \begin{pmatrix} E & f \\ A & b \end{pmatrix}$$

The input argument kw is assumed to have the value:

kw = the number of rows in the dimension statement for W in the calling program

Thus it is required that $kw \geqslant m$.

RNORM is a real variable. The ***permissible vectors*** for the problem are those vectors x for which $x_i \geqslant 0 \ (i > \ell)$. When the routine is called, a permissible vector is found that minimizes $\|Ax - b\|$ subject to $\|Ex - f\| = \min\{\|Eu - f\| : u \text{ is permissible}\}$. Also RNORM is assigned the value $\sqrt{\|Ax - b\|^2 + \|Ex - f\|^2}$.[2]

OPT is an array, called the ***option vector***, which permits the user to take advantage of certain options that are supplied by the routine. If no options are desired then OPT may be declared to have dimension 1 and OPT(1) must be assigned the value 1. The details concerning the available options and how to specify them in OPT are given below.

IWK is an array of dimension m + n or larger, and WK is an array of dimension m + 5n or larger. IWK and WK are work spaces.

Error Return. MODE is an integer variable that is set by the routine. If the problem is successfully solved then MODE is assigned the value 0. Otherwise, MODE is assigned one of the following values:

MODE = 1 The maximum number of iterations ($3(n-\ell)$ iterations) was exceeded.

[1] Throughout this section $\|c\|$ denotes the norm $\sqrt{\Sigma\, c_i^2}$ for any vector $c = (c_1, c_2, \dots)$.

[2] If $m_a = 0$ then RNORM $= \|Ex - f\|$, and if $m_e = 0$ then RNORM $= \|Ax - b\|$.

An approximate solution and its residual norm are stored in x and RNORM.

MODE = 2 An input error was detected. Either one of the restrictions $kw \geqslant m$ and $0 \leqslant \ell \leqslant n$ is violated, or the option vector OPT is not properly defined.

When an input error is detected then the routine immediately terminates. In this case x and RNORM are not defined.

Remarks

(1) W is modified by the routine.

(2) If $m \leqslant 0$ or $n \leqslant 0$ then MODE is set to 0 and the routine terminates.

The option vector OPT. If no options are desired then OPT may be declared to be of dimension 1 and OPT(1) must have the value 1. Otherwise, OPT is a linked list consisting of groups of data link_i, key_i, data_i $(i = 1,\ldots,s)$. Each link_i and key_i requires one word of storage. The number of words required by data_i depends on the value of key_i. The general layout of OPT is as follows:

OPT(1) = link_1 (index of the first entry of the next group)
OPT(2) = key_1 (key to the option)
OPT(3) = the first word of the data (data_1) for this option

•
•
•

OPT(link_1) = link_2 (index of the first entry of the next group)
OPT(link_1 + 1) = key_2 (key to the option)
OPT(link_1 + 2) = the first word of the data (data_2) for this option

•
•
•

OPT(link_s) = 1.0 (There are no more options to be considered.)

The following options are permitted:

key = 6 Scale the nonzero columns of the matrix $\binom{E}{A}$ so that they have length 1. The data for this option is a single value. It must be nonzero for the scaling to be performed.

key = 7 Scale the columns of the matrix $\binom{E}{A}$. The data for this option consists of n scaling factors, one for each matrix column.

key = 8 Change the internal tolerance τ which is used for rank determination. The data for this option is the new tolerance. τ may be set to any value $\geqslant \epsilon$ where ϵ is the smallest floating point number for which $1 + \epsilon > 1$.

($\epsilon = 2^{-47}$ for the CDC 6700.) If the new value is less than ϵ then it is ignored and τ is set to ϵ. The default value employed for τ is $\sqrt{\epsilon}$.[1]

The order of the options in the array OPT is arbitrary. If an option has an unrecognized key then the option is ignored. It is assumed that the dimension of OPT is no greater than 100000 and that the number of options $\leqslant$ 1000. If either of these assumptions is violated then MODE will be set to 2 and the routine will terminate. It is also required that $\text{link}_i \neq \text{link}_j$ for $i \neq j$. If this restriction is not satisfied then the linked list OPT will be circular, MODE will be set to 2, and the routine will terminate.

Example. Assume that we have an array D containing n scaling factors for the columns of the matrix $\binom{E}{A}$, and that TOL is the tolerance to be used for rank determination. Then OPT will have to be of dimension $\geqslant$ n + 6 and OPT can be defined as follows:

```
         OPT(1) = N + 3              (Scaling option)
         OPT(2) = 7.0
         DO 10 I = 1, N
10       OPT(I + 2) = D(I)
         OPT(N + 3) = N + 6          (Tolerance option)
         OPT(N + 4) = 8.0
         OPT(N + 5) = TOL
         OPT(N + 6) = 1.0            (There are no more options.)
```

Programming. WNNLS employs the subroutines WNLSM and WNLIT. These routines were written by Karen H. Haskell and Richard J. Hanson (Sandia Laboratories), and modified by A. H. Morris. The subroutines H12, SROTM, SROTMG, SCOPY, SSWAP, SSCAL, SAXPY and functions SPMPAR, SASUM, SNRM2, ISAMAX are also used.

References

(1) Haskell, K. H. and Hanson, R. J., ***Selected Algorithms for the Linearly Constrained Least Squares Problem – A User's Guide.*** Report SAND 78-1290, Sandia Laboratories, Albuquerque, New Mexico, 1979.

(2) ____________, ***An Algorithm for Linear Least Squares Problems with Equality and Nonnegativity Constraints.*** Report SAND 77-0552, Sandia Laboratories, Albuquerque, New Mexico, 1978.

[1]This option may also be invoked by setting key = 5.

LEAST SQUARES ITERATIVE IMPROVEMENT SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH EQUALITY CONSTRAINTS

Let A be an $m_a \times n$ matrix, E an $m_e \times n$ matrix, B an $m_a \times \ell$ matrix, and F an $m_e \times \ell$ matrix. It is assumed that $0 \leqslant m_e \leqslant n$ and that the rank of E is m_e. Let $b_1,\ldots,b_\ell$ denote the column vectors of B and $f_1,\ldots,f_\ell$ the column vectors of F. The subroutine L2SLV is available for finding the unique minimum length column vector x_j of dimension n that minimizes $\|Ax_j - b_j\|$ subject to the equality constraints $Ex_j = f_j$ (if there are any) for $j = 1,\ldots,\ell$.[1] Iterative improvement is performed to compute the vectors $x_1,\ldots,x_\ell$ to machine accuracy. It is assumed that $m_a \geqslant 0$. If $m_a = 0$ then L2SLV finds the unique minimum length solution x_j to $Ex_j = f_j$ for $j = 1,\ldots,\ell$.

<u>CALL L2SLV(m,n,m_e,ℓ,$\tilde{A}$,ka,$\tilde{B}$,kb,WGTS,TOL,N1,IPIVOT,</u>
<u>X,kx,R,kr,T,kt,WK,IERR)</u>

If $m = m_e + m_a$ then $\tilde{A}$ is the $m \times n$ matrix $\begin{pmatrix} E \\ A \end{pmatrix}$ and $\tilde{B}$ is the $m \times \ell$ matrix $\begin{pmatrix} F \\ B \end{pmatrix}$. The input arguments ka and kb have the values:

ka = the number of rows in the dimension statement for $\tilde{A}$ in the calling program

kb = the number of rows in the dimension statement for $\tilde{B}$ in the calling program

It is assumed that $m \geqslant 1$, $n \geqslant 1$, $\ell \geqslant 1$, $ka \geqslant m$, and $kb \geqslant m$. $\tilde{A}$ and $\tilde{B}$ are not modified by the routine.

WGTS is an array containing m nonnegative weights. The first m_e weights are set to 1.0 by the routine. Let $w_1,\ldots,w_{m_a}$ denote the remaining weights (i.e., let w_i = WGTS($m_e + i$) for $i = 1,\ldots,m_a$). The remaining weights are supplied by the user. In effect, w_i is the weighting that is given to the ith equation in the least squares problem $Ax_j = b_j$. If W denotes the $m_a \times m_a$ diagonal matrix $\mathrm{diag}(w_1,\ldots,w_{m_a})$ then L2SLV finds the unique minimum length vector that minimizes $\|WAx_j - Wb_j\|$ subject to $Ex_j = f_j$ for $j = 1,\ldots,\ell$. Alternatively, x_j can be characterized as the unique minimum length vector that minimizes $\|Ax_j - b_j\|_w$ subject to $Ex_j = f_j$. Here $\|r\|_w$ denotes the weighted least squares norm $\sqrt{\Sigma_i w_i^2 r_i^2}$ for any vector $r = (r_1,\ldots,r_{m_a})$. For convenience, in the remainder of this section let $\hat{W}$ denote the $m \times m$ diagonal matrix $\mathrm{diag}(1,\ldots,1,w_1,\ldots,w_{m_a})$.

X is an $n \times \ell$ matrix that contains the solution vectors $x_1,\ldots,x_\ell$ when the routine terminates. The input argument kx is the number of rows in the dimension statement for X in the calling program. It is assumed that $kx \geqslant n$.

[1] Throughout this section $\|c\|$ denotes the norm $\sqrt{\Sigma_i c_i^2}$ for any vector $c = (c_1,\ldots,c_{m_a})$.

R is an $m \times \ell$ matrix. Let $\tilde{b}_j$ denote the jth column vector $\begin{pmatrix} f_j \\ b_j \end{pmatrix}$ of $\tilde{B}$ for $j = 1,\ldots,\ell$. Then L2SLV stores the residual vector $r_j = \tilde{W}\tilde{b}_j - \tilde{W}\tilde{A}x_j$ in the jth column of R. The input argument kr is the number of rows in the dimension statement for R in the calling program. It is assumed that $kr \geqslant m$.

WK is an array of dimension $6(m+n) + 2\ell$ or larger that is used for a work space. When L2SLV terminates, for $j = 1,\ldots,\ell$

$$WK(j) = \begin{cases} n_j & \text{if iterative improvement of the solution } x_j \text{ converged} \\ -n_j & \text{if iterative improvement of } x_j \text{ failed to converge} \end{cases}$$

where n_j is the number of iterations in the iterative improvement process that were performed in computing x_j. Also

$WK(\ell + j)$ = the estimated number of correct digits in x_j before iterative improvement was performed

for $j = 1,\ldots,\ell$.

TOL and N1 correspond to the parameters τ and k in the least squares subroutines HFTI and HFTI2. TOL is a nonnegative number that is specified by the user, and N1 is a variable that is set by the routine. When L2SLV is called, modified Gram-Schmidt orthogonalization with column pivoting is used to reduce $\tilde{W}\tilde{A}$ to the form (A_1A_2) where A_1 is an $m \times N_1$ matrix having rank N_1. A_1 is of the form QU where $Q^tQ = \text{diag}(d_1,\ldots,d_{N_1})$ and U is an upper unit triangular matrix. The values $d_1,d_2,\ldots$ correspond to the diagonal elements $c_{11}, c_{22},\ldots$ generated by HFTI and HFTI2 ($d_i = c_{ii}^2$ for $i = 1,2,\ldots$). The values are ordered so that $d_i \geqslant d_{i+1}$, and $d_1,d_2,\ldots$ are stored in $WK(2\ell+1)$, $WK(2\ell+2),\ldots$. If $m = m_e$ then N1 is assigned the value m_e. Otherwise, if $m > m_e$ then $N1 = k - 1$ where k is the smallest integer greater than m_e for which $d_k \leqslant \tau$. Here τ = TOL if TOL > 0, and $\tau = (n\epsilon)^2 d_{m_e+1}$ where ϵ is the smallest value for which $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ on the CDC 6700) if TOL = 0. Thus, if TOL = 0 then a tolerance based on the computer precision is used to determine the rank N_1 of $\tilde{W}\tilde{A}$. Otherwise, if TOL > 0 then TOL is the tolerance that is used to specify the rank of the problem to be solved. If the user inadvertently sets TOL to be negative then L2SLV resets TOL to be 0.

IPIVOT is an array of dimension n or larger that is used by L2SLV to record the order in which the columns of $\tilde{W}\tilde{A}$ are selected by the pivoting procedure when $\tilde{W}\tilde{A}$ is reduced to (A_1A_2). If $N1 < n$ then the first N1 elements of IPIVOT are the indices of the columns of $\tilde{W}\tilde{A}$ from which the matrix A_1 is generated.

T is a 2-dimensional array of dimension $kt \times n$ that is used for temporary storage. It is assumed that $kt \geqslant m + n$. When L2SLV terminates, if $N1 = n$ then the unscaled $n \times n$

covariance matrix is stored in the first n rows and columns of T. Iterative improvement is not performed on the covariance matrix.

Error Return. IERR is an integer variable that is set by the routine. If no input errors are detected and the results appear to be satisfactory, then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	Either m, n, or ℓ is not positive.
IERR = 2	The restriction $0 \leqslant m_e \leqslant \min\{m,n\}$ is not satisfied.
IERR = 3	One of the dimensioning restrictions $ka \geqslant m$, $kb \geqslant m$, $kx \geqslant n$, $kr \geqslant m$, $kt \geqslant m + n$ is violated.
IERR = 4	$\text{WGTS}(i)$ is negative for some $i > m_e$.
IERR = 5	Either $\widetilde{W}\widetilde{A} = 0$ or $E = 0$.
IERR = 6	The rank of E is less than m_e.
IERR = 7	Iterative improvement of all the solutions $x_1,\ldots,x_\ell$ failed to converge.
IERR = 8	Iterative improvement of one or more solutions failed to converge.
IERR = 9	More than μ iterations of the iterative improvement procedure were performed in computing some x_j. (Here it is assumed that a μ decimal digit floating-point arithmetic is being used. $\mu = 14$ for the CDC 6700.)
IERR = 10	The accuracy of some x_j before iterative improvement was estimated to be less than half a decimal digit.
IERR = 11	One or more of the computed diagonal elements of the covariance matrix is negative. This is due to roundoff error. Theoretically, all the diagonal elements should be nonnegative. No evidence of severe ill-conditioning was detected.
IERR = 12	One or more of the computed diagonal elements of the covariance matrix is negative. This is due to roundoff error. Theoretically, all the diagonal elements should be nonnegative. The problem appears to be extremely ill-conditioned.

When an input error is detected (IERR = 1, 2, ..., 6) then L2SLV immediately terminates. If evidence of severe ill-conditioning is detected, then IERR is set to 8, 9, or 10 and computation of the solutions continues. If iterative improvement appears to converge for one or more of the solutions, then the covariance matrix is also computed (when N1 = n). However, if iterative improvement fails for all the solutions $x_1,\ldots,x_\ell$, then IERR is set to 7 and the covariance matrix is not computed.

Note. WK(1),...,WK(2ℓ) should be examined when severe ill-conditioning is detected.

The Covariance Matrix. L2SLV computes the unscaled covariance matrix

$$Q\begin{pmatrix} 0 & 0 \\ 0 & \bar{C} \end{pmatrix}Q^t$$

defined in the section concerning the least squares subroutine LSEI.

Programming. L2SLV employs the subroutines DECOM2, SOLVE2, SOLVE3, and COVAR. These routines were written by Roy Wampler (National Bureau of Standards). L2SLV is a slightly modified version by A. H. Morris of the subroutine L2B discussed in reference (4). The algorithm employed for finding and iteratively improving the least squares solutions is described in references (1)–(3). The function SPMPAR is also used.

References

(1) Bjorck, Ake, "Solving Linear Least Squares Problems by Gram-Schmidt Orthogonalization," ***BIT*** *7* (1967), pp. 1–21.

(2) ____________, "Iterative Refinement of Linear Least Squares Solutions I," ***BIT*** *7* (1967), pp. 257–278.

(3) ____________, "Iterative Refinement of Linear Least Squares Solutions II," ***BIT*** *8* (1968), pp. 8–30.

(4) Wampler, Roy, "Solutions to Weighted Least Squares Problems by Modified Gram-Schmidt with Iterative Refinement," ***ACM Trans. Math Software*** *5* (1979), pp. 457–465.

ITERATIVE LEAST SQUARES SOLUTION OF BANDED LINEAR EQUATIONS

Given an m $\times$ n matrix A, a column vector b of dimension m, and a real number λ. Let $\bar{A} = \binom{A}{\lambda I}$ where I is the n $\times$ n identity matrix, and let $\bar{b} = \binom{b}{0}$. The problem is to find a column vector x of dimension n which is a least squares solution of $\bar{A}x = \bar{b}$ If A is stored in band form then the following subroutine is available for solving this problem.

<u>CALL BLSQ(m,n,A,ka,m_ℓ,m_u,λ,b,x,ATOL,BTOL,CONLIM,MXITER, IND,ITER,COND,RNORM,XNORM,WK)</u>

A is an m $\times$ n matrix stored in band form, m_ℓ the number of diagonals below the main diagonal containing nonzero elements, and m_u the number of diagonals above the main diagonal containing nonzero elements. The argument ka is the number of rows in the dimension statement for A in the calling program. It is assumed that $0 \leqslant m_\ell < m$, $0 \leqslant m_u < n$, and ka $\geqslant$ m. When BLSQ is called, an iterative procedure is used to obtain a least squares solution x of $\bar{A}x = \bar{b}$. The vector b is modified by the routine.

ATOL and BTOL are input arguments which specify the relative accuracy of A and b respectively. For example, if it is estimated that b is accurate to k decimal digits then one may set BTOL = 10^{-k}. It is required that ATOL $\geqslant 0$ and BTOL $\geqslant 0$. If ATOL = 0 or BTOL = 0, then it is assumed that A or b is accurate to machine precision.

Let cond($\bar{A}$) denote the condition number of $\bar{A}$ relative to the Frobenius norm.[1] In each iteration of the algorithm being used, an estimate is made of the condition number cond($\bar{A}$). The estimates form a monotonically nondecreasing sequence. The input argument CONLIM is an upper limit on cond($\bar{A}$). If CONLIM > 0 then BLSQ terminates when an estimate of cond($\bar{A}$) exceeds CONLIM. This termination may be needed to prevent small or zero singular values of A from coming into effect and causing damage to the solution x. CONLIM may be ignored by being set to 0. It is assumed that CONLIM $\geqslant 0$.

The input argument MXITER is the maximum number of iterations that are permitted. Normally BLSQ requires less than 4n iterations. The related argument ITER is a variable. When the routine terminates ITER = the number of iterations that were performed.

COND, RNORM, and XNORM are variables. When BLSQ terminates COND = the last estimate made for cond ($\bar{A}$), RNORM = $\|\bar{A}x - \bar{b}\|$, and XNORM = $\|x\|$.[2]

[1] cond $(\bar{A}) = \|\bar{A}\|_F \, \|\bar{A}^+\|_F$ where $\bar{A}^+$ is the pseudoinverse of $\bar{A}$. Here $\|C\|_F = \sqrt{\Sigma c_{ij}^2}$ for any matrix $C = (c_{ij})$.
[2] $\|c\| = \sqrt{\Sigma c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.

The equations $\bar{A}x - \bar{b}$ are considered to be ***compatible*** if for any least squares solution x, $\|\bar{A}x - \bar{b}\| = 0$. IND is a variable that reports the status of the results. When BLSQ terminates, IND has one of the following values:

IND = 0	The solution is x = 0. No iterations were performed.
IND = 1	The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is sufficiently accurate, given the values ATOL and BTOL.
IND = 2	The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is sufficiently accurate, given the value ATOL.
IND = 3	An estimate COND of cond($\bar{A}$) exceeds CONLIM. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.
IND = 4	The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is as accurate as seems reasonable on this machine.
IND = 5	The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is as accurate as seems reasonable on this machine.
IND = 6	cond($\bar{A}$) appears to be so large that there is not much point in doing further iterations. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.
IND = 7	MXITER iterations were performed. More iterations are needed. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

Remarks

(1) A large estimate of the condition number cond($\bar{A}$) may be due to rank deficiency or near rank deficiency of the matrix $\bar{A}$. If it is suspected that a large estimate of cond($\bar{A}$) has occurred for this reason, then it is recommended that CONLIM be set to a moderate value such as $\epsilon^{-1/2}$ where ϵ is the smallest value such that $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6000-7000 series computers). Setting CONLIM to 0 is equivalent to setting CONLIM to ϵ^{-1}.

(2) The vector b is the only input argument modified by the routine.

Algorithm. BLSQ employs an iteration algorithm developed by Golub and Kahan.

Programming. BLSQ calls the subroutines NORMLZ, BVPRD1, BTPRD1, SCOPY, and SSCAL. The function SNRM2 is also used. BSLQ is an adaptation by A. H. Morris of the subroutine LSQR, written by Christopher C. Paige (McGill University, Montreal, Canada) and Michael A. Saunders (Stanford University).

References

(1) Paige, C. C. and Saunders, M. A., "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Trans. Math Software 8* (1982), pp. 43-71.

(2) ——————, "Algorithm 583. LSQR: Sparse Linear Equations and Least Squares Problems," *ACM Trans. Math Software 8* (1982), pp. 195-209.

ITERATIVE LEAST SQUARES SOLUTION OF SPARSE LINEAR EQUATIONS

Given an m $\times$ n matrix A, a column vector b of dimension m, and a real number λ. Let $\bar{A} = \binom{A}{\lambda I}$ where I is the n $\times$ n identity matrix, and let $\bar{b} = \binom{b}{0}$. The problem is to find a column vector x of dimension n which is a least squares solution of $\bar{A}x = \bar{b}$. If A is sparse then the following subroutines are available for solving this problem.

CALL SPLSQ(m,n,A,IA,JA,λ,b,x,ATOL,BTOL,CONLIM,MXITER,
IND,ITER,COND,RNORM,XNORM,WK)

CALL STLSQ(m,n,TA,ITA,JTA,λ,b,x,ATOL,BTOL,CONLIM,MXITER,
IND,ITER,COND,RNORM,XNORM,WK)

If SPLSQ is called then A, IA, JA are arrays containing the matrix A in sparse form. Otherwise, if STLSQ is called then TA, ITA, JTA are arrays containing the transpose matrix A^t in sparse form. An iterative procedure is used to obtain a least squares solution x of $\bar{A}x = \bar{b}$. The vector b is modified by the routines.

ATOL and BTOL are input arguments which specify the relative accuracy of A and b respectively. For example, if it is estimated that b is accurate to k decimal digits then one may set BTOL = 10^{-k}. It is required that ATOL $\geqslant 0$ and BTOL $\geqslant 0$. If ATOL = 0 or BTOL = 0, then it is assumed that A or b is accurate to machine precision.

Let cond($\bar{A}$) denote the condition number of $\bar{A}$ relative to the Frobenius norm.[1] In each iteration of the algorithm being used, an estimate is made of the condition number cond($\bar{A}$). The estimates form a monotonically nondecreasing sequence. The input argument CONLIM is an upper limit on cond($\bar{A}$). If CONLIM > 0 then the routines terminate when an estimate of cond($\bar{A}$) exceeds CONLIM. This termination may be needed to prevent small or zero singular values of $\bar{A}$ from coming into effect and causing damage to the solution x. CONLIM may be ignored by being set to 0. It is assumed that CONLIM $\geqslant 0$.

The input argument MXITER is the maximum number of iterations that are permitted. Normally the routines require less than 4n iterations. The related argument ITER is a variable. When the routines terminate ITER = the number of iterations that were performed.

COND, RNORM, and XNORM are variables. When the routines terminate COND = the last estimate made for cond ($\bar{A}$), RNORM = $\|\bar{A}x - \bar{b}\|$, and XNORM = $\|x\|$.[2]

[1] cond $(\bar{A}) = \|\bar{A}\|_F \, \|\bar{A}^+\|_F$ where $\bar{A}^+$ is the pseudoinverse of $\bar{A}$. Here $\|C\|_F = \sqrt{\Sigma c_{ij}^2}$ for any matrix $C = (c_{ij})$.
[2] $\|c\| = \sqrt{\Sigma c_i^2}$ for any vector $c = (c_1, c_2, \ldots)$.

The equations $\bar{A}x - \bar{b}$ are considered to be ***compatible*** if for any least squares solution x, $\|\bar{A}x - \bar{b}\| = 0$. IND is a variable that reports the status of the results. When the routines terminate, IND has one of the following values:

IND = 0	The solution is x = 0. No iterations were performed.
IND = 1	The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is sufficiently accurate, given the values ATOL and BTOL.
IND = 2	The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is sufficiently accurate, given the value ATOL.
IND = 3	An estimate COND of cond($\bar{A}$) exceeds CONLIM. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.
IND = 4	The equations $\bar{A}x = \bar{b}$ are probably compatible. A solution x has been obtained which is as accurate as seems reasonable on this machine.
IND = 5	The equations $\bar{A}x = \bar{b}$ are probably not compatible. A least squares solution x has been obtained which is as accurate as seems reasonable on this machine.
IND = 6	cond($\bar{A}$) appears to be so large that there is not much point in doing further iterations. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.
IND = 7	MXITER iterations were performed. More iterations are needed. The vector x is the most recent approximation of a solution for $\bar{A}x = \bar{b}$.

Remarks

(1) A large estimate of the condition number cond($\bar{A}$) may be due to rank deficiency or near rank deficiency of the matrix $\bar{A}$. If it is suspected that a large estimate of cond($\bar{A}$) has occurred for this reason, then it is recommended that CONLIM be set to a moderate value such as $\epsilon^{-1/2}$ where ϵ is the smallest value such that $1 + \epsilon > 1$ ($\epsilon = 2^{-47}$ for the CDC 6000-7000 series computers). Setting CONLIM to 0 is equivalent to setting CONLIM to ϵ^{-1}.

(2) The vector b is the only input argument modified by the routine.

Algorithm. SPLSQ and STLSQ employ an iterative algorithm developed by Golub and Kahan.

Programming. SPLSQ and STLSQ call the subroutines NORMLZ, MVPRD1, MTPRD1, SCOPY, and SSCAL. The function SNRM2 is also used. SPLSQ and STLSQ are adaptations by A. H. Morris of the subroutine LSQR, written by Christopher C. Paige (McGill University, Montreal, Canada) and Michael A. Saunders (Stanford University).

References

(1) Paige, C. C. and Saunders, M. A., "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *ACM Trans. Math Software 8* (1982), pp. 43-71.

(2) ————————, "Algorithm 583. LSQR: Sparse Linear Equations and Least Squares Problems," *ACM Trans. Math Software 8* (1982), pp. 195-209.

MINIMIZATION OF FUNCTIONS OF A SINGLE VARIABLE

Let F(x) be a continuous real-valued function defined for $a \leqslant x \leqslant b$. Then the following subroutine is available for finding a local minimum of F(x).

CALL FMIN(F,a,b,x,w,AERR,RERR,ERROR,IND)

It is assumed that $a \leqslant b$. FMIN finds a value x in the interval [a,b] which is a local minimum of F. ERROR and w are variables. When FMIN terminates, w = F(x) and ERROR is the estimated maximum absolute error of x.

The input arguments AERR and RERR are the absolute and relative error tolerances to be satisfied. For example, if k significant digit accuracy is desired then one may set RERR = 10^{-k}. It is assumed that AERR $\geqslant 0$ and RERR $\geqslant 0$. The setting AERR = 0 is equivalent to the setting AERR = 10^{-20}, and the setting RERR = 0 is a request for machine precision.

IND is a variable that reports the status of the results. When FMIN terminates, IND = 0 if x was found to the desired accuracy. Otherwise, IND = 1 when the value obtained for x could not be refined to the desired accuracy. In this case, w satisfies the tolerances AERR and RERR.

Note. F must be declared in the calling program to be of type EXTERNAL.

Algorithm. The golden section search procedure is used.

Programming. The function SPMPAR is called. FMIN was written by A. H. Morris.

UNCONSTRAINED MINIMUM OF THE SUM OF SQUARES OF NONLINEAR FUNCTIONS

Let $f_1(x),\dots,f_m(x)$ be m real-valued functions of n real variables $x = (x_1,\dots,x_n)$ where $m \geqslant n$. The problem under consideration is to find a point x which minimizes the function $\phi(x) = \sum_{i=1}^{m} f_i(x)^2$. Assume that each $f_i(x)$ is differentiable and that an initial guess $a = (a_1,\dots,a_n)$ to a minimum of $\phi(x)$ is given. Then the following subroutine is available for finding a point which minimizes $\phi(x)$.

CALL LMDIFF(F,m,n,X,FVEC,EPS,TOL,INFO,IWK,WK,ℓ)

X is an array of dimension n and FVEC is an array of dimension m. On input X contains the starting point $a = (a_1,\dots,a_n)$. When LMDIFF terminates, X contains the final estimate $x = (x_1,\dots,x_n)$ of a minimum of ϕ and FVEC contains the values of the functions $f_1,\dots,f_m$ at the output point in X.

The argument F is the name of a user defined subroutine that has the format:

CALL F(m,n,X,FVEC,IFLAG)

Here X is an array of dimension n, FVEC is an array of dimension m, and IFLAG is an integer variable. The array X contains a point $x = (x_1,\dots,x_n)$. Normally F will evaluate the functions $f_1,\dots,f_m$ at this point and store the results in FVEC. However, if x does not lie in the domain of $f_1,\dots,f_m$ then this cannot be done. In this case, the argument IFLAG (which will have been assigned a nonnegative value by LMDIFF) should be reset by F to a negative value. This will signal LMDIFF to terminate. F must be declared in the calling program to be of type EXTERNAL.

EPS is an input argument which specifies the relative accuracy of F. If it is estimated that the subroutine F produces results accurate to k significant decimal digits then one may set EPS = 10^{-k}. It is required that EPS $\geqslant 0$. If EPS = 0 then it is assumed that F produces results accurate to machine precision.

TOL is an input argument which specifies the desired accuracy to be attained. The Euclidean norm $\|x\| = \sqrt{\Sigma_i x_i^2}$ is employed. If $\bar{x}$ denotes an actual minimum of ϕ, then LMDIFF terminates when an iterate x is generated for which it is estimated that

(1) $\phi(x) \leqslant (1 + \mathrm{TOL})^2 \phi(\bar{x})$ or

(2) $\|D(x - \bar{x})\| \leqslant \mathrm{TOL} \cdot \|D\bar{x}\|$

is satisfied. In (2) x and $\bar{x}$ are regarded as column vectors, and D is a diagonal matrix generated by LMDIFF whose entries are scaling factors. For convenience, criterion (1) is called the F-convergence (or ϕ-convergence) test and criterion (2) is called the x-convergence test. It is required that TOL $\geqslant 0$. In order for the convergence tests to work properly, it is recommended that TOL always be smaller than 10^{-5}.

IWK is an array of dimension n and WK is an array of dimension ℓ. IWK and WK are work spaces. It is assumed that the argument ℓ is greater than or equal to $mn + 5n + m$.

INFO is an integer variable that reports the status of the results. When LMDIFF terminates, INFO will have one of the following values:

INFO < 0	This occurs when the user terminates the execution of LMDIFF by resetting the argument IFLAG in the subroutine F to a negative value. Then INFO = the negative value of IFLAG.
INFO = 0	(Input Error) Either $1 \leqslant n \leqslant m$, EPS $\geqslant 0$, TOL $\geqslant 0$, or $\ell \geqslant mn + 5n + m$ is violated.
INFO = 1	The F-convergence test has been satisfied.
INFO = 2	The x-convergence test has been satisfied.
INFO = 3	Both the F-convergence and x-convergence tests have been satisfied.
INFO = 4	The gradient of ϕ is 0 at point X.
INFO = 5	The number of calls to the subroutine F has reached or exceeded $200(n + 1)$.
INFO = 6	TOL is too small. No further reduction in the value of $\phi(x)$ is possible.
INFO = 7	TOL is too small. No further improvement in the accuracy of X is possible.

When LMDIFF terminates, if INFO $\neq 0$ then X contains the final iterate that was generated. Also, if INFO $\geqslant 1$ then FVEC contains the values of the functions $f_1,\dots,f_m$ at this iterate. If INFO = 4 then X should be examined very closely. The gradient of ϕ can be 0 when X is a local minimum or maximum, or when X is a saddle point. If INFO = 5 then it may (or may not) be helpful to continue the procedure by recalling LMDIFF with the current point in X as the new starting point. Since TOL is a relative tolerance, this setting can occur when $\phi(0) = 0$.

Algorithm. A modified form of the Levenberg-Marquardt algorithm is employed.

Programming. LMDIFF is a slightly modified version of the MINPACK-1 subroutine LMDIF1. The MINPACK-1 subroutines LMDIF, SPMPAR, ENORM, FDJAC2, LMPAR, QRFAC, and QRSOLV are employed. The subroutines were written by Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom (Argonne National Laboratory).

References

(1) More, J. J., Garbow, B. S., and Hillstrom, K. E., ***User Guide for MINPACK-1***, Argonne National Laboratory Report ANL-80-74, Argonne, Illinois, 1980.

(2) More, J. J., "The Levenberg-Marquardt Algorithm: Implementation and Theory," ***Numerical Analysis***, G. A. Watson (ed.), Springer-Verlag, 1977.

LINEAR PROGRAMMING

Let $A = (a_{ij})$ be an $m \times n$ matrix, B an array containing $b_1,\dots,b_m$, and C an array containing $c_1,\dots,c_n$ where a_{ij}, b_i, c_j are real. Consider the problem of finding nonnegative values $x_1,\dots,x_n$ which maximize or minimize the function $c_1x_1 + \cdots + c_nx_n$ subject to the constraints:

$$\begin{gathered} a_{11}x_1 + \cdots + a_{1n}x_n \{\leqslant, =, \geqslant\} b_1 \\ \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \{\leqslant, =, \geqslant\} b_m \end{gathered}$$

In each constraint

$$a_{i1}x_1 + \cdots + a_{in}x_n \{\leqslant, =, \geqslant\} b_i$$

only one of the relations $\leqslant, =, \geqslant$ is used, but the relation may vary from constraint to constraint. The following subroutines are available for solving this problem.

CALL SMPLX(A,B,C,ka,m,n,IND,IBASIS,X,z,ITER,MXITER,
NUMLE,NUMGE,BI,WK,IWK)
CALL SSPLX(TA,ITA,JTA,B,C,m,n,IND,IBASIS,X,z,ITER,MXITER,
NUMLE,NUMGE,BI,WK,IWK)

It is assumed that $m \geqslant 2$, $n \geqslant 2$, and that each $b_i \geqslant 0$. If SMPLX is called then ka is the number of rows in the dimension statement for A in the calling program. Otherwise, if SSPLX is called then TA,ITA,JTA are arrays containing the transpose matrix A^t in sparse form.

The constraints $a_{i1}x_1 + \cdots + a_{in}x_n \{\leqslant, =, \geqslant\} b_i$ are assumed to be ordered so that the $\leqslant$ constraints are followed by the $\geqslant$ constraints, and the = constraints come last. NUMLE and NUMGE have the values:

NUMLE = the number of $\leqslant$ constraints
NUMGE = the number of $\geqslant$ constraints

NUMLE and NUMGE must satisfy NUMLE $\geqslant 0$, NUMGE $\geqslant 0$, and NUMLE + NUMGE $\leqslant m$.

When SMPLX or SSPLX is called, the routine attempts to maximize $\Sigma_j c_j x_j$ subject to the constraints. A modified form of the primal simplex algorithm is employed. Frequently the procedure requires less than 5m iterations to perform the task. The argument MXITER has the value:

MXITER = the maximum number of iterations that may be performed

This argument is provided by the user. The related argument ITER is a variable that is set by the routine. When the routine terminates, ITER has for its value the number of iterations that were performed.

IND is a variable and IBASIS an array of dimension m. IBASIS contains the indices i of the current basic variables x_i, and IND is used for input/output purposes. On input IND is normally set by the user to 0. If IND = 0 then the routine selects its own beginning basis and stores the appropriate indices in IBASIS. [The remainder of this paragraph may be skipped by anyone not acquainted with the simplex algorithm.] If the user wishes to use his own beginning basis, then IND must be set to 1 and the indices of the initial basic variables stored in IBASIS. ***It is not required that the initial basis be selected so that the basic variables are nonnegative.*** The initial basic variables may be original, slack, surplus, or artificial variables. Slack and surplus variables are automatically provided for the $\leqslant$ and $\geqslant$ constraints, and artificial variables for the = constraints. The routine defines x_{n+i} to be the slack, surplus, or artificial variable for the ith constraint $a_{i1}x_1 + \cdots + a_{in}x_n\{\leqslant,=,\geqslant\}b_i$. If IND = 0 then the slack, surplus, and artificial variables are the initial basic variables that are employed.

On output IND reports the status of the results. The routine assigns IND one of the following values:

IND = 0	The problem was solved.
IND = 1	The problem has no solution.
IND = 2	MXITER iterations were performed. More iterations are needed.
IND = 3	Sufficient accuracy cannot be maintained to solve the problem.
IND = 4	The problem has an unbounded solution.
IND = 5	An input error was detected. (See below)
IND = 6	A possible solution was obtained. The routine is not certain if the solution is correct.

X is an array of dimension n + NUMLE + NUMGE and z is a variable. If IND = 0 or IND = 6, then z has for its value the maximum value obtained for $\Sigma_j c_j x_j$ and X contains the values obtained for the original, slack, and surplus variables. If IND $\neq$ 5 then IBASIS contains the indices of the basic variables currently in effect when the routine terminates.

BI is an array of dimension m^2 that is used for storing the inverse of the basis matrix. The order of the column vectors of the basis matrix corresponds to the order of the basic variables given in IBASIS. If IND $\neq$ 3,5 on output then BI contains the inverse of the basis matrix currently in effect when the routine terminates.

WK is an array of dimension 2m or larger, and IWK is an array of dimension 2m + n or larger. WK and IWK are work spaces.

Input Errors. IND = 5 occurs on output when one of the following conditions is violated:
(1) $n \geqslant 2$ and $ka \geqslant m \geqslant 2$
(2) NUMLE + NUMGE $\leqslant$ m
(3) Each $b_i \geqslant 0$.
(4) Tne basis matrix specified by the user in IBASIS (when IND = 1 on input) is nonsingular and sufficiently well conditioned so that its inverse can be computed.

Remarks

(1) A,B,C,TA,ITA,JTA are not modified by the routines.

(2) The routines maximize $\Sigma_j c_j x_j$. This function can be minimized by maximizing $\Sigma_j(-c_j)x_j$ and then changing the sign of the result.

(3) SMPLX and SSPLX generate the same results. For efficiency, SSPLX should be used when A is sparse.

Algorithm. A three step procedure is used. The first step eliminates the negative variables. Then phases (1) and (2) of the primal simplex algorithm are invoked. Negative variables are eliminated as follows: Let $x_{B1},\ldots,x_{Bm}$ be the basic variables and y_{ij} the components of the simplex tableau.

(1) Compute $d_j = \Sigma_i' y_{ij}$ for each nonbasic variable x_j where the sum Σ_i' is for all i where $x_{Bi} < 0$. If all $d_j \geqslant 0$ then the problem has no feasible solution. Otherwise, select k so that $d_k = \min_j d_j$. Then x_k is the variable to be made basic.

(2) If $x_{Bj} \geqslant 0$ and $y_{jk} > 0$ for some j then go to (3). Otherwise, select a negative variable x_{Br} to become nonbasic where $x_{Br}/y_{rk} = \max\{x_{Bj}/y_{jk} : x_{Bj} < 0 \text{ and } y_{jk} < 0\}$. Then update the basis and go to (5).

(3) Compute $\epsilon = \min\{x_{Bj}/y_{jk} : x_{Bj} \geqslant 0 \text{ and } y_{jk} > 0\}$ and check if a negative variable x_{Bj} exists that satisfies the conditions:

$(*)\quad y_{jk} < 0 \text{ and } \epsilon \geqslant x_{Bj}/y_{jk}$

If such a variable exists then go to (4). Otherwise, select a nonnegative variable x_{Br} to become nonbasic where $y_{rk} > 0$ and $x_{Br}/y_{rk} = \epsilon$. Then update the basis and go to (1).

(4) Select a negative variable x_{Br} to become nonbasic where $x_{Br}/y_{rk} = \max\{x_{Bj}/y_{jk} : x_{Bj} < 0$ and x_{Bj} satisfies $(*)\}$. Then update the basis and go to (5).

(5) Check if there are any remaining negative variables. If not, then we are finished. Otherwise go to (1).

Programming. SMPLX and SSPLX employ the subroutines SMPLX1, SSPLX1, and CROUT1. These routines were written by A. H. Morris. The function SPMPAR is also used.

Reference. Cooper, Leon and Steinberg, David, ***Methods and Applications of Linear Programming,*** W. B. Saunders Co., Philadelphia, 1974.

THE ASSIGNMENT PROBLEM

Let $C = (c_{ij})$ be an $n \times n$ matrix (the cost matrix). The problem under consideration is to find an $n \times n$ matrix $x = (x_{ij})$ which minimizes $T = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$ and satisfies:

(1) $\sum_{i=1}^{n} x_{ij} = 1$ for $j = 1,\dots,n$

(2) $\sum_{j=1}^{n} x_{ij} = 1$ for $i = 1,\dots,n$

(3) Each $x_{ij} = 0$ or 1

Each x which satisfies (1)-(3) is called an ***assignment.*** For each such x, from (1) and (3) we note that for each j there exists a unique integer $\pi(j)$ such that $x_{\pi(j),j} = 1$. Also, (2) and (3) assert that π is a permutation of $\{1,\dots,n\}$. Conversely, for any permutation π there corresponds an assignment x defined by $x_{\pi(j),j} = 1$ and $x_{ij} = 0$ for $i \neq \pi(j)$. Thus, the problem is to find a permutation π of $\{1,\dots,n\}$ which minimizes $T = \sum_{j=1}^{n} c_{\pi(j),j}$. The following subroutine is available for solving this problem when all c_{ij} are integers.

CALL ASSGN(n,C,JC,T,IWK,IERR)

C is a 2-dimensional integer array of dimension $n \times (n+1)$, JC an integer array of dimension n, and T an integer variable. It is assumed that $n \geqslant 2$ and that the first n columns of C contain the cost matrix (c_{ij}). [The $(n+1)^{st}$ column of C is a work space for the routine.] When ASSGN is called, the desired permutation π is obtained and the values $\pi(1),\dots,\pi(n)$ stored in JC. Also T is assigned the minimized value $\Sigma_j\, c_{\pi(j),j}$.

IWK is an array of dimension 7n + 2 or larger that is a work space for the routine.

IERR is a variable that is set by the routine. If JC and T are obtained then IERR is assigned the value 0. Otherwise, if the problem cannot be solved because of integer overflow, then IERR = 1.

Remarks

(1) C is destroyed by the routine.

(2) ASSGN minimizes $T = \Sigma_j c_{\pi(j),j}$. This function can be maximized by minimizing $\Sigma_j(-c_{\pi(j),j})$ and then changing the sign of the result.

Programming. ASSGN calls the subroutine ASSGN1. ASSGN1 was written by Giorgio Carpaneto and Paolo Toth (University of Bologna, Italy), and modified by A. H. Morris. The function I1MACH is also used.

Reference. Carpaneto, G., and Toth, P., "Algorithm 548, Solution of the Assignment Problem," ***ACM Trans. Math Software 6*** (1980), pp. 104-111.

FAST FOURIER TRANSFORM

Let n be a positive integer and $\theta_j = 2\pi j/n$ for $j = 0, 1, \ldots, n-1$. For any complex-valued functions f and g defined on the points θ_j let $(f,g) = \sum_{j=0}^{n-1} f(\theta_j)\overline{g(\theta_j)}$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_j. Also $e^{ij\theta}$ $(j = 0, 1, \ldots, n-1)$ form an orthogonal set of functions where each $e^{ij\theta}$ has norm $\sqrt{n}$.[1] Thus, if f is a function that is approximated by $f(\theta) = \sum_{j=0}^{n-1} c_j e^{ij\theta}$ then each $c_j = \frac{1}{n}(f(\theta), e^{ij\theta})$. The mapping $f(\theta_j) \rightarrow c_j$ given by

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} f(\theta_k)\, e^{-2\pi ijk/n}$$

is called the ***discrete Fourier transform*** and its inverse

$$f(\theta_j) = \sum_{k=0}^{n-1} c_k\, e^{2\pi ijk/n}$$

the ***inverse discrete Fourier transform***. The following subroutines are available for computing these transforms.

CALL FFT(C,n,ℓ,IERR)
CALL FFT1(A,B,n,ℓ,IERR)

Let $c_j = a_j + ib_j$ $(j = 0, 1, \ldots, n-1)$ be the data to be transformed. If FFT is called then C is a complex array containing $c_0, c_1, \ldots, c_{n-1}$ (where $C(j+1) = c_j$ for $j < n$). Otherwise, if FFT1 is called then A and B are real arrays containing $a_0, a_1, \ldots, a_{n-1}$ and $b_0, b_1, \ldots, b_{n-1}$ respectively.

The argument ℓ may have the values 1 or −1, and IERR is a variable. When FFT or FFT1 is called, if there are no input errors then IERR is set to 0 and

$$\hat{c}_j = \sum_{k=0}^{n-1} c_k \exp(2\pi \ell ijk/n)$$

is computed. The results $\hat{c}_j = \hat{a}_j + i\hat{b}_j$ replace the original data $c_j = a_j + ib_j$ in C (or A and B).

Restrictions on the argument n. When FFT and FFT1 are called, n is factored by the routine into its prime factors. It is assumed that the largest prime factor of n is $\leqslant 23$. If $n = \mu^2 \bar{n}$ where $\bar{n}$ is the square free portion of n, then it is further assumed that $\bar{n} \leqslant 210$ whenever $\bar{n}$ is a product of two or more primes.

[1] Throughout this section $i = \sqrt{-1}$.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1	if $n < 1$
IERR = 2	if n has too many factors
IERR = 3	if n has a prime factor greater than 23 or the square free portion of n is greater than 210
IERR = 4	if $\ell \neq \pm 1$

The setting IERR = 2 can occur only when $n > 4251528$.

Remark. The complex array C is interpreted by FFT as a real array of dimension 2n. If this association is not permitted by the FORTRAN being employed then use FFT1.

Programming. FFT and FFT1 are interface routines for the subroutine SFFT, which was written by Richard C. Singleton (Stanford Research Institute).

Reference. Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," ***IEEE Trans. Audio and Electroacoustics***, vol. AU-17 (1969), pp. 93-103.

MULTIVARIATE FAST FOURIER TRANSFORM

Let $n_1,\ldots,n_m$ be positive integers, and for any $J = (j_1,\ldots,j_m)$ where $j_\nu = 0, 1,\ldots,n_\nu - 1$ $(\nu = 1,\ldots,m)$ let θ_J denote the point $(2\pi j_1/n_1,\ldots,2\pi j_m/n_m)$. Also, for any complex-valued functions f and g defined on the points θ_J let $(f,g) = \Sigma_J\, f(\theta_J)\,\overline{g(\theta_J)}$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_J. Also the functions $\phi_J(\theta) = \exp(ij_1\theta^1)\cdots\exp(ij_m\theta^m)$ form an orthogonal set where each ϕ_J has the norm $\sqrt{n_1\cdots n_m}$.[1] Thus, if f is a function that is approximated by $f = \Sigma_J\, c_J\, \phi_J$ then each $c_J = \dfrac{1}{n_1\cdots n_m}(f,\phi_J)$. The mapping $f(\theta_J) \to c_J$ given by

$$c_J = \frac{1}{n_1\cdots n_m}\ \Sigma_K\, f(\theta_K)\exp(-2\pi i\, j_1 k_1/n_1)\cdots\exp(-2\pi i\, j_m k_m/n_m)$$

is called the ***discrete multivariate Fourier transform*** and its inverse

$$f(\theta_J) = \Sigma_K c_K\, \exp(2\pi i\, j_1 k_1/n_1)\cdots \exp(2\pi i\, j_m k_m/n_m)$$

the ***inverse discrete multivariate Fourier transform.*** The sums Σ_K are for all $K = (k_1,\ldots,k_m)$ where $k_\nu = 0, 1,\ldots,n_\nu - 1$ $(\nu = 1,\ldots,m)$. The following subroutines are available for computing these transforms.

CALL MFFT(C,N,m,ℓ,IERR)
CALL MFFT1(A,B,N,m,ℓ,IERR)

Let $c_J = a_J + ib_J$ be the data to be transformed where $J = (j_1,\ldots,j_m)$ for $j_\nu = 0, 1,\ldots,n_\nu - 1$ $(\nu = 1,\ldots,m)$. If MFFT is called then C is a 1-dimensional complex array containing the values c_J (where $c_J = C(1 + j_1 + j_2 n_1 + j_3 n_1 n_2 + \cdots + j_m n_1\cdots n_{m-1})$). Otherwise, if MFFT1 is called then A and B are 1-dimensional real arrays containing the data a_J and b_J respectively.

Note. If MFFT is used and m = 2 or 3, then instead of having to store the m-dimensional data c_J into a 1-dimensional array C, the data may be stored in C where C is defined to be an m-dimensional array. If m = 2 then C may be declared to be of dimension $n_1 \times n_2$, in which case $C(j_1 + 1,\ j_2 + 1) = c_J$ for all $J = (j_1, j_2)$. Similarly, if m = 3 then C may be declared to be of dimension $n_1 \times n_2 \times n_3$, in which case $C(j_1 + 1,\ j_2 + 1,\ j_3 + 1) = c_J$ for all $J = (j_1, j_2, j_3)$. Similar comments hold for A and B if MFFT1 is employed.

N is an array containing the integers $n_1,\ldots,n_m$. The argument ℓ may have the values 1 and −1, and IERR is a variable. When MFFT or MFFT1 is called, if there are no input errors then IERR is set to 0 and the transform

[1] Throughout this section $i = \sqrt{-1}$ and $\theta = (\theta^1,\ldots,\theta^m)$ denotes an arbitrary point.

$$\hat{c}_J = \Sigma_K \, c_K \exp(2\pi \ell i j_1 k_1/n_1) \cdots \exp(2\pi \ell i j_m k_m/n_m)$$

is computed. The results $\hat{c}_J = \hat{a}_J + i\hat{b}_J$ replace the original data $c_J = a_J + ib_J$ in C (or A and B).

Restrictions on the arguments $n_1,\ldots,n_m$. When MFFT and MFFT1 are called, each n_ν is factored by the routine into its prime factors. It is assumed that the largest prime factor of n_ν is $\leqslant 23$. If $\bar{n}_\nu = \mu_\nu^2 \bar{n}_\nu$ where $\bar{n}_\nu$ is the square free portion of n_ν, then it is further assumed that $\bar{n}_\nu \leqslant 210$ whenever $\bar{n}_\nu$ is a product of two or more primes.

Error Return. If an input error is detected then IERR is set as follows:

IERR = 1	if some $n_\nu < 1$
IERR = 2	if some n_ν has too many factors
IERR = 3	if some n_ν has a prime factor greater than 23 or the square free portion of some n_ν is greater than 210
IERR = 4	if $\ell \neq \pm 1$
IERR = 5	if $m \leqslant 0$

The setting IERR = 2 can occur only when some $n_\nu > 4251528$.

Remark. The complex array C of dimension $n_1 \cdots n_m$ is interpreted by MFFT as a real array of dimension $2n_1 \cdots n_m$. If this association is not permitted by the FORTRAN being employed then use MFFT1.

Programming. MFFT and MFFT1 are interface routines for the subroutine SFFT, which was written by Richard C. Singleton (Stanford Research Institute).

Reference. Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," ***IEEE Trans. Audio and Electroacoustics***, vol. AU–17 (1969), pp 93-103.

DISCRETE COSINE AND SINE TRANSFORMS

Let n be a positive integer and $\theta_\nu = (\nu + 1/2)\pi/n$ for $\nu = 0,1,\ldots,n-1$. For any real-valued functions f and g defined on the points θ_ν let $(f,g) = \sum_{\nu=0}^{n-1} f(\theta_\nu)g(\theta_\nu)$. Then (f,g) is an inner product when f and g are regarded as functions defined only on θ_ν. Also $\cos j\theta$ $(j = 0,1,\ldots,n-1)$ form an orthogonal set of functions where $\cos j\theta$ has norm $\sqrt{n}$ when $j = 0$ and norm $\sqrt{n/2}$ when $j \geqslant 1$. Thus, if f is a function that is approximated by $f(\theta) = a_0 + 2\sum_{j=1}^{n-1} a_j \cos j\theta$ then each $a_j = \frac{1}{n}(f(\theta), \cos j\theta)$. The mapping $f(\theta_\nu) \to a_j$ is called the ***discrete cosine transform*** and its inverse $a_j \to f(\theta_\nu)$ the ***inverse discrete cosine transform.***

Alternatively, the functions $\sin j\theta$ $(j = 1,\ldots,n)$ also form an orthogonal set where $\sin j\theta$ has norm $\sqrt{n/2}$ when $j < n$ and norm $\sqrt{n}$ when $j = n$. Thus, if f is a function that is approximated by $f(\theta) = 2\sum_{j=1}^{n-1} b_j \sin j\theta + b_n \sin n\theta$ then each $b_j = \frac{1}{n}(f(\theta), \sin j\theta)$. The mapping $f(\theta_\nu) \to b_j$ is called the ***discrete sine transform*** and its inverse $b_j \to f(\theta_\nu)$ the ***inverse discrete sine transform.***

The subroutines COSQB and COSQF are available for computing the discrete cosine transform and its inverse, and the subroutines SINQB and SINQF are available for computing the discrete sine transform and its inverse. The subroutine COSQI provides information that is needed for the cosine and sine transform routines.

CALL COSQI(n,WK)

WK is an array of dimension $3n + 15$ or larger that is a work space for the routines COSQB, COSQF, SINQB, and SINQF. COSQI stores in WK information needed for the Fast Fourier computation of the discrete cosine and sine transforms and their inverses. A preliminary call must be made to COSQI before COSQB, COSQF, SINQB, and SINQF can be used. After this preliminary call, COSQI need only be recalled when n is modified.

Programming. COSQI employs the subroutines RFFTI and RFFTI1. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL COSQB(n,X,WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $f(\theta_0), f(\theta_1), \ldots, f(\theta_{n-1})$. When COSQB is called, $4na_j$ is computed and stored in $X(j+1)$ for $j = 0,1,\ldots,n-1$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before COSQB can be used.

Programming. COSQB employs the subroutines COSB1, RFFTB, RFFTB1, RADB2, RADB3, RADB4, RADB5, and RADBG. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL COSQF(n,X,WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $a_0, a_1, \ldots, a_{n-1}$. When COSQF is called, $f(\theta_\nu)$ is computed and stored in $X(\nu + 1)$ for $\nu = 0,1,\ldots,n-1$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before COSQF can be used.

Example. Assume that X contains the data $f(\theta_0),\ldots,f(\theta_{n-1})$. When the statements

```
CALL COSQI(n,WK)
CALL COSQB(n,X,WK)
CALL COSQF(n,X,WK)
```

are called, COSQB stores $4na_0,\ldots,4na_{n-1}$ in X and COSQF then sets $X(\nu + 1) = 4nf(\theta_\nu)$ for $\nu = 0,1,\ldots,n-1$. Thus, the terms of the original sequence X are multiplied by 4n.

Programming. COSQF employs the subroutines COSQF1, RFFTF, RFFTF1, RADF2, RADF3, RADF4, RADF5, and RADFG. These routines were written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL SINQB(n,X,WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $f(\theta_0),\ldots,f(\theta_{n-1})$. When SINQB is called, $4nb_j$ is computed and stored in X(j) for $j = 1,\ldots,n$.

WK is an array of dimension $3n + 15$ or larger that is a work space for the routine. WK must be set up by the routine COSQI before SINQB can be used.

Programming. SINQB calls the subroutine COSQB. SINQB was written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

CALL SINQF(n,X,WK)

X is an array of dimension n or larger. On input it is assumed that X contains the data $b_1,\ldots,b_n$. When SINQF is called, $f(\theta_\nu)$ is computed and stored in $X(\nu + 1)$ for $\nu = 0,1,\ldots,n-1$.

WK is an array of dimension 3n + 15 or larger that is a work space for the routine. WK must be set up by the routine COSQI before SINQF can be used.

Example. Assume that X contains the data $b_1, \ldots, b_n$. When the statements

```
CALL COSQI(n,WK)
CALL SINQF(n,X,WK)
CALL SINQB(n,X,WK)
```

are called, SINQF stores $f(\theta_0), \ldots, f(\theta_{n-1})$ in X and SINQB then sets $X(j) = 4nb_j$ for $j = 1, \ldots, n$. Thus, the terms of the original sequence X are multiplied by 4n.

Programming. SINQF calls the subroutine COSQF. SINQF was written by Paul N. Swarztrauber (National Center for Atmospheric Research, Boulder, Colorado).

RATIONAL MINIMAX APPROXIMATION OF FUNCTIONS

Let $a < b$ and $g(x)$ be a continuous nonvanishing function on the interval $[a,b]$. For any continuous function $f(x)$, let $\|f\|$ denote the weighted norm $\max\{|f(x)|/|g(x)| : a \leqslant x \leqslant b\}$. Also let $\phi(x)$ be a continuous strictly monotonic mapping on $[a,b]$. Then for any nonnegative integers ℓ and m, the subroutine CHEBY is available for finding a rational function

$$R(x) = \frac{p_0 + p_1\phi(x) + \cdots + p_\ell\phi(x)^\ell}{q_0 + q_1\phi(x) + \cdots + q_m\phi(x)^m}$$

which minimizes $\|R - f\|$. The subroutine performs the calculations in double precision. It is assumed that the error curve $\delta(x) = (R(x) - f(x))/g(x)$ satisfies $|\delta(x_i)| = \|R - f\|$ at precisely $\ell + m + 2$ critical points $x_0 < x_1 < \cdots < x_n (n = \ell + m + 1)$, and that $\delta(x_{i+1}) = -\delta(x_i)$ for each $i < n$.

<u>CALL CHEBY(a,b,F,G,PHI,ϵ,ITER,MXITER,ℓ,m,P,Q,</u>
<u>ERROR, IERR, WK)</u>

The arguments a and b are double precision real numbers. F, G, and PHI are functions whose arguments and values are double precision real numbers. The functions must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL. The functions evaluate $f(x)$, $g(x)$, and $\phi(x)$ respectively.

The argument ϵ is a double precision tolerance that is supplied by the user. If λ denotes the estimated value of $\|R - f\|$, then the routine converges when the error curve $\delta(x)$ satisfies $\lambda(1 - \epsilon) \leqslant |\delta(x_i)| \leqslant \lambda(1 + \epsilon)$ for each x_i. Thus ϵ specifies the relative agreement that must be attained between $\|f - R\|$ and the $|\delta(x_i)|$. Normally the setting $\epsilon = 10^{-4}$ will give satisfactory results. It is required that $0 < \epsilon < 10^{-2}$ be satisfied.

The Remes-type algorithm designed by Cody, Fraser, and Hart is employed. This algorithm normally requires less than 20 iterations. The argument MXITER = the maximum number of iterations that may be performed. This argument is set by the user. The related argument ITER is an integer variable that is set by the routine. When CHEBY terminates, ITER will have for its value the number of iterations that were actually performed.

P is a double precision array of dimension $\ell + 1$, Q a double precision array of dimension $m + 1$, ERROR a double precision variable, and IERR an integer variable. When CHEBY terminates, if the rational function approximation $R(x)$ has been obtained then IERR is assigned the value 0 and ERROR is the estimated error $\|R - f\|$. The coefficient p_i of the numerator of $R(x)$ is stored in $P(i + 1)$ for $i = 0,1,\ldots,\ell$, and the coefficient q_j of the denominator is stored in $Q(j + 1)$ for $j = 0,1,\ldots,m$. The coefficient q_0 will always have the value 1.

Let $k = \ell + m + 2$. Then WK is a double precision array of dimension $k(k + 5)$ or larger that is used for a work space.

Error Return. IERR is assigned one of the following values when the desired minimizing rational function $R(x)$ is not obtained.

IERR = 1	An input error was detected. Either $\ell < 0$, $m < 0$, $\epsilon \leqslant 0$, $\epsilon \geqslant 10^{-2}$, or $g(x) = 0$ for some point x.
IERR = 2	MXITER iterations were performed. More iterations are needed to obtain $R(x)$.
IERR = 3	The system of linear equations that define the coefficients p_i and q_j was found to be singular. This indicates that for the current values of ℓ and m, the numerator and denominator of $R(x)$ may have common factors.
IERR = 4	A nonmonotonic sequence of critical points x_i was obtained. Modify ℓ and/or m.
IERR = 5	The value of the error curve $\delta(x)$ at some critical point x_i appears to be too large. This indicates that $R(x)$ may have poles, and that m (or possibly a or b) may have to be modified.
IERR = 6	CHEBY completely failed to find (or roughly approximate) $R(x)$. All information in P, Q, and ERROR should be ignored.

If IERR = 2, 3, 4, or 5 then P and Q contain the coefficients of the most recent rational function approximation $R(x)$ obtained, and ERROR is an estimate of the error $\|R - f\|$ of the approximation.

Remark. The two most common weighting functions employed are $g(x) = 1$ and $g(x) = f(x)$. If $g(x) = 1$ then the absolute error is minimized in constructing $R(x)$. If $g(x) = f(x)$ then the relative error is minimized.

Programming. CHEBY employs the subroutines CHEBY1, CERR, and DPSLV. These routines were written by A. H. Morris. CHEBY, CHEBY1, and CERR are slightly modified translations of the ALGOL 60 procedures Chebychev, lineq, del, and surmis given in the reference.

Reference. Cody, W. J., Fraser, W., and Hart, J. F., "Rational Chebychev Approximation using Linear Equations," ***Numerische Mathematik 12*** (1968), pp. 242-251.

L_p APPROXIMATION OF FUNCTIONS

For any continuous real-valued function $f(x)$ defined on the interval $[a,b]$, let $\|f\|_p$ denote the L_p norm defined by

$$\|f\|_p = \left(\int_a^b |f(x)|^p \, dx\right)^{1/p} \qquad \text{if } 0 < p < \infty$$

$$\|f\|_p = \max\{|f(x)| : a \leqslant x \leqslant b\} \qquad \text{if } p = \infty$$

If $p = \infty$ then the norm is also known as the ***Chebychev*** norm. For any continuous function f, $0 < p \leqslant \infty$, and $\epsilon > 0$, the subroutine ADAPT is available for finding a continuous piecewise polynomial function ϕ that satisfies $\|f - \phi\|_p \leqslant \epsilon$.

CALL ADAPT (F,a,b,ϵ,KNOTS,ERROR,XKNOTS,C,IERR,max,n,ℓ,
ANORM,DX,MO,m,XBREAK,KDIFF,DLEFT,DRIGHT)

It is assumed that the polynomials which form the approximation ϕ are of degree $\leqslant n$. The argument n must satisfy $1 \leqslant n \leqslant 19$, and IERR is a variable. When ADAPT is called, if there are no input errors and ϕ is successfully constructed, then IERR is set to 0, a sequence of points $a = x_1 < \cdots < x_k < x_{k+1} = b$ is selected, and ϕ takes the form

$$\phi(x) = c_{i0} + c_{i1}(x - x_i) + \cdots + c_{in}(x - x_i)^n \qquad x_i \leqslant x \leqslant x_{i+1}$$

for $i = 1,\ldots,k$. The points $x_1,\ldots,x_{k+1}$ are called the ***knots*** (or ***nodes***) of ϕ.

The argument max is the maximum number of polynomials that may be used in forming ϕ. KNOTS and ERROR are variables, XKNOTS an array of dimension max + 1 or larger, and C a 2-dimensional array of dimension max $\times$ (n+1). ADAPT stores the knots $x_1,\ldots,x_{k+1}$ in the XKNOTS array and the coefficients $c_{i0},\ldots,c_{in}$ in C(i,1),...,C(i,n+1) for $i = 1,\ldots,k$. The number k of polynomials actually generated always satisfies $k \leqslant$ max. ADAPT sets KNOTS to k + 1 (the total number of knots), and ERROR is a rough estimate of the error $\|f - \phi\|_p$ actually obtained.

The argument ℓ specifies the degree of smoothness that the approximation ϕ must satisfy. It is assumed that $0 \leqslant \ell < 10$ and $n > 2\ell$. If $\ell = 0$ then it is only required that ϕ be continuous on the interval $[a,b]$. Otherwise, if $\ell \geqslant 1$ then it is assumed that f is of class C^ℓ on $[a,b]$ except at possibly a finite number of points (called ***break points***), and it is required that ϕ be of class C^ℓ on $[a,b]$ except possibly at the break points.

The argument m specifies the number of break points of f. It is assumed that $m \leqslant 20$. If $m = 0$ then the arguments XBREAK,KDIFF,DLEFT, and DRIGHT can be ignored. Otherwise, if $m \geqslant 1$ then it is assumed that XBREAK,KDIFF,DLEFT, and DRIGHT are arrays of dimension m or larger, and that

XBREAK(i)	=	the ith break point, call it u_i
KDIFF(i)	=	the smallest integer ν_i for which the ν_i^{th} derivative of f does not exist or is not continuous at u_i
DLEFT(i)	=	the value from the left of the ν_i^{th} derivative at u_i
DRIGHT(i)	=	the value from the right of the ν_i^{th} derivative at u_i

for $i = 1,\ldots,m$. It is also assumed that $a \leqslant u_1 < \cdots < u_m \leqslant b$ and $n > 2\nu_i$ for each ν_i.

F is the name of a user defined function that has the value $F(x,D) = f(x)$ for $a \leqslant x \leqslant b$. If $\ell = 0$ then D can be ignored. (However, D must still be given as an argument of F.) Otherwise, if $\ell \geqslant 1$ then D is an array of dimension greater than or equal to ℓ. For any x not a break point in XBREAK, the user must set $D(j)$ = the jth derivative of f at x for $j \leqslant \ell$. However, if x = XBREAK(i) then the user need only set $D(j)$ = the jth derivative of f at x for $j <$ KDIFF(i). The function F must be declared in the calling program to be of type EXTERNAL.

The argument DX specifies the maximum distance to be permitted between the knots x_i and the argument ANORM specifies the norm to be used. Set

ANORN = ± 1.0	for L_1 approximation
ANORM = ± 2.0	for L_2 (least squares) approximation
ANORM = 3.0	for L_∞ (minimax) approximation
ANORM = −p	for L_p $(0 < p < \infty)$ approximation

Before considering the argument MO, one should be briefly acquainted with how ADAPT operates. ADAPT employs the following procedure to construct ϕ.

(1) Set I = [a,b] and k = 0. Let a be the first knot of ϕ.

(2) If the interior of I contains no break points then go to (3). Otherwise, if I = [c,d] then partition I into the subintervals [c,u] and [u,d] where u is the smallest break point greater than c. Stack the right subinterval [u,d] and reset I to [c,u].

(3) Construct a polynomial ϕ_I on I using Hermite interpolation. If the length of the interval I is $\leqslant$ DX and ϕ_I satisfactorily approximates f on I, then go to (4). Otherwise go to (5).

(4) Set k to be k + 1. Let ϕ_I be the kth polynomial forming ϕ and let the right end point of I be the (k + 1)-st knot of ϕ. If the interval stack is empty then the procedure is finished. Otherwise, obtain from the stack the next interval I to be considered and return to (2).

(5) The polynominal ϕ_I cannot be used. Partition I into halves, stack the right subinterval, and reset I to be the left subinterval. Then go to (3).

The argument MO specifies the accuracy criterion that the approximation ϕ is to satisfy on a subinterval $I = [c,d]$ of $[a,b]$. It is assumed that MO = 0,1,2. If the L_∞ norm is used then MO is ignored[1] and ϕ is required to satisfy $|f(x) - \phi(x)| \leqslant \epsilon$ for $c \leqslant x \leqslant d$. Otherwise, if the L_p $(0 < p < \infty)$ norm is used then ϕ is required to satisfy:

$$\int_c^d |f(x) - \phi(x)|^p \, dx \leqslant \frac{d - c}{b - a} \epsilon^p \qquad \text{for MO} = 0$$

$$\int_c^d |f(x) - \phi(x)|^p \, dx \leqslant \epsilon^p \qquad \text{for MO} = 2$$

The setting MO = 0, which is the most commonly used setting, requires the total error $||f-\phi||_p \leqslant \epsilon$. The alternate setting MO = 2 employs ϵ to control local accuracy. If ϕ consists of k polynomials then the total error $||f - \phi||_p \leqslant k^{1/p}\epsilon$. This setting can be useful when f is rough. A (heuristic) compromise strategy is provided when MO = 1. At each step in the formation of ϕ, the MO = 1 strategy estimates the total number of subintervals that will finally be needed and adjusts the error requirement for the subinterval I accordingly. This strategy attempts to keep the total error to a minimum while relaxing the local accuracy criterion demanded by the MO = 0 setting.

Remarks. KNOTS, IERR, max, n, ℓ, MO, m, and KDIFF are integer arguments. All other arguments (including F) are double precision arguments.

Error Return. ADAPT assigns IERR one of the following values:

IERR = 0	The approximation was successfully constructed.
IERR = -1	Either $a \geqslant b$ or one of the arguments ϵ, n, ℓ, ANORM, MO, m was assigned an incorrect value.
IERR = -2	[a,b] is too small an interval.
IERR = -3	DX is less than (b – a)/max. Since only max subintervals can be used and each subinterval must be of length $\leqslant$ DX, the interval [a,b,] cannot be covered. Make DX or max larger.
IERR = -4	The restriction $a \leqslant u_1 < \cdots < u_m \leqslant b$ on the break points was violated.
IERR = -5	Either KDIFF(i) < 0 or KDIFF(i) > (n – 1)/2 for some i.
IERR = 1	ADAPT selected max + 1 knots. More knots are needed to complete the problem.
IERR = 2	A subinterval I = [c,d] must be partitioned into subintervals [c,u] and [u,d] where u is a break point. However, this cannot be done either because the interval stack is full, or partitioning will produce too small an interval. (The stack can hold only 50 subintervals.)
IERR = 3	A subinterval must be partitioned because its length is greater than DX. However, this cannot be done since the interval stack is full.

[1]However, it is still required that MO = 0,1,2.

IERR = 4	A subinterval must be partitioned so that the accuracy criterion can be satisfied. However, this cannot be done either because the stack is full, or partitioning will produce too small an interval.

If an input error is detected (i.e., if IERR < 0) then no computation is performed. Otherwise, if IERR $\geqslant 0$ then when ADAPT terminates KNOTS = the number of knots generated, XKNOTS contains the knots, C contains the coefficients of the polynomials generated, and ERROR contains the error estimate for f-ϕ over the interval covered.

Remarks.

(1) If the L_∞ norm is used then ϵ controls absolute accuracy, not relative accuracy. This should be kept in mind when ϵ is to be set for any L_p norm.

(2) ADAPT requires more time when $\ell \geqslant 2$ than when $\ell = 0$ or 1. However, the choice of the norm normally has little effect on the efficiency of the routine.

(3) ADAPT can yield excellent results even when the derivatives of f have singularities. The one major exception is when the first derivative of f is not bounded. Then the routine can be expected to fail.

Example. The following code can be used for approximating $f(x) = e^x$ on the interval [0,1].

```
      DOUBLE PRECISION F, A, B, EPS, ERROR, ANORM, DX
      DOUBLE PRECISION XKNOTS (11), C(10,20)
      EXTERNAL F
      DATA MAX, A, B, DX/10, 0.D0, 1.D0, 1.D0/
      N = 8
      L = 1
      EPS = 1.D-12
      ANORM = 3.D0
      CALL ADAPT(F,A,B,EPS,KNOTS,ERROR,XKNOTS,C,IERR,
     *      MAX,N,L,ANORM,DX,0,0)
```

Here F may be defined by:

```
      DOUBLE PRECISION FUNCTION F(X,D)
      DOUBLE PRECISION X,D(1)
      F = DEXP(X)
      D(1) = F
      RETURN
      END
```

In the ADAPT statement the arrays XBREAK, KDIFF, DLEFT, and DRIGHT have been omitted since m = 0.

Programming. ADAPT employs the subroutines ADAPT1, ADSET, ADTAKE, ADCOMP, NEWTON, ADCHK, ADPUT, ADTRAN and the functions ERRINT, POLYDD. These routines exchange information in labeled common blocks. The block names are INPUTZ, RESULZ, KONTRL, and COMDIF. The routines were written by John R. Rice (Purdue University) and modified by A. H. Morris. The function DPMPAR is also used.

References

(1) Rice, J. R., "Algorithm 525. ADAPT, Adaptive Smooth Curve Fitting," ***A.C.M. Trans. Math Software 4*** (1978), pp. 82-94.

(2) __________, "Adaptive Approximation," ***J. Approx. Theory 16*** (1976), pp. 329-337.

LINEAR INTERPOLATION

Let a be a real number and $(x_1, y_1),\ldots,(x_n, y_n)$ a sequence of points. The following function performs a linear interpolation at point a.

TRP(a,n,X,Y)

It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$. X and Y are arrays containing the abscissas $x_1,\ldots,x_n$ and ordinates $y_1,\ldots,y_n$ respectively. TRP (a,n,X,Y) = b where b is the value of the interpolation at a.

Programmer. A. H. Morris

LAGRANGE INTERPOLATION

Let $\{(x_i,y_i) : i = 1,...,n\}$ be a set of $n \geqslant 2$ points where $x_1 < \cdots < x_n$, m be an integer where $2 \leqslant m \leqslant n$, and $\bar{x}_1,...,\bar{x}_k$ be $k \geqslant 1$ points at which m point Lagrange interpolation is to be performed. The subroutine LTRP is available for performing this interpolation.

CALL LTRP(m,X,Y,n,XI,YI,k,T,IERR)

X is an array containing $x_1,...,x_n$, Y an array containing $y_1,...,y_n$, XI an array containing $\bar{x}_1,...,\bar{x}_k$, and YI an array of dimension k or larger. When LTRP is called, if no input errors are detected then interpolation is performed at each $\bar{x}_j$ and the result stored in YI(j) for $j = 1,...,k$.

T is an array of dimension m or larger. The array is used as a temporary storage area by the routine.

Error Return. IERR is a variable that is set by the routine. If no input errors are detected then IERR is assigned the value 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	if	$m < 2$
IERR = 2	if	$m > n$
IERR = 3	if	$k < 1$

When an error is detected LTRP immediately terminates.

Algorithm. If $\bar{x}_j = (x_i + x_{i+m})/2$ for some i, then $(x_i,y_i),...,(x_{i+m-1},y_{i+m-1})$ are the m data points used in the Lagrange interpolation at $\bar{x}_j$. Otherwise, the data points selected for the interpolation are those m points (x_i, y_i) whose abscissas are closest to $\bar{x}_j$.

Linear Interpolation. For m = 2, if the abscissae x_i are not equally spaced then LTRP can produce different results than the linear interpolating function TRP. If $\bar{x}_j$ lies in the interval $[x_i, x_{i+1})$ then TRP always uses the data points (x_i, y_i) and (x_{i+1}, y_{i+1}) to find the interpolated value at $\bar{x}_j$. However, the operation of LTRP is somewhat different. For example, if the point $\bar{x}_j$ in $[x_i, x_{i+1})$ is closer to x_{i-1} than to x_{i+1}, then (x_{i-1}, y_{i-1}) and (x_i, y_i) will be the data points employed in the interpolation. Thus, TRP will normally be the procedure that will be desired for linear interpolation.

Programming. Developed by A. H. Morris. The portion of the code for finding the subinterval containing $\bar{x}_j$ was written by Rondall E. Jones (Sandia Laboratories).

HERMITE INTERPOLATION

Let $x_1,\ldots,x_k$ be $k \geqslant 1$ distinct points, $n_1,\ldots,n_k$ be positive integers, and $n = n_1 + \cdots + n_k$. Also given the data $y_i, y_i', \ldots, y_i^{(n_i-1)}$ for $i = 1,\ldots,k$. Then there exists a unique $n-1$ degree polynomial $p(x)$ which satisfies

$$\begin{aligned} p(x_i) &= y_i \\ p'(x_i) &= y_i' \\ &\vdots \\ p^{(n_i-1)}(x_i) &= y_i^{(n_i-1)} \end{aligned}$$

for each $i = 1,\ldots,k$. The subroutine HTRP is available for obtaining this polynomial.

<u>CALL HTRP(n,X,Y,A,WK,IERR)</u>

X and Y are arrays of dimension n containing the following information: $X(j) = x_1$ for $j = 1,\ldots,n_1$ and $Y(1),\ldots,Y(n_1)$ contain the values $y_1, y_1', \ldots, y_1^{(n_1-1)}$. For $i = 2,\ldots,k$ let $m_i = n_1 + \cdots + n_{i-1}$. Then $X(m_i + j) = x_i$ for $j = 1,\ldots,n_i$ and $Y(m_i + 1),\ldots,Y(m_i + n_i)$ contain the values $y_i, y_i', \ldots, y_i^{(n_i-1)}$.

A is an array of dimension n and IERR an integer variable. When HTRP is called, if no errors are detected then IERR is assigned the value 0 and the coefficients a_j of the polynomial $p(x) = a_0 + \sum_{j=1}^{n-1} a_j(x - X(1)) \cdots (x - X(j))$ are computed and stored in $A(j+1)$ for $j = 0, 1, \ldots, n-1$.

WK is an array of dimension n or larger that is a work space for the routine.

Error Return. If an error is detected then IERR is assigned one of the following values:

IERR = 1 The argument n is not positive.

IERR = 2 There exists integers i and ℓ for which $X(i) = X(\ell)$ but $X(i) \neq X(j)$ for some j where $i < j < \ell$. In this case, the values i and ℓ are stored (in floating point form) in WK(1) and WK(2).

When an error is detected, the routine immediately terminates.

Example. If $p(0) = 2$, $p(-1) = 1$, and $p'(-1) = 2$ where $x_1 = 0$ and $x_2 = -1$, then HTRP stores 2, 1, -1 in A. Hence, $p(x) = 2 + x - x(x+1)$ is the desired polynomial.

Remark. The Newton representation $a_0 + \sum_{j=1}^{n-1} a_j(x - X(1)) \cdots (x - X(j))$ of the polynomial $p(x)$ can be converted to the Taylor series representation $\sum_{j=0}^{n-1} c_j(x-\alpha)^j$ by the subroutine PCOEFF.

Programmer. A. H. Morris

CONVERSION OF REAL POLYNOMIALS FROM NEWTON TO TAYLOR SERIES FORM

For $n \geqslant 1$ let $p(x) = a_0 + \sum_{j=1}^{n-1} a_j(x-x_1) \cdots (x-x_j)$. Then for any real number α, the subroutine PCOEFF is available for converting the polynomial $p(x)$ to the Taylor series form $\sum_{j=0}^{n-1} c_j(x-\alpha)^j$.

<u>CALL PCOEFF(α,n,X,A,C,T)</u>

X is a single precision real array containing $x_1, ..., x_{n-1}$, A a single precision real array containing $a_0, a_1, ..., a_{n-1}$ where a_j is stored in A(j + 1) for j = 0, 1, ..., n − 1, and C a single precision real array of dimension n or larger. When PCOEFF is called then the coefficients c_j of the Taylor series representation are computed and stored in C(j + 1) for j = 0, 1, ..., n − 1.

T is a ***double precision*** array of dimension n or larger. The array is a work space for the routine. (The conversion of the coefficients is done in double precision.)

Note. A and C may reference the same storage area. If they do reference the same storage area then the results c_j will overwrite the input data a_j.

Programmer. A. H. Morris

LEAST SQUARES POLYNOMIAL FIT

Let $\{(x_i, y_i) : i = 1,...,m\}$ be a set of $m \geqslant 2$ points where $x_i \neq x_j$ for $i \neq j$. Then for any positive integer n where $n < m$, the subroutine PFIT is available for obtaining the (unique) nth degree polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$ which minimizes $\sum_{i=1}^{m} (p(x_i) - y_i)^2$.

CALL PFIT(n,m,X,Y,A,RNORM,PHI,WK,IERR)

X is an array containing $x_1,...,x_m$, Y an array containing $y_1,...,y_m$, and A an array of dimension n + 1 or larger. RNORM and IERR are variables. When PFIT is called, if no input errors are detected then IERR is set to 0, the coefficients a_j of p(x) are stored in A(j + 1) for j = 0,1,...,n, and RNORM is assigned the value $\sqrt{\Sigma_i (p(x_i) - y_i)^2}$.

PHI is an array of dimension 2(n + 1) or larger, and WK is an array of dimension 4m or larger. PHI and WK are work spaces for the routine.

Error Return. IERR = 1 if $n < 1$ or $n \geqslant m$.

Algorithm. The abscissas x_i are first mapped into values in the interval $[-1,1]$. Then the Forsythe procedure is used.

Programmer. A. H. Morris

WEIGHTED LEAST SQUARES POLYNOMIAL FIT

Let $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of $m \geqslant 2$ points where $x_i \neq x_j$ for $i \neq j$, and let $w_i \geqslant 0\,(i = 1, \ldots, m)$ be weights. It is assumed that $m_w \geqslant 2$ where m_w is the number of nonzero weights. For any positive integer n where $n < m_w$, the subroutine WPFIT is available for finding the (unique) nth degree polynomial $p(x) = \sum_{j=0}^{n} a_j x^j$ which minimizes $\sum_{i=1}^{m} w_i(p(x_i) - y_i)^2$.

CALL WPFIT(n,m,X,Y,W,A,RNORM,PHI,WK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, W an array containing $w_1, \ldots, w_m$, and A an array of dimension n + 1 or larger. RNORM and IERR are variables. When WPFIT is called, if no input errors are detected then IERR is set to 0, the coefficients a_j of $p(x)$ are computed and stored in A(j + 1) for j = 0, 1, ..., n, and RNORM is assigned the value $\sqrt{\Sigma_i\, w_i(p(x_i) - y_i)^2}$.

PHI is an array of dimension 2(n + 1) or larger, and WK is an array of dimension 4m or larger. PHI and WK are work spaces for the routine.

Error Return. IERR = 1 if $n < 1$ or $n \geqslant m_w$, and IERR = 3 if some w_i is negative.

Algorithm. The abscissas x_i corresponding to the positive weights are first mapped into values in the interval $[-1,1]$. Then the Forsythe procedure is used.

Programmer. A. H. Morris

CUBIC SPLINE INTERPOLATION

Given $x_1 < \cdots < x_n$. A function f(x) is a ***cubic spline having the nodes (knots)*** $x_1,\ldots,x_n$ if f is a polynomial of degree $\leqslant 3$ on the interval $[x_i, x_{i+1}]$ for $i = 1,\ldots,n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, f is obtained by fitting the polynomials $f_1,\ldots,f_{n-1}$ together at the points $x_2,\ldots,x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1,\ldots,n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1,\ldots,n$.

Assume now that the ordinates $y_1,\ldots,y_n$ are given. Then there exist an infinitude of splines with nodes $x_1,\ldots,x_n$ that interpolate the points (x_i, y_i). However, only one of these splines satisfies the conditions:

$$f''(x_1) = \alpha f''(x_2) + \beta \qquad |\alpha| < 1$$
$$f''(x_n) = \bar{\alpha} f''(x_{n-1}) + \bar{\beta} \qquad |\bar{\alpha}| < 1$$

The following subroutine is available for obtaining this spline.

CALL SPLIFT(X,Y,DY,DDY,n,W,IERR,MO,$\alpha,\beta,\bar{\alpha},\bar{\beta}$)

X and Y are arrays containing the abscissas $x_1,\ldots,x_n$ and ordinates $y_1,\ldots,y_n$ respectively. It is assumed that $x_1 < \cdots < x_n$ and $n \geqslant 4$. DY and DDY are arrays of dimension n or larger, and IERR is an integer variable. When SPLIFT is called, if there are no input errors then IERR is assigned the value 0, the first derivatives $f'(x_1),\ldots,f'(x_n)$ are computed and stored in DY(1),...,DY(n), and the second derivatives $f''(x_1),\ldots,f''(x_n)$ are computed and stored in DDY(1),...,DDY(n).

W is an array of dimension 3n or larger that is used for a storage area. On the first call to SPLIFT the argument MO must be set to 0. When SPLIFT is initially called, certain calculations which depend only on the values of α, $\bar{\alpha}$, and $x_1,\ldots,x_n$ are performed and the results stored in W. On subsequent calls to SPLIFT, if only the values of β, $\bar{\beta}$ and/or $y_1,\ldots,y_n$ are modified, then the information in W need not be recomputed. Set MO = 1 and the information in W will be reused.

Error Return. If there is an input error then IERR is set as follows:

IERR = 1	if $|\alpha| \geqslant 1$ or $|\bar{\alpha}| \geqslant 1$
IERR = 2	if $n < 4$
IERR = 3	if the restriction $x_1 < \cdots < x_n$ is not satisfied

Remarks

(1) After DY and DDY have been obtained, then either SCOMP1 or SCOMP2 may be used to evaluate the spline at any point x. SEVAL1 or SEVAL2 may be used if derivatives are also desired.

(2) Given the values y_1' and y_n'. Then there exists a unique interpolating cubic spline f that satisfies $f'(x_1) = y_1'$ and $f'(x_n) = y_n'$. This spline can be obtained by setting $\alpha = \bar{\alpha} = -1/2$ and

$$\beta = \frac{3}{x_2 - x_1}\left[\frac{y_2 - y_1}{x_2 - x_1} - y_1'\right]$$

$$\bar{\beta} = \frac{-3}{x_n - x_{n-1}}\left[\frac{y_n - y_{n-1}}{x_n - x_{n-1}} - y_n'\right] .$$

Programmer. Rondall E. Jones (Sandia Laboratories).

WEIGHTED LEAST SQUARE CUBIC SPLINE FITTING

Let $t_1 < \cdots < t_\ell$ be a sequence where $\ell \geqslant 2$, $\{(x_i, y_i) : i = 1, \ldots, m\}$ be a set of $m \geqslant 4$ points where $t_1 \leqslant x_1 < \cdots < x_m \leqslant t_\ell$, and $w_1, \ldots, w_m$ be positive weights. Then the subroutine SPFIT is available for obtaining a cubic spline f(x) with the nodes $t_1, \ldots, t_\ell$ which minimizes $\sum_{i=1}^{m} w_i (f(x_i) - y_i)^2$. This spline is represented by

$$f(x) = z_j + a_j (x - t_j) + b_j (x - t_j)^2 + c_j (x - t_j)^3$$

for $t_i \leqslant x < t_{j+1}$ $(j = 1, \ldots, \ell - 1)$. If the nodes are selected so that $\ell \leqslant m - 2$ and each interval (t_j, t_{j+1}) contains a data point x_{ν_j}, then this least squares approximation is unique.

CALL SPFIT(X,Y,W,m,T,ℓ,Z,A,B,C,WK,IERR)

X is an array containing $x_1, \ldots, x_m$, Y an array containing $y_1, \ldots, y_m$, W an array containing $w_1, \ldots, w_m$, and T an array containing $t_1, \ldots, t_\ell$. Z, A, B, C are arrays of dimension $\ell - 1$ or larger, and IERR is an integer variable. When SPFIT is called, if no input errors are detected then IERR is set to 0. Also, the coefficients z_j, a_j, b_j, c_j of the least squares approximating spline f(x) are computed and stored in Z, A, B, C.

WK is an array of dimension $7\ell + 18$ or larger that is a work space for the routine.

Error Return. IERR is set to one of the following values when an input error is detected.

IERR = 1 if $\ell < 2$.
IERR = 2 if $t_1 < \cdots < t_\ell$ is not satisfied.
IERR = 3 if $m \geqslant 4$ and $t_1 \leqslant x_1 \leqslant \cdots \leqslant x_m \leqslant t_\ell$ are not satisfied.

If an error is detected, the routine immediately terminates.

Remark. After A, B, C, and Z have been obtained, then SCOMP may be used to evaluate the spline at any point x. SEVAL may be used if derivatives are also desired.

Programming. SPFIT employs the subroutines BSPP, BSL2, BSPEV, BCHFAC, and BCHSLV. SPFIT was written by A. H. Morris.

CUBIC SPLINE EVALUATION

Given $x_1 < \cdots < x_n$. A function f(x) is a ***cubic spline having the nodes (knots)*** $\boldsymbol{x_1,\ldots,x_n}$ if f is a polynomial of degree $\leqslant 3$ on the interval $[x_i, x_{i+1}]$ for $i = 1,\ldots,n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, f is obtained by fitting the polynomials $f_1,\ldots,f_{n-1}$ together at the points $x_2,\ldots,x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1,\ldots,n - 1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1,\ldots,n$.

A cubic spline f given by the polynomials $f_1,\ldots,f_{n-1}$ is uniquely defined by any of the following three sets of data:

(1) the points (x_i, y_i) and coefficients a_i, b_i, c_i for $i = 1,\ldots,n - 1$
(2) the points (x_i, y_i) and first derivatives $f'(x_i)$ for $i = 1,\ldots,n$
(3) the points (x_i, y_i) and second derivatives $f''(x_i)$ for $i = 1,\ldots,n$

The subroutines SCOMP, SCOMP1, SCOMP2 are available for computing the spline at any point x. SCOMP is used if data set (1) is given, SCOMP1 is used if data set (2) is given, and SCOMP2 is used if data set (3) is given. SCOMP is faster than SCOMP1, and SCOMP1 is faster than SCOMP2.

CALL SCOMP(X,Y,A,B,C,N,XI,YI,m,IERR)

Let $N = n - 1$. Then N is the number of polynomials f_i that form the spline, X and Y are arrays containing the abscissas $x_1,\ldots,x_N$ and ordinates $y_1,\ldots,y_N$, and A, B, C are arrays containing the coefficients a_i, b_i, c_i $(i = 1,\ldots,N)$. It is assumed that $N \geqslant 1$ and that $x_1 < \cdots < x_N$.

Let $\bar{x}_1,\ldots,\bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1,\ldots,\bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1,\ldots,m$.

Note. SCOMP does not require f to be a spline. It is only required that $f_i(x)$ be a cubic polynomial $y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$ and that

$$f(x) = f_1(x) \quad \text{for} \quad x < x_1$$
$$f(x) = f_i(x) \quad \text{for} \quad x_i \leqslant x < x_{i+1} \quad (1 \leqslant i < N)$$
$$f(x) = f_N(x) \quad \text{for} \quad x \geqslant x_N.$$

In this case, SCOMP computes the value $f(\bar{x}_j+)$ for $j = 1,\ldots,m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

<u>CALL SCOMP1(X,Y,DY,n,XI,YI,m,IERR)</u>

X, Y, DY are arrays containing the abscissas $x_1,...,x_n$, ordinates $y_1,...,y_n$, and first derivatives $f'(x_1),...,f'(x_n)$ respectively. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1,...,\bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1,...,\bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP1 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1,...,m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

<u>CALL SCOMP2(X,Y,DDY,n,XI,YI,m,IERR)</u>

X, Y, DDY are arrays containing the abscissas $x_1,...,x_n$, ordinates $y_1,...,y_n$, and second derivatives $f''(x_1),...,f''(x_n)$ respectively. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1....,\bar{x}_m$ be the points at which the spline f is to be evaluated. XI is an array containing $\bar{x}_1,...,\bar{x}_m$, YI an array of dimension m or larger, and IERR a variable. When SCOMP2 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and $f(\bar{x}_j)$ is computed and stored in YI(j) for $j = 1,...,m$.

Programmer. Rondall E. Jones (Sandia Laboratories)

CUBIC SPLINE EVALUATION AND DIFFERENTIATION

Given $x_1 < \cdots < x_n$. A function f(x) is a ***cubic spline having the nodes (knots)*** $x_1,\dots,x_n$ if f is a polynomial of degree $\leqslant 3$ on the interval $[x_i, x_{i+1}]$ for $i = 1,\dots,n-1$, and the first and second derivatives $f'(x)$ and $f''(x)$ exist and are continuous for all x. If f_i denotes the polynomial for the interval $[x_i, x_{i+1}]$ then f_i has the form:

$$f_i(x) = y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$$

Consequently, f is obtained by fitting the polynomials $f_1,\dots,f_{n-1}$ together at the points $x_2,\dots,x_{n-1}$. For $x < x_1$ $f(x) = f_1(x)$, and for $x > x_n$ $f(x) = f_{n-1}(x)$. Also $f(x_i) = y_i$ for $i = 1,\dots,n-1$. Hence, if $f(x_n) = y_n$ then f interpolates the points (x_i, y_i) for $i = 1,\dots,n$.

A cubic spline f given by the polynomials $f_1,\dots,f_{n-1}$ is uniquely defined by any of the following three sets of data:

(1) the points (x_i, y_i) and coefficients a_i, b_i, c_i for $i = 1,\dots,n-1$
(2) the points (x_i, y_i) and first derivatives $f'(x_i)$ for $i = 1,\dots,n$
(3) the points (x_i, y_i) and second derivatives $f''(x_i)$ for $i = 1,\dots,n$

The subroutines SEVAL, SEVAL1, SEVAL2 are available for computing the spline and its first and second derivatives at any point x. SEVAL is used if data set (1) is given, SEVAL1 is used if data set (2) is given, and SEVAL2 is used if data set (3) is given. SEVAL is faster than SEVAL1, and SEVAL1 is faster than SEVAL2.

CALL SEVAL (X,Y,A,B,C,N,XI,YI,DYI,DDYI,m,IERR)

Let $N = n - 1$. Then N is the number of polynomials f_i that form the spline, X and Y are arrays containing the abscissas $x_1,\dots,x_N$ and ordinates $y_1,\dots,y_N$, and A, B, C are arrays containing the coefficients a_i, b_i, c_i $(i = 1,\dots,N)$. It is assumed that $N \geqslant 1$ and that $x_1 < \cdots < x_N$.

Let $\bar{x}_1,\dots,\bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1,\dots,\bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger, and IERR is a variable. When SEVAL is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and the values $f(\bar{x}_j)$, $f'(\bar{x}_j)$, $f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1,\dots,m$.

Note. SEVAL does not require f to be a spline. It is only required that $f_i(x)$ be a cubic polynomial $y_i + a_i(x - x_i) + b_i(x - x_i)^2 + c_i(x - x_i)^3$ and that

$$f(x) = f_1(x) \quad \text{for} \quad x < x_1$$
$$f(x) = f_i(x) \quad \text{for} \quad x_i \leqslant x < x_{i+1} \quad (1 \leqslant i < N)$$
$$f(x) = f_N(x) \quad \text{for} \quad x \geqslant x_N.$$

In this case, SEVAL computes the values $f(\bar{x}_j+)$, $f'(\bar{x}_j+)$, $f''(\bar{x}_j+)$ for $j = 1,\dots,m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SEVAL1(X,Y,DY,n,XI,YI,DYI,DDYI,m,IERR)

X, Y, DY are arrays containing the abscissas $x_1, \dots, x_n$, ordinates $y_1, \dots, y_n$, and first derivatives $f'(x_1), \dots, f'(x_n)$ respectively. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1, \dots, \bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \dots, \bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger, and IERR is a variable. When SEVAL1 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and the values $f(\bar{x}_j)$, $f'(\bar{x}_j)$, $f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \dots, m$.

Programming. Adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

CALL SEVAL2 (X,Y,DDY,n,XI,YI,DYI,DDYI,m,IERR)

X, Y, DDY are arrays containing the abscissas $x_1, \dots, x_n$, ordinates $y_1, \dots, y_n$, and second derivatives $f''(x_1), \dots, f''(x_n)$ respectively. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$.

Let $\bar{x}_1, \dots, \bar{x}_m$ be the points at which the spline f and its first two derivatives are to be evaluated. XI is an array containing $\bar{x}_1, \dots, \bar{x}_m$, YI, DYI, DDYI are arrays of dimension m or larger, and IERR is a variable. When SEVAL2 is called, if $m < 1$ then IERR is set to 1 and the routine terminates. Otherwise, if $m \geqslant 1$ then IERR is set to 0 and the values $f(\bar{x}_j)$, $f'(\bar{x}_j)$, $f''(\bar{x}_j)$ are computed and stored in YI(j), DYI(j), DDYI(j) for $j = 1, \dots, m$.

Programmer. Rondall E. Jones (Sandia Laboratories)

SPLINE UNDER TENSION INTERPOLATION

Given real σ and $x_1 < \cdots < x_n$. A function f(x) is a ***spline having the tension factor σ and the nodes (knots) $x_1,\ldots,x_n$*** if f(x) and its first two derivatives are continuous on $[x_1, x_n]$, and $f''(x) - \bar{\sigma}^2 f(x) = a_i x + b_i$ on the interval $[x_i, x_{i+1}]$ for $i = 1,\ldots,n-1$. Here $\bar{\sigma} = |\sigma|(n-1)/(x_n - x_1)$ and a_i, b_i are constants. For $x_i \leqslant x \leqslant x_{i+1}$ f(x) can be represented by

$$f(x) = A_i \sinh \bar{\sigma}(x - x_i) + B_i \sinh \bar{\sigma}(x_{i+1} - x) - (a_i x + b_i)/\bar{\sigma}^2$$

when $\sigma \neq 0$, and by a cubic polynomial when $\sigma = 0$.

Assume now that n ordinates $y_1,\ldots,y_n$ are given. Then there exist an infinitude of splines f(x) having tension σ for which $f(x_i) = y_i$ $(i = 1,\ldots,n)$. However, if values y_1' and y_n' are given then only one of these splines will satisfy $f'(x_1) = y_1'$ and $f'(x_n) = y_n'$. For convenience, denote this spline by f_σ. If $\sigma = 0$ then it is clear that f_σ is the standard cubic spline. Also it can be verified that when $\sigma \to \infty$, f_σ converges uniformly on $[x_1, x_n]$ to the piecewise linear function $\ell(x)$ where $\ell(x) = y_i + m_i(x - x_i)$ for $x_i \leqslant x \leqslant x_{i+1}$ $(i = 1,\ldots,n-1)$. Here $m_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$. The following subroutine is available for obtaining the spline f_σ.

CALL CURV1(n,X,Y,SLP1,SLPN,IND,DDY,TEMP,σ,IERR)

X and Y are arrays containing the abscissas $x_1,\ldots,x_n$ and ordinates $y_1,\ldots,y_n$. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$.

SLP1 and SLPN are assigned the values y_1' and y_n'. The user may omit values for either or both of these arguments. IND specifies the information that is provided.

IND = 0	Values are supplied for SLP1 and SLPN.
IND = 1	A value is supplied for SLP1 but not for SLPN.
IND = 2	A value is supplied for SLPN but not for SLP1.
IND = 3	Values are not supplied for SLP1 and SLPN.

If a value is not supplied by the user, then the routine provides a value.

DDY is an array of dimension n or larger, and IERR is an integer variable. When CURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the second derivatives $f_\sigma''(x_1),\ldots,f_\sigma''(x_n)$ are computed and stored in DDY.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$

IERR = 2 if $x_1 < \cdots < x_n$ is not satisfied

When either of these errors is detected then the routine immediately terminates.

Remarks

(1) After DDY is obtained then CURV2 may be used to evaluate the spline at any point x.

(2) X, Y, n, SLP1, SLPN, IND, σ are not modified by CURV1.

Programming. CURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. CURV1, CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

Reference. Cline, A. K., "Scalar and Planar Valued Curve Fitting using Splines under Tension," ***Communications ACM 17*** (April 1974), pp. 218–220.

SPLINE UNDER TENSION EVALUATION

Given σ and $x_1 < \cdots < x_n$. A function f(x) is a ***spline having the tension factor σ and the nodes (knots) $x_1,\ldots,x_n$*** if f(x) and its first two derivatives are continuous on $[x_1, x_n]$, and $f''(x) - \bar{\sigma}^2 f(x) = a_i x + b_i$ on the interval $[x_i, x_{i+1}]$ for $i = 1,\ldots,n-1$. Here $\bar{\sigma} = |\sigma|(n-1)/(x_n - x_1)$ and a_i, b_i are constants. If $f(x) = f_i(x)$ for $x_i \leqslant x \leqslant x_{i+1}$ then $f_i(x)$ can be represented by

$$f_i(x) = A_i \sinh \bar{\sigma}(x - x_i) + B_i \sinh \bar{\sigma}(x_{i+1} - x) - (a_i x + b_i)/\bar{\sigma}^2$$

when $\sigma \neq 0$, and by a cubic polynomial when $\sigma = 0$. For $x < x_1$ we let $f(x) = f_1(x)$, and for $x > x_n$ we let $f(x) = f_{n-1}(x)$.

Assume now that $f(x_i) = y_i$ for $i = 1,\ldots,n$. Then for a fixed σ, f(x) is uniquely defined by the points (x_i, y_i) and the second derivatives $f''(x_i)$ $(i = 1,\ldots,n)$. When this data is available, the following function may be used for computing the spline at any point t.

CURV2(t,n,X,Y,DDY,σ)

X and Y are arrays containing the abscissas $x_1,\ldots,x_n$ and ordinates $y_1,\ldots,y_n$, and DDY is an array containing the second derivatives $f''(x_1),\ldots,f''(x_n)$. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$. CURV2(t,n,X,Y,DDY,σ) = f(t) for any real t.

Remark. After DDY has been obtained, CURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified, then it should be emphasized that the derivative information in DDY will have to be recomputed before CURV2 can be used with the new tension factor.

Programming. CURV2 employs the function INTRVL and subroutine SNHCSH. CURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

Reference. Cline, A. K., "Scalar and Planar Valued Curve Fitting using Splines under Tension," ***Communications ACM 17*** (April 1974), pp. 218–220.

DIFFERENTIATION AND INTEGRATION OF SPLINES UNDER TENSION

Let f(x) be a spline having the tension factor σ and the nodes $x_1,...,x_n$. Assume that $f(x_i) = y_i$ for $i = 1,...,n$. If the second derivatives $f''(x_1),...,f''(x_n)$ are known then the following functions may be used for differentiating and integrating the spline.

CURVD(t,n,X,Y,DDY,σ)

X and Y are arrays containing the abscissas $x_1,...,x_n$ and ordinates $y_1,...,y_n$, and DDY is an array containing the second derivatives $f''(x_1),...,f''(x_n)$. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$. For any real t, the derivative f′(t) is computed and assigned to be the value of CURVD(t, n, X, Y, DDY, σ).

Programming. CURVD employs the function INTRVL and subroutine SNHCSH. CURVD was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

CURVI(a,b,n,X,Y,DDY, σ)

X and Y are arrays containing the abscissas $x_1,...,x_n$ and ordinates $y_1,...,y_n$, and DDY is an array containing the second derivatives $f''(x_1),...,f''(x_n)$. It is assumed that $n \geqslant 2$ and $x_1 < \cdots < x_n$. CURVI(a, b, n, X, Y, DDY, σ) = $\int_a^b f(t)\,dt$ for any real a and b.

Note. It is not required that $a \leqslant b$.

Programming. CURVI employs the function INTRVL and subroutine SNHCSH. CURVI was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

TWO DIMENSIONAL SPLINE UNDER TENSION CURVE FITTING

Given a sequence of points $(x_1, y_1), \dots, (x_n, y_n)$. One procedure for fitting a curve to the points is to let $s_1 = 0$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for $i = 2, \dots, n$, and then to find two splines $x(s)$ and $y(s)$ with tension σ that satisfy $x(s_i) = x_i$ and $y(s_i) = y_i$ for $i = 1, \dots, n$. If θ_1 and θ_n are the desired angles for the curve $s \to (x(s), y(s))$ at the points (x_1, y_1) and (x_n, y_n), then the splines $x(s)$ and $y(s)$ can be selected so that $x'(s_i) = \cos\theta_i$ and $y'(s_i) = \sin\theta_i$ for $i = 1, n$. The curve $s \to (x(s), y(s))$ then passes through the points (x_i, y_i) and has the required slopes at the end points. The subroutine KURV1 is available for obtaining the second derivatives $x''(s_i)$, $y''(s_i)$ $(i = 1, \dots, n)$ which characterize this curve, and the subroutine KURV2 is available for computing the curve.

CALL KURV1(n,X,Y,SLP1,SLPN,IND,DDX,DDY,TEMP,S,σ,IERR)

X and Y are arrays containing the abscissas $x_1, \dots, x_n$ and ordinates $y_1, \dots, y_n$. It is assumed that $n \geqslant 2$ and that the points (x_i, y_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ for $i = 1, \dots, n-1$.

SLP1 and SLPN are assigned the values θ_1 and θ_n. These angles are measured counter-clockwise (in radians) from the positive x-axis. The user may omit values for SLP1 and/or SLPN. IND specifies the information that is provided.

IND = 0	Values are supplied for SLP1 and SLPN.
IND = 1	A value is supplied for SLP1 but not for SLPN.
IND = 2	A value is supplied for SLPN but not for SLP1.
IND = 3	Values are not supplied for SLP1 and SLPN.

If a value is not supplied by the user, then the routine provides a value.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s)$ and $y(s)$ approximate cubic splines and the curve may be quite wavy. However, if $|\sigma|$ is large, say $|\sigma| > 100$, then the resulting curve will approximate the polygonal line from (x_1, y_1) to (x_n, y_n).

IERR is an integer variable and S, DDX, DDY are arrays of dimension n or larger. When KURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1, \dots, s_n$ are computed and stored in S. Also, the second derivatives $x''(s_1), \dots, x''(s_n)$ and $y''(s_1), \dots, y''(s_n)$ are computed and stored in DDX and DDY.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1   if $n < 2$

IERR = 2   if $(x_i, y_i) = (x_{i+1}, y_{i+1})$ for some i

When either of these errors is detected then the routine immediately terminates.

Remarks

(1) After S, DDX, DDY are obtained then KURV2 may be used to compute the curve.

(2) X, Y, n, SLP1, SLPN, IND, σ are not modified by KURV1.

Programming. KURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. KURV1, CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

<u>CALL KURV2(t,XT,YT,n,X,Y,DDX,DDY,S,σ)</u>

X and Y are arrays containing the abscissas $x_1,\ldots,x_n$ and ordinates $y_1,\ldots,y_n$, S is an array containing $s_1,\ldots,s_n$, and DDX and DDY are arrays containing the second derivatives $x''(s_1),\ldots,x''(s_n)$ and $y''(s_1),\ldots,y''(s_n)$.

Now consider the change of variables $t = s/s_n$, and let $t \rightarrow (\bar{x}(t), \bar{y}(t))$ denote the curve in terms of the new parameter t. XT and YT are real variables. For any $0 \leqslant t \leqslant 1$, KURV2 computes the point $(\bar{x}(t), \bar{y}(t))$ on the curve and assigns XT the value $\bar{x}(t)$ and YT the value $\bar{y}(t)$.

Remark. After DDX and DDY have been obtained, KURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified, then it should be emphasized that the derivative information in DDX and DDY will have to be recomputed before KURV2 can be used with the new tension factor.

Programming. KURV2 employs the function INTRVL and subroutine SNHCSH. KURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

TWO DIMENSIONAL SPLINE UNDER TENSION CLOSED CURVE FITTING

Given a sequence of points $(x_1, y_1), \ldots, (x_n, y_n)$. One procedure for fitting a closed curve to the points is to let $s_1 = \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2}$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for $i = 2, \ldots, n$, and then to find periodic splines $x(s)$ and $y(s)$ with tension σ that pass through the points $(s_1, x_1), \ldots, (s_n, x_n)$, $(s_1 + s_n, x_1)$ and $(s_1, y_1), \ldots, (s_n, y_n)$, $(s_1 + s_n, y_1)$. The mapping $s \rightarrow (x(s), y(s))$ then defines a closed curve that passes through the points (x_i, y_i). The subroutine KURVP1 is available for obtaining the second derivatives $x''(s_i)$, $y''(s_i)$ $(i = 1, \ldots, n)$ which characterize this curve, and the subroutine KURVP2 is available for computing the curve.

CALL KURVP1(n,X,Y,DDX,DDY,TEMP,S,σ,IERR)

X and Y are arrays containing the abscissas $x_1, \ldots, x_n$ and ordinates $y_1, \ldots, y_n$. It is assumed that $n \geqslant 2$ and that the points (x_i, y_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ for $i = 1, \ldots, n-1$.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s)$ and $y(s)$ approximate periodic cubic splines and the curve may be quite wavy. However, if $|\sigma|$ is large, say $|\sigma| > 100$, then the curve will approximate the closed polygonal path that traverses the points (x_i, y_i).

IERR is an integer variable and S, DDX, DDY are arrays of dimension n or larger. When KURVP1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1, \ldots, s_n$ are computed and stored in S. Also, the second derivatives $x''(s_1), \ldots, x''(s_n)$ and $y''(s_1), \ldots, y''(s_n)$ are computed and stored in DDX and DDY.

TEMP is an array of dimension 2n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$

IERR = 2 if $(x_i, y_i) = (x_{i+1}, y_{i+1})$ for some i

When either of these errors is detected then the routine immediately terminates.

Remark. After S, DDX, DDY are obtained the KURVP2 may be used to compute the curve.

Programming. KURVP1 employs the subroutines TERMS and SNHCSH. KURVP1 and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL KURVP2(t,XT,YT,n,X,Y,DDX,DDY,S,σ)

X and Y are arrays containing the abscissas $x_1,...,x_n$ and ordinates $y_1,...,y_n$, S is an array containing $s_1,...,s_n$, and DDX and DDY are arrays containing the second derivatives $x''(s_1),...,x''(s_n)$ and $y''(s_1),...,y''(s_n)$.

Now consider the change of variables $t = (s - s_1)/s_n$, and let $t \rightarrow (\bar{x}(t), \bar{y}(t))$ denote the mapping $s \rightarrow (x(s), y(s))$ in terms of the new parameter t. Then $t \rightarrow (\bar{x}(t), \bar{y}(t))$ maps the interval [0,1] onto the entire closed curve, taking both 0 and 1 into the point (x_1, y_1). Also $t \rightarrow (\bar{x}(t), \bar{y}(t))$ is a periodic function (with period 1).

XT and YT are real variables. For any real t, KURVP2 computes the point $(\bar{x}(t), \bar{y}(t))$ on the curve and assigns XT the value $\bar{x}(t)$ and YT the value $\bar{y}(t)$.

Remark. After DDX and DDY have been obtained, KURVP2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified then it should be emphasized that the derivative information in DDX and DDY will have to be recomputed before KURVP2 can be used with the new tension factor.

Programming. KURVP2 employs the function INTRVL and subroutine SNHCSH. KURVP2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin). INTRVL was written by A. H. Morris.

THREE DIMENSIONAL SPLINE UNDER TENSION CURVE FITTING

Given a sequence of points $(x_1,y_1,z_1),\ldots,(x_n,y_n,z_n)$. One procedure for fitting a curve to the points is to let $s_1=0$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2}$ for $i=2,\ldots,n$, and then to find splines $x(s), y(s), z(s)$ with tension σ that satisfy $x(s_i) = x_i$, $y(s_i) = y_i$, and $z(s_i) = z_i$ for $i = 1,\ldots,n$. If (x_1',y_1',z_1') and (x_n',y_n',z_n') are the desired slopes for the curve $s \rightarrow (x(s), y(s), z(s))$ at the points (x_1,y_1,z_1) and (x_n,y_n,z_n), then the splines $x(s), y(s), z(s)$ can be selected so that $x'(s_i) = x_i'$, $y'(s_i) = y_i'$, and $z'(s_i) = z_i$ for $i = 1,n$. The curve $s \rightarrow (x(s), y(s), z(s))$ then passes through the points (x_i, y_i, z_i) and has the required slopes at the end points. The subroutine QURV1 is available for obtaining the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1,\ldots,n)$ which characterize this curve, and the subroutine QURV2 is available for computing the curve.

<u>CALL QURV1 (n,X,Y,Z,SLP1X,SLP1Y,SLP1Z,SLPNX,SLPNY,SLPNZ, IND,DDX,DDY,DDZ,TEMP,S,σ,IERR)</u>

X, Y, and Z are arrays containing the x-coordinates $x_1,\ldots,x_n$, y-coordinates $y_1,\ldots,y_n$, and z-coordinates $z_1,\ldots,z_n$. It is assumed that $n \geqslant 2$ and that the points (x_i, y_i, z_i) are indexed in the order that they are to be traversed by the curve. It is also assumed that $(x_i, y_i, z_i) \neq (x_{i+1}, y_{i+1}, z_{i+1})$ for $i = 1,\ldots,n-1$.

SLP1X, SLP1Y, SLP1Z and SLPNX, SLPNY, SLPNZ are assigned the values x_1', y_1', z_1' and x_n', y_n', z_n'. The user may omit values for SLP1X, SLP1Y, SLP1Z and/or SLPNX, SLPNY, SLPNZ. The argument IND specifies the information that is provided.

IND = 0	Values are supplied for SLP1X, SLP1Y, SLP1Z and SLPNX, SLPNY, SLPNZ.
IND = 1	Values are supplied for SLP1X, SLP1Y, SLP1Z but not for SLPNX, SLPNY, SLPNZ.
IND = 2	Values are supplied for SLPNX, SLPNY, SLPNZ but not for SLP1X, SLP1Y, SLP1Z .
IND = 3	No values are supplied for SLP1X, SLP1Y, SLP1Z and SLPNX, SLPNY, SLPNZ.

If a value is not supplied by the user, then the routine provides a value.

σ is the tension factor to be employed. If $|\sigma|$ is small, say $|\sigma| < 10^{-3}$, then $x(s), y(s), z(s)$ approximate cubic splines. However, if $|\sigma|$ is large, say $|\sigma| > 100$, then the resulting curve will approximate the polygonal line from (x_1, y_1, z_1) to (x_n, y_n, z_n).

IERR is an integer variable and S, DDX, DDY, DDZ are arrays of dimension n or larger. When QURV1 is called, if no input errors are detected then IERR is assigned the value 0 and the values $s_1,\ldots,s_n$ are computed and stored in S. Also, the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1,\ldots,n)$ are computed and stored in DDX, DDY, DDZ.

TEMP is an array of dimension n or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1 if $n < 2$

IERR = 2 if $(x_i, y_i, z_i) = (x_{i+1}, y_{i+1}, z_{i+1})$ for some i

When either of these errors is detected, the routine immediately terminates.

Remark. X, Y, Z, n, SLP1X, SLP1Y, SLP1Z, SLPNX, SLPNY, SLPNZ, IND, and σ are not modified by QURV1.

Programming. QURV1 employs the subroutines CEEZ, TERMS, and SNHCSH. QURV1, CEEZ, and TERMS were written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL QURV2(t,XT,YT,ZT,n,X,Y,Z,DDX,DDY,DDZ,S,σ)

X, Y, and Z are arrays containing the x-coordinates $x_1,\dots,x_n$, y-coordinates $y_1,\dots,y_n$, and z-coordinates $z_1,\dots,z_n$. S is an array containing $s_1,\dots,s_n$ and DDX, DDY, DDZ are arrays containing the second derivatives $x''(s_i), y''(s_i), z''(s_i)$ $(i = 1,\dots,n)$.

Now consider the change of variables $t = s/s_n$ and let $t \to (\overline{x}(t), \overline{y}(t), \overline{z}(t))$ denote the curve in terms of the new parameter t. XT, YT, ZT are real variables. For any $0 \leqslant t \leqslant 1$, QURV2 computes the point $(\overline{x}(t), \overline{y}(t), \overline{z}(t))$ on the curve and assigns XT, YT, ZT the values $\overline{x}(t), \overline{y}(t), \overline{z}(t)$.

Remark. After DDX, DDY, DDZ have been obtained, QURV2 may be repeatedly called to evaluate the curve at different points so long as the tension factor σ remains fixed. However, if σ is modified, then it should be emphasized that the derivative information in DDX, DDY, DDZ will have to be recomputed before QURV2 can be used with the new tension factor.

Programming. QURV2 employs the function INTRVL and subroutine SNHCSH. QURV2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin).

B-SPLINES

For $k \geqslant 1$ let $f(t) = (t-x)^{k-1}$ when $t > x$ and $f(t) = 0$ when $t \leqslant x$. Then for any sequence $t_i \leqslant \cdots \leqslant t_{i+k}$ where $t_i < t_{i+k}$, let $B_{ik}(x) = (t_{i+k} - t_i)\, f[t_i, \ldots, t_{i+k}]$ where $f[t_i, \ldots, t_{i+k}]$ is the k^{th} order divided difference of $f(t)$. The function B_{ik} is called a ***B-spline of order k***. For $k = 1$ it follows that

$$B_{i1}(x) = \begin{cases} 1 & \text{if } t_i \leqslant x < t_{i+1} \\ 0 & \text{otherwise} \end{cases} .$$

More generally, for $k \geqslant 2$ $\quad B_{ik}(x) = 0$ when $x \notin [t_i, t_{i+k})$. For $t_i \leqslant x < t_{i+k}$

$$B_{ik}(x) = \left(\frac{t_{i+k} - x}{t_{i+k} - t_i}\right)^{k-1} \qquad \text{when } t_i = \cdots = t_{i+k-1} \text{ and}$$

$$B_{ik}(x) = \left(\frac{x - t_i}{t_{i+k} - t_i}\right)^{k-1} \qquad \text{when } t_{i+1} = \cdots = t_{i+k}.$$

Otherwise, if no point appears more than $k-1$ times in the sequence $\{t_i, \ldots, t_{i+k}\}$ then

$$B_{ik}(x) = \frac{x - t_i}{t_{i+k-1} - t_i}\, B_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}}\, B_{i+1,k-1}(x).$$

From these relations it follows that $B_{ik}(x) > 0$ when $t_i < x < t_{i+k}$. Now let $\xi_1 < \cdots < \xi_r$ be the distinct points in $\{t_i, \ldots, t_{i+k}\}$, where ξ_j appears m_j times for $j = 1, \ldots, r$. Then it can be verified that B_{ik} is a polynomial of order $\leqslant k$ (degree $\leqslant k-1$) on each interval $[\xi_j, \xi_{j+1})$ $(j = 1, \ldots, r-1)$, and that B_{ik} is of class C^{k-m_j-1} at ξ_j for $j = 1, \ldots, r$.

We note in passing that if t_i $(i = 0, \pm 1, \pm 2, \ldots)$ is a sequence where $t_i \leqslant t_{i+1}$ and $t_i < t_{i+k}$ for each i, then for any $x \in [t_j, t_{j+1})$, $B_{ik}(x) \neq 0$ only when $i = j-k+1, \ldots, j$. Moreover, it can be verified that $\Sigma_i B_{ik}(x) = 1$.

Now let $\xi_1 < \cdots < \xi_{\ell+1}$ be a sequence of points, which we shall call ***knots*** or ***break points***. If $\ell \geqslant 2$ then $\xi_2, \ldots, \xi_\ell$ will be called the ***interior knots***. For each interior knot ξ_j let there be associated an integer $m_j \geqslant 1$, called the ***multiplicity*** of the knot. Then for any $k \geqslant \max\{m_2, \ldots, m_\ell\}$ let $t_1 \leqslant \cdots \leqslant t_{n+k}$ $(n = k + m_2 + \cdots + m_\ell)$ be any sequence where

(1) $t_1 \leqslant \cdots \leqslant t_k = \xi_1$,

(2) $t_{k+1}, \ldots, t_n$ are the interior knots, where each interior knot ξ_j appears exactly m_j times, and

(3) $\xi_{\ell+1} = t_{n+1} \leqslant \cdots \leqslant t_{n+k}$.

Otherwise, if $\ell = 1$ then for $k \geqslant 1$ let $t_1 \leqslant \cdots \leqslant t_{n+k}$ $(n = k)$ be any sequence where $t_1 \leqslant \cdots \leqslant t_k = \xi_1$ and $\xi_2 = t_{n+1} \leqslant \cdots \leqslant t_{n+k}$. Then for $\ell \geqslant 1$, we note that $B_{1k}, \ldots, B_{nk}$ are the only B-splines of order k which need not be 0 on the interval $[t_k, t_{n+1})$. Let $\mathscr{P}_k[t_k, \ldots, t_{n+1}]$ denote the collection of all piecewise polynomials $p(x)$ defined on the interval $[t_k, t_{n+1})$ where $p(x)$ is a polynomial of order $\leqslant k$ (degree $\leqslant k-1$) on $[\xi_j, \xi_{j+1})$ for $j = 1, \ldots, \ell$, and $p(x)$ is of class C^{k-m_j-1} at each interior knot ξ_j where $m_j < k$. Then by the above remarks $B_{1k}, \ldots, B_{nk}$ are in $\mathscr{P}_k[t_k, \ldots, t_{n+1}]$. Also it can be verified that $B_{1k}, \ldots, B_{nk}$ form a basis for the vector space $\mathscr{P}_k[t_k, \ldots, t_{n+1}]$. Thus any piecewise polynomial $p(x)$ in $\mathscr{P}_k[t_k, \ldots, t_{n+1}]$ can be represented uniquely in the form $p(x) = \sum_{i=1}^{n} a_i B_{ik}(x)$ for $t_k \leqslant x < t_{n+1}$. This representation is called a ***B-spline representation for p(x).***

PIECEWISE POLYNOMIAL INTERPOLATION

For $n \geq k \geq 1$ let $t_1 \leq \cdots \leq t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1, \ldots, n$. Further assume that $t_k < t_{k+1}$ and $t_n < t_{n+1}$, and consider a set of n points $\{(x_i, y_i) : i = 1, \ldots, n\}$ where $t_k \leq x_1 < \cdots < x_n \leq t_{n+1}$. Then we wish to find a piecewise polynomial $f = \sum_{i=1}^{n} a_i B_{ik}$ defined on the interval $[t_k, t_{n+1})$ which satisfies $f(x_i) = y_i$ for $i = 1, \ldots, n$. [If $x_n = t_{n+1}$ then by $f(x_n) = y_n$ we mean $f(x_n -) = y_n$.] This problem has a unique solution when $x_1 < t_{k+1}$, $t_i < x_i < t_{i+k}$ for $1 < i < n$, and $x_n > t_n$. The following subroutine is available for obtaining the coefficients $a_1, \ldots, a_n$ of the interpolating piecewise polynomial.

<u>CALL BSTRP(X,Y,T,n,k,A,WK,IFLAG)</u>

X is an array containing $x_1, \ldots, x_n$, Y an array containing $y_1, \ldots, y_n$, and T an array containing $t_1, \ldots, t_{n+k}$. A is an array of dimension n or larger, and IFLAG an integer variable. On an initial call to the routine the user may assign IFLAG any nonzero value. In this case, if no errors are detected then IFLAG is reset by the routine to 0 and the B-spline coefficients $a_1, \ldots, a_n$ are computed and stored in A. The routine may be recalled with IFLAG = 0 on input when only Y is modified. In this case, no error checking is performed and IFLAG = 0 on output. Also the B-spline coefficients $a_1, \ldots, a_n$ of the new interpolating piecewise polynomial are computed and stored in A.

WK is an array of dimension $(2k-1)n$ or larger that is used for temporary storage by the routine. When BSTRP terminates, WK contains information needed for subsequent calls to the routine.

Error Return. IFLAG is assigned the value 1 if a violation of any of the conditions

$$x_1 < \cdots < x_n$$
$$t_k \leq x_1 < t_{k+1}$$
$$t_i < x_i < t_{i+k} \quad \text{for } 1 < i < n$$
$$t_n < x_n \leq t_{n+1}$$

is detected. When an error is detected, the routine immediately terminates.

Example. Given $n > 4$ data points (x_i, y_i), then for $k = 4$ one may set $t_1 = \cdots = t_k = x_1$, $t_{k+i} = x_{i+2}$ for $i = 1, \ldots, n-k$, and $x_n = t_{n+1} = \cdots = t_{n+k}$. Then $x_3, \ldots, x_{n-2}$ are the interior knots for the interpolating piecewise polynomial f. Here we have cubic spline interpolation where the data points x_2 and x_{n-1} are not knots for f.

Selection of $t_i \leq \cdots \leq t_{n+k}$ given the data (x_i, y_i). It is recommended that one set $t_1 = \cdots = t_k$ and $t_{n+1} = \cdots = t_{n+k}$. For $k \geq 2$ it is frequently convenient to select $n - k$ points in $x_2, \ldots, x_{n-1}$ to be the interior knots for f. (This was done in the above example.) If $k > 2$ then an alternative approach, which often gives excellent results, is to set $t_{k+i} = (x_{i+1} + \cdots + x_{i+k-1})/(k-1)$ for $i = 1, \ldots, n-k$.

Remark. After the B-spline representation $\Sigma_i a_i B_{ik}$ is obtained, then the subroutine BSPP can be used to obtain the Taylor series representation. The Taylor series representation is what is normally used for evaluating piecewise polynomials.

Programming. BSTRP calls the subroutines BSPEV, BANFAC, and BANSLV. BSTRP is a modified version by A. H. Morris of the subroutine SPLINT. The routines SPLINT, BANFAC, and BANSLV were written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines,*** Springer-Verlag, 1978.

CONVERSION OF PIECEWISE POLYNOMIALS FROM B-SPLINE TO TAYLOR SERIES FORM

For $n \geqslant k \geqslant 1$ let $t_1 \leqslant \cdots \leqslant t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1, \dots, n$. Further assume that $t_k < t_{n+1}$ and let $f(x) = \sum_{i=1}^{n} a_i B_{ik}(x)$ for $t_k \leqslant x < t_{n+1}$. If $\xi_1 < \cdots < \xi_{\ell+1}$ are the distinct points in the sequence $\{t_k, \dots, t_{n+1}\}$ then the piecewise polynomial f can be represented in the form $f(x) = \sum_{i=1}^{k} c_{ij}(x - \xi_j)^{i-1}$ for $\xi_j \leqslant x < \xi_{j+1}$ $(j = 1, \dots, \ell)$. The following subroutine is available for obtaining the coefficients c_{ij} of this representation.

CALL BSPP(T,A,n,k,BREAK,C,L,WK)

T is an array containing $t_1, \dots, t_{n+k}$ and A an array containing $a_1, \dots, a_n$. BREAK is an array of dimension $\ell + 1$ or larger, C a 2-dimensional array of dimension $k \times \ell$, and L a variable. When BSPP is called then L is assigned the value ℓ (which is computed by the routine), the break points $\xi_1 < \cdots < \xi_{\ell+1}$ are found and stored in BREAK, and the coefficients c_{ij} are computed and stored in C. The jth column of the matrix C then contains the coefficients of the jth polynomial forming f $(j = 1, \dots, \ell)$.

WK is an array of dimension $k(k+1)$ or larger that is used for a work space by the routine.

Remarks

(1) Since $\ell \leqslant n - k + 1$, BREAK may be declared to be of dimension $n - k + 2$ and C to be of dimension $k \times (n - k + 1)$.

(2) After C is obtained, then PPVAL may be used to evaluate f at any point x.

Programming. BSPP is a reformulation by A. H. Morris of the subroutine BSPLPP, written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

PIECEWISE POLYNOMIAL EVALUATION

Given $x_1 < \cdots < x_{\ell+1}$ and let f be a piecewise polynomial where $f(x) = \sum_{i=1}^{k} c_{ij}(x - x_j)^{i-1}$ for $x_j \leqslant x < x_{j+1}$ $(j = 1, \ldots, \ell)$. Then the following subroutine is available for computing f at any point x.

<u>CALL PPVAL (X,C,k,ℓ,XI,YI,m)</u>

X is an array containing the knots $x_1, \ldots x_\ell$ and C a $k \times \ell$ matrix containing the coefficients c_{ij}. It is assumed that $k \geqslant 1$ and $\ell \geqslant 1$. Let $\overline{x}_1, \ldots, \overline{x}_m$ be the points at which f is to be evaluated. XI is an array containing $\overline{x}_1, \ldots, \overline{x}_m$ and YI is an array of dimension m or larger. When PPVAL is called, $f(\overline{x}_j)$ is computed and stored in YI(j) for $j = 1, \ldots, m$.

Remarks

(1) X need not contain the knot $x_{\ell+1}$.

(2) It is not required that f be continuous at an interior knot x_i. If x_i appears in XI then $f(x_i+)$ is computed.

(3) It is not required that the output points $\overline{x}_j$ in XI be in the interval $[x_1, x_{\ell+1})$. If $\overline{x}_j < x_1$ then $\Sigma_i c_{i1}(x - x_1)^{i-1}$ is evaluated at $\overline{x}_j$. Otherwise, if $\overline{x}_j \geqslant x_{\ell+1}$ then $\Sigma_i c_{i\ell}(x - x_\ell)^{i-1}$ is evaluated at $\overline{x}_j$.

Programming. PPVAL is an adaptation by A. H. Morris of code written by Rondall E. Jones (Sandia Laboratories).

WEIGHTED LEAST SQUARES PIECEWISE POLYNOMIAL FITTING

For $n \geqslant k \geqslant 1$ let $t_1 \leqslant \cdots \leqslant t_{n+k}$ be a sequence where $t_i < t_{i+k}$ for $i = 1,\dots,n$. Further assume that $t_k < t_{k+1}$ and $t_n < t_{n+1}$, and consider a set of points $\{(x_i,y_i) : i = 1,\dots,m\}$ where $t_k \leqslant x_1 \leqslant \cdots \leqslant x_m \leqslant t_{n+1}$. Let $w_i > 0$ $(i = 1,\dots,m)$ be weights. Then the subroutine BSL2 is available for finding a piecewise polynomial $f = \sum_{i=1}^{n} a_i B_{ik}$ defined on the interval $[t_k, t_{n+1})$ which minimizes $\sum_{i=1}^{m} w_i (f(x_i) - y_i)^2$.[1]

CALL BSL2 (T,n,k,X,Y,WGT,m,A,WK,Q,IERR)

T is an array containing $t_1,\dots,t_{n+k}$, X an array containing $x_1,\dots,x_m$, Y an array containing $y_1,\dots,y_m$, and WGT an array containing $w_1,\dots,w_m$. A is an array of dimension n or larger, and IERR an integer variable. When BSL2 is called, if no input errors are detected then IERR is set to 0 and the B-spline coefficients $a_1,\dots,a_n$ of the least squares approximation f are computed and stored in A.

WK is an array of dimension n or larger, and Q an array of dimension kn or larger. WK and Q are work spaces for the routine.

Error Return. IERR is assigned the value 1 if any of the conditions

$n \geqslant k \geqslant 1$

$t_n < t_{n+1}$

$t_k \leqslant x_1 \leqslant \cdots \leqslant x_m \leqslant t_{n+1}$

is violated. When an error is detected, the routine immediately terminates.

Selection of $t_1 \leqslant \cdots \leqslant t_{n+k}$ given the data (x_i, y_i). It is recommended that the knots t_ν be selected so that there are data points $x_{i_1} < \cdots < x_{i_n}$ satisfying

$t_k \leqslant x_{i_1} < t_{k+1}$

$t_\nu < x_{i_\nu} < t_{\nu+k}$ for $\nu = 2,\dots,n$.

If these conditions are satisfied then the least squares approximation is unique.

Remark. After the B-spline representation $\Sigma_i a_i B_{ik}$ of the least squares approximation is obtained, then the subroutine BSPP can be used to obtain the Taylor series representation. The Taylor series representation is what is normally used for evaluating piecewise polynomials.

Programming. BSL2 calls the subroutines BSPEV, BCHFAC, and BCHSLV. BSL2 is a modified version by A. H. Morris of the subroutine L2APPR. The routines L2APPR, BCHFAC, and BCHSLV were written by Carl de Boor (University of Wisconsin).

Reference. de Boor, Carl, ***A Practical Guide to Splines***, Springer-Verlag, 1978.

[1]If $x_i = t_{n+1}$ then by $f(x_i)$ we mean $f(x_i-)$.

BI-SPLINES UNDER TENSION

Given a tension factor σ and a set $X = \{x_1,...,x_m\}$ where $x_1 < \cdots < x_m$. Let $S_\sigma(X)$ denote the collection of splines having tension σ and the knots $x_1,...,x_m$. Then $S_\sigma(X)$ is a vector space. Also each spline $f(x)$ in $S_\sigma(X)$ is uniquely characterized by the values $f(x_1),...,f(x_m)$ and the slopes $f'(x_1)$ and $f'(x_m)$. Let $\psi_i(x)$ $(i = 1,...,m)$ denote the spline in $S_\sigma(X)$ satisfying

$$\begin{aligned} &\psi_i(x_i) = 1 \\ &\psi_i(x_k) = 0 \quad \text{for} \quad k \neq i \\ &\psi_i'(x_1) = \psi_i'(x_m) = 0 \end{aligned}$$

and let ψ_{m+1}, ψ_{m+2} be the splines satisfying

$$\begin{aligned} &\psi_{m+1}(x_i) = 0 \quad \psi_{m+2}(x_i) = 0 \quad (i = 1,...,m) \\ &\psi_{m+1}'(x_1) = 1 \quad \psi_{m+2}'(x_1) = 0 \\ &\psi_{m+1}'(x_m) = 0 \quad \psi_{m+2}'(x_m) = 1. \end{aligned}$$

Then $\{\psi_1,...,\psi_{m+2}\}$ is a basis for $S_\sigma(X)$ and $f = \sum_{i=1}^{m} f(x_i)\psi_i + f'(x_1)\psi_{m+1} + f'(x_m)\psi_{m+2}$ for each spline $f(x)$ in $S_\sigma(X)$.

Let $Y = \{y_1,...,y_n\}$ where $y_1 < \cdots < y_n$ and let $\{\overline{\psi}_1,...,\overline{\psi}_{n+2}\}$ be the corresponding basis for $S_\sigma(Y)$. Then we note that there exists an unique surface

$$F(x,y) = \sum_{i=1}^{m+2} \sum_{j=1}^{n+2} a_{ij}\psi_i(x)\overline{\psi}_j(y)$$

in the tensor product space $S_\sigma(X) \otimes S_\sigma(Y)$ which satisfies the conditions

$$(*) \quad \begin{aligned} &F(x_i,y_j) = f(x_i,y_j) && i = 1,...,m \quad j = 1,...,n \\ &\left.\begin{aligned} D_2F(x_i,y_1) = D_2f(x_i,y_1) \\ D_2F(x_i,y_n) = D_2f(x_i,y_n) \end{aligned}\right\} && i = 1,...,m \\ &\left.\begin{aligned} D_1F(x_1,y_j) = D_1f(x_1,y_j) \\ D_1F(x_m,y_j) = D_1f(x_m,y_j) \end{aligned}\right\} && j = 1,...,n \\ &D_1D_2F(x_k,y_\ell) = D_1D_2f(x_k,y_\ell) \quad k = 1, m \quad \ell = 1, n \end{aligned}$$

for given data $f(x_i,y_j)$, $D_2f(x_i,y_1),...,D_1D_2f(x_k,y_\ell)$.[1] Such a surface is called a ***bi-spline with tension*** σ. It is easily checked that the bi-spline $F(x,y)$ has the following properties:

(1) $F(x,y)$ is a C^2 mapping on $[x_1,x_m] \times [y_1,y_n]$.

(2) The partial derivatives $D_1^2D_2^2F(x,y)$ and $D_2^2D_1^2F(x,y)$ exist and are continuous, and $D_1^2D_2^2F(x,y) = D_2^2D_1^2F(x,y)$.

(3) For each fixed y the mapping $x \to F(x,y)$ is a spline in $S_\sigma(X)$, and for each fixed x the mapping $y \to F(x,y)$ is a spline in $S_\sigma(Y)$.

For a given tension factor σ, let F_σ denote the unique bi-spline in $S_\sigma(X) \otimes S_\sigma(Y)$ which satisfies conditions (*). If $\sigma = 0$ then F_σ is the standard bicubic spline. Also it can be verified that when $\sigma \to \infty$, F_σ converges uniformly on $[x_1,x_m] \times [y_1,y_n]$ to the piecewise bilinear function $\ell(x,y)$ where

[1] D_1F and D_2F denote the partial derivatives of F.

$$\ell(x,y) = f(x_i,y_j)\,\frac{x_{i+1}-x}{h_i}\,\frac{y_{j+1}-y}{k_j} + f(x_i,y_{j+1})\,\frac{x_{i+1}-x}{h_i}\,\frac{y-y_j}{k_j}$$

$$+f(x_{i+1},y_j)\,\frac{x-x_i}{h_i}\,\frac{y_{j+1}-y}{k_j} + f(x_{i+1},y_{j+1})\,\frac{x-x_i}{h_i}\,\frac{y-y_j}{k_j}$$

for $x_i \leqslant x \leqslant x_{i+1}$ and $y_j \leqslant y \leqslant y_{j+1}$. Here $h_i = x_{i+1} - x_i$ and $k_j = y_{j+1} - y_j$.

BI-SPLINE UNDER TENSION SURFACE INTERPOLATION

Given $x_1 < \cdots < x_m$ and $y_1 < \cdots < y_n$. Also assume that we are given the values z_{ij} $(i = 1,\ldots,m;\ j = 1,\ldots,n)$ and a tension factor σ. Then the subroutine SURF is available for finding a bi-spline $F(x,y)$ with tension σ that satisfies $F(x_i,y_j) = z_{ij}$ for each i,j. Boundary conditions can be imposed on the surface $F(x,y)$ if desired.

CALL SURF (m,n,X,Y,Z,kz,OPT,DDZ,WK,σ,IERR)

X is an array containing $x_1,\ldots,x_m$, Y an array containing $y_1,\ldots,y_n$, and Z the $m \times n$ matrix (z_{ij}). The argument kz is the number of rows in the dimension statement for Z in the calling program. It is assumed that $m \geq 2$, $n \geq 2$, and $kz \geq m$.

OPT is an array, called the ***option vector***, which permits the user to specify any boundary conditions that are to be imposed on the surface. If no boundary conditions are to be specified then OPT may be declared to be of dimension 1 and OPT(1) must be assigned the value 0. The details concerning the specification of boundary conditions in OPT are given below.

DDZ is a 3-dimensional array of dimension $m \times n \times 3$ and IERR is a variable. When SURF is called, if no input errors are detected then IERR is assigned the value 0 and the partial derivatives $D_1^2F(x_i,y_j)$, $D_2^2F(x_i,y_j)$, $D_1^2D_2^2F(x_i,y_j)$ $(i = 1,\ldots,m;\ j = 1,\ldots,n)$ are computed and stored in DDZ. $DDZ(i,j,1) = D_2^2F(x_i,y_j)$, $DDZ(i,j,2) = D_1^2F(x_i,y_j)$, and $DDZ(i,j,3) = D_1^2D_2^2F(x_i,y_j)$ for each i,j.

WK is an array of dimension $m + 2n$ or larger that is used for a work space.

Error Return. IERR reports the following input errors:

IERR = 1	if $m < 2$ or $n < 2$
IERR = 2	if $x_1 < \cdots < x_m$ or $y_1 < \cdots < y_n$ is not satisfied
IERR = 3	if OPT contains an error.

When an error is detected, the routine immediately terminates.

Remark. After DDZ is obtained then SURF2 and NSURF2 may be used to evaluate the bi-spline $F(x,y)$.

The option vector OPT. If no boundary conditions are to be imposed then OPT may be declared to be of dimension 1 and OPT(1) must have the value 0. Otherwise, OPT is an array containing the information key_1, $data_1$, key_2, $data_2,\ldots,$ key_s, $data_s$, 0. The last entry in OPT is the value 0. Each group of data key_i, $data_i$ $(i = 1,\ldots,s)$ is called an ***option***. Each key_i is an integer and $data_i$ a list of partial derivative values that the surface $F(x,y)$ is required to satisfy. The following options are available:

key = 1	The values $D_1F(x_1, y_j)$ ($j = 1,...,n$) must be satisfied.
key = 2	The values $D_1F(x_m, y_j)$ ($j = 1,...,n$) must be satisfied.
key = 3	The values $D_2F(x_i, y_1)$ ($i = 1,...,m$) must be satisfied.
key = 4	The values $D_2F(x_i, y_n)$ ($i = 1,...,m$) must be satisfied.
key = 5	The value $D_1D_2F(x_1, y_1)$ must be satisfied.
key = 6	The value $D_1D_2F(x_m, y_1)$ must be satisfied.
key = 7	The value $D_1D_2F(x_1, y_n)$ must be satisfied.
key = 8	The value $D_1D_2F(x_m, y_n)$ must be satisfied.

The order of the options in OPT is arbitrary. If an unrecognized key is used then the error indicator IERR is assigned the value 3 and the routine terminates.

Example. Assume that we have an array DY1 containing values $D_2F(x_i, y_1)$ ($i = 1,...,m$) which are to be satisfied, and that we also want $D_1D_2F(x_m, y_n) = -1.3$ to be satisfied. Then OPT must be of dimension $\geqslant m+4$ and OPT can be defined as follows:

```
      OPT (1) = 3.0              (First option)
      DO 10 I = 1,M
10    OPT (I + 1) = DY1(I)
      OPT (M + 2) = 8.0          (Second option)
      OPT (M + 3) = -1.3
      OPT (M + 4) = 0.0          (Terminates the option vector)
```

Background. The evaluation of $D_1^2F(x_i, y_j)$, $D_2^2F(x_i, y_j)$, and $D_1^2D_2^2F(x_i, y_j)$ reduce to the evaluation of second derivatives of splines. Specifically, for each $i \leqslant m$ $D_2^2F(x_i, y_1),...,D_2^2F(x_i, y_n)$ are the second derivatives that characterize the spline $y \rightarrow F(x_i, y)$, and for each $j \leqslant n$ $D_1^2F(x_i, y_j),...,D_1^2F(x_m, y_j)$ are the second derivatives that characterize the spline $x \rightarrow F(x, y_j)$. Also $D_1D_2^2F(x_1, y_j)$ and $D_1D_2^2F(x_m, y_j)$ ($j = 1,...,n$) are the second derivatives that characterize the splines $y \rightarrow D_1F(x_1, y)$ and $y \rightarrow D_1F(x_m, y)$. For each $j \leqslant m$, after one obtains the values $D_2^2F(x_i, y_j)$ through which the spline $x \rightarrow D_2^2F(x, y_j)$ will pass and the end slopes $D_1D_2^2F(x_1, y_j)$ and $D_1D_2^2F(x_m, y_j)$ which this spline must have, then the second derivatives that characterize this spline can be computed. $D_1^2D_2^2F(x_1, y_j),...,D_1^2D_2^2F(x_m, y_j)$ are the second derivatives that characterize $x \rightarrow D_2^2F(x, y_j)$.

Programming. SURF employs the subroutines CEEZ, TERMS, and SNHCSH. SURF was written by A. K. Cline and R. J. Renka (University of Texas at Austin), and modified by A. H. Morris.

BI-SPLINE UNDER TENSION EVALUATION

Given $x_1 < \cdots < x_m$ and $y_1 < \cdots < y_n$, and let F(x, y) be a bi-spline with tension σ. If the partial derivatives $D_1^2 F(x_i, y_j)$, $D_2^2 F(x_i, y_j)$, $D_1^2 D_2^2 F(x_i, y_j)$ are known for $i = 1,\dots,m$ and $j = 1,\dots,n$, then the function SURF2 may be used for evaluating F(x, y) at a single point, and the subroutine NSURF2 may be used for evaluating F(x, y) on a grid of points.

SURF2(s,t,m,n,X,Y,Z,kz,DDZ,σ)

X is an array containing $x_1,\dots,x_m$, Y an array containing $y_1,\dots,y_n$, and Z an $m \times n$ matrix containing the values $F(x_i, y_j)$. The argument kz is the number of rows in the dimension statement for Z in the calling program. It is assumed that $m \geqslant 2$, $n \geqslant 2$, and $kz \geqslant m$.

DDZ is a 3-dimensional array of dimension $m \times n \times 3$ containing the partial derivatives where

$$\begin{aligned} DDZ(i,j,1) &= D_2^2F(x_i, y_j) \\ DDZ(i,j,2) &= D_1^2F(x_i, y_j) \\ DDZ(i,j,3) &= D_1^2D_2^2F(x_i, y_j) \end{aligned}$$

for each i, j. SURF2(s, t, m, n, X, Y, Z, kz, DDZ, σ) = F(s, t) for any point (s, t).

Remark. After DDZ has been obtained, SURF2 may be repeatedly called to evaluate the surface at different points so long as the tension factor σ remains fixed. However, if σ is modified then it should be emphasized that the derivative information in DDZ will have to be recomputed before SURF2 can be used with the new tension factor.

Programming. SURF2 employs the function INTRVL and subroutine SNHCSH. SURF2 was written by A. K. Cline and R. J. Renka (University of Texas at Austin).

CALL NSURF2(s_{min},s_{max},m_s,t_{min},t_{max},n_t,W,kw,m,n,
X,Y,Z,kz,DDZ,WORK,σ)

The arguments s_{min} and s_{max} are the lower and upper limits of the x-coordinates of the grid on which F(x, y) is to be evaluated, and the arguments t_{min} and t_{max} are the lower and upper limits of the y-coordinates. The purpose of the routine is to evaluate the surface at the points (s_i, t_j) where

$$s_i = s_{min} + (i-1)\frac{s_{max} - s_{min}}{m_s - 1}$$

$$t_j = t_{min} + (j-1)\frac{t_{max} - t_{min}}{n_t - 1}$$

for $i = 1,\dots,m_s$ and $j = 1,\dots,n_t$. It is assumed that $m_s \geqslant 1$ and $n_t \geqslant 1$.

W is a 2-dimensional array of dimension $kw \times n_t$ where $kw \geqslant m_s$. When NSURF2 is called W(i, j) is assigned the value $F(s_i, t_j)$ for $i = 1,\ldots,m_s$ and $j = 1,\ldots,n_t$.

The arguments m, n, X, Y, Z, kz, DDZ, σ are the same as in SURF2. WORK is an array of dimension $4m_s$ or larger that is used for a work space.

Programming. NSURF2 employs the subroutine SNHCSH. NSURF2 was written by A. K. Cline (University of Texas at Austin).

SURFACE INTERPOLATION FOR ARBITRARILY POSITIONED DATA POINTS

Let $\{(x_i, y_i, z_i) : i = 1,...,n\}$ be a set of 4 or more points which are not collinear. If $(x_i, y_i) \neq (x_j, y_j)$ for $i \neq j$ then the problem is to find a smooth mapping $z = F(x, y)$ for which $z_i = F(x_i, y_i)$ for $i = 1,...,n$. The desired degree of smoothness might vary, but it is almost always required that $F(x, y)$ be at least continuously differentiable.

A procedure for constructing a smooth mapping $F(x, y)$ generally contains the following components:

(1) An algorithm for forming a triangular grid for the convex hull of $\{(x_i, y_i) : i = 1,...,n\}$. The data points (x_i, y_i) are the vertices of the triangular cells of the grid.

(2) A procedure for estimating the first (and possibly higher order) partial derivatives of $F(x, y)$ at the data points (x_i, y_i). There is currently no known best method for performing this task. At a point (x_i, y_i), it is agreed that the derivative estimation should depend not only on z_i, but also on z_j for neighboring points (x_j, y_j). However, the number of neighboring points that should be used in the derivative estimation is normally unclear.

(3) For any point (x, y) in the grid, a routine for finding the triangular cell which contains the point. If extrapolation is to be permitted, then the region outside of the grid must be partitioned and a routine provided for locating any point which lies outside the grid.

(4) A smooth interpolating algorithm for evaluating $F(x, y)$ on each triangular cell of the grid. If extrapolation is to be permitted, then an algorithm must also be provided to compute $F(x, y)$ on each cell of the partitioned region outside of the grid.

Generally, the derivative estimation appears to be the most ad hoc portion of most smooth interpolating procedures. This quite probably is unavoidable at the present time. However, it is unfortunate since the manner in which the derivative estimation is performed can significantly effect the results obtained from any interpolating procedure.

The subroutines BVIP and BVIP2 are available for obtaining a continuously differentiable surface $z = F(x, y)$ for which $z_i = F(x_i, y_i)$ for $i = 1,...,n$. Extrapolation is allowed, and the user is permitted to specify (via the argument n_c) the number of neighboring points to be used for derivative estimation. BVIP is used if $F(x, y)$ is to be evaluated on an arbitrary collection of output points, whereas BVIP2 is applicable only if $F(x, y)$ is to be evaluated on a rectangular grid of output points. If BVIP is employed to evaluate $F(x, y)$ in a rectangular grid, then BVIP will produce the same results as BVIP2 but it will be less efficient.

CALL BVIP(MO,n_c,n,X,Y,Z,m,XI,YI,ZI,IWK,WK,IERR)

X is an array containing $x_1,...,x_n$, Y an array containing $y_1,...,y_n$, and Z an array containing $z_1,...,z_n$. The input argument n_c is the number of neighboring points to be used for derivative estimation. It is assumed that $2 \leq n_c < n$ and $n_c \leq 25$. Currently no theory is available for indicating how n_c should be set. The only comment that can be made is that setting n_c to 3, 4, or 5 normally produces satisfactory results.

It is assumed that F(x, y) is to be evaluated at the points $(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_m, \bar{y}_m)$. XI is an array containing $\bar{x}_1, \ldots, \bar{x}_m$, YI an array containing $\bar{y}_1, \ldots, \bar{y}_m$, and ZI an array of dimension m or larger. When BVIP is called, if no input errors are detected then $F(\bar{x}_i, \bar{y}_i)$ is computed and stored in ZI(i) for i = 1,...,m.

One may wish to recall BVIP a number of times to compute F(x, y) for different sets of points. If recalls are needed then a portion of the information that is generated on the first call to BVIP can frequently be reused. The reuse of information is controlled by the input argument MO. MO must have the value 1 on the first call to BVIP. For subsequent calls MO may be assigned the following values:

MO = 1	This setting must be employed when any of the data n_c, n, X, Y is modified. In this case, none of the previously generated information can be reused.
MO = 2	This setting may be used when n_c, n, X, Y are not modified.
MO = 3	This setting is permissible when only Z is modified.

If MO $\neq$ 1 then the contents of IWK and WK must not be altered.

IWK is an array of dimension kn + m or larger where $k = \max\{31, 27 + n_c\}$, and WK is an array of dimension 8n or larger. IWK and WK are storage areas for the routine.

Error Return. IERR is an integer variable. If no errors are detected then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	MO is not 1, 2, or 3.
IERR = 2	Either $2 \leqslant n_c < n$ or $n_c \leqslant 25$ is violated.
IERR = 3	$n < 4$
IERR = 4	$m < 1$
IERR = 5	Either n_c or n has been modified. This cannot occur when MO $\neq$ 1.
IERR = 6	The argument m has been modified. This cannot occur when MO = 3.
IERR = 7	Points (x_i, y_i) and (x_j, y_j) are equal or are too close where IWK(1) = i and IWK(2) = j.
IERR = 8	The points (x_i, y_i, z_i) (i = 1,...,n) are collinear or almost collinear.

When an error is detected, the routine immediately terminates.

Remarks

(1) Accuracy may be lost due to roundoff error.

(2) The procedure is invariant under a rotation of the x-y coordinate system.

(3) The results are exact when F(x, y) represents a plane.

(4) Derivative estimation at a data point depends on points closest to the data point. Thus the procedure is dependent on the scaling of the abscissae x_i and ordinates y_i of the data points.

Programming. BVIP employs the subroutines IDTANG, IDCLDP, IDLCTN, IDPDRV, IDPTIP and the function IDXCHG. The routines save and exchange information in labeled common blocks. The block names are IDLC and IDPI. The routines were written by Hiroshi Akima (Institute for Telecommunication Sciences, Boulder, Colorado). IDPTIP was modified by Albrecht Preusser (Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin). The error handling was modified by A. H. Morris.

References

(1) Akima, Hiroshi, "A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points," ***ACM Trans. Math Software 4*** (1978), pp. 148-159.

(2) Preusser, A., "Remark on Algorithm 526," ***ACM Trans. Math Software 11*** (1985), pp. 186-187.

CALL BVIP2(MO,n_c,n,X,Y,Z,ℓ,m,XI,YI,ZI,IWK,WK,IERR)

X is an array containing $x_1,...,x_n$, Y an array containing $y_1,...,y_n$, and Z an array containing $z_1,...,z_n$. The input argument n_c is the number of neighboring points to be used for derivative estimation. It is assumed that $2 \leqslant n_c < n$ and $n_c \leqslant 25$. Currently no theory is available for indicating how n_c should be set. The only comment that can be made is that setting n_c to 3, 4, or 5 normally produces satisfactory results.

It is assumed that F(x, y) is to be evaluated at $(\bar{x}_i, \bar{y}_j)$ for $i = 1,...,\ell$ and $j = 1,...,m$. XI is an array containing $\bar{x}_1,...,\bar{x}_\ell$, YI an array containing $\bar{y}_1,...,\bar{y}_m$, and ZI a 2-dimensional array of dimension $\ell \times m$. When BVIP2 is called, if no input errors are detected then $F(\bar{x}_i, \bar{y}_j)$ is computed and stored in ZI(i, j) for $i = 1,...,\ell$ and $j = 1,...,m$.

One may wish to recall BVIP2 a number of times for different grids $(\bar{x}_i, \bar{y}_j)$. If results are needed then a portion of the information that is generated on the first call to BVIP2 can frequently be reused. The reuse of information is controlled by the input argument MO. MO must have the value 1 on the first call to BVIP2. For subsequent calls MO may be assigned the following values:

MO = 1	This setting must be employed when any of the data n_c, n, X, Y is modified. In this case, none of the previously generated information can be reused.
MO = 2	This setting may be used when n_c, n, X, Y are not modified.
MO = 3	This setting is permissible when only Z is modified.

If MO $\neq$ 1 then the contents of IWK and WK must not be altered.

IWK is an array of dimension $kn + \ell m$ or larger when $k = \max\{31, 27 + n_c\}$, and WK is an array of dimension 5n or larger. IWK and WK are storage areas for the routine.

Error Return. IERR is an integer variable. If no errors are detected then IERR is set to 0. Otherwise, IERR is assigned one of the following values:

IERR = 1	MO is not 1, 2, or 3.
IERR = 2	Either $2 \leqslant n_c < n$ or $n_c \leqslant 25$ is violated.
IERR = 3	$n < 4$

IERR = 4	Either $\ell < 1$ or $m < 1$.
IERR = 5	Either n_c or n has been modified. This cannot occur when MO $\neq$ 1.
IERR = 6	Either ℓ or m has been modified. This cannot occur when MO = 3.
IERR = 7	Points (x_i, y_i) and (x_j, y_j) are equal or are too close where IWK(1) = i and IWK(2) = j.
IERR = 8	The points (x_i, y_i, z_i) (i = 1,...,n) are collinear or almost collinear.

When an error is detected, the routine immediately terminates.

Remarks

(1) Accuracy may be lost due to roundoff error.

(2) The procedure is invariant under a rotation of the x-y coordinate system.

(3) The results are exact when F(x, y) represents a plane.

(4) Derivative estimation at a data point depends on points closest to the data point. Thus the procedure is dependent on the scaling of the abscissae x_i and ordinates y_i of the data points.

Programming. BVIP2 employs the subroutines IDTANG, IDCLDP, IDGRID, IDPDRV, IDPTIP and the function IDXCHG. The routines save and exchange information in a labeled common block named IDPI. The routines were written by Hiroshi Akima (Institute for Telecommunication Sciences, Boulder, Colorado). IDPTIP was modified by Albrecht Preusser (Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin). The error handling was modified by A. H. Morris.

References

(1) Akima, Hiroshi, "A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points," *ACM Trans. Math Software 4* (1978), pp. 148-159.

(2) Preusser, A., "Remark on Algorithm 526," *ACM Trans. Math Software 11* (1985), pp. 186-187.

WEIGHTED LEAST SQUARES FITTING WITH POLYNOMIALS OF N VARIABLES

Let $\{(x_1^{(i)},\ldots,x_n^{(i)}) \,:\, i = 1,\ldots,\ell\}$ be a set of ℓ distinct points, $z_1,\ldots,z_\ell$ be the corresponding function values to be approximated, and $w_1,\ldots,w_\ell$ be positive weights. Then for any nonnegative integer IDEG where $\binom{n+\text{IDEG}}{n} \leqslant \ell$,[1] the subroutines MFIT and DMFIT are available for obtaining the coefficients of the unique polynomial $F(x_1,\ldots,x_n)$ of degree IDEG which minimizes $\sum_{i=1}^{\ell} w_i[F(x_1^{(i)},\ldots,x_n^{(i)}) - z_i]^2$. Also, the subroutines MEVAL and DMEVAL are available for computing this polynomial. MFIT and MEVAL yield single precision results, and DMFIT and DMEVAL yield double precision results.

<u>CALL MFIT(n,IDEG,m,ℓ,X,kx,Z,W,R,IERR,IWK,WK,LIWK,LWK,
MIWK,MWK)</u>
<u>CALL DMFIT(n,IDEG,m,ℓ,X,kx,Z,W,R,IERR,IWK,WK,LIWK,LWK,
MIWK,MWK)</u>

It is assumed that $n \geqslant 1$ and $\ell \geqslant 1$. X is an $\ell \times n$ matrix whose ith row contains the point $(x_1^{(i)},\ldots,x_n^{(i)})$; i.e., $X(i,j) = x_j^{(i)}$ for $i = 1,\ldots,\ell$ and $j = 1,\ldots,n$. The argument kx is the number of rows in the dimension statement for X in the calling program. Z is an array containing $z_1,\ldots,z_\ell$ and W an array containing $w_1,\ldots,w_\ell$. X and Z are modified by the routine.

Remark. For IDEG $\geqslant 0$, $\binom{n+\text{IDEG}}{n}$ polynomials $1, x_1,\ldots,x_n, x_1^2, x_1x_2,\ldots$ are needed for a basis of the space of polynomials of degree $\leqslant$ IDEG. The basis polynomials are ordered. For $k \geqslant 1$, the degree $k - 1$ basis polynomials precede the degree k polynomials. The degree k basis polynomials are $x_{i_1}\cdots x_{i_k}$ where $1 \leqslant i_1 \leqslant \cdots \leqslant i_k \leqslant n$. For any two such polynomials $x_{i_1}\cdots x_{i_k}$ and $x_{j_1}\cdots x_{j_k}$, let r be the smallest integer such that $i_r \neq j_r$. Then $x_{i_1}\cdots x_{i_k}$ precedes $x_{j_1}\cdots x_{j_k}$ when $i_r < j_r$.

IDEG and m are variables. If IDEG $\geqslant 0$ then the routine attempts to obtain the polynomial $F(x_1,\ldots,x_n)$ of degree IDEG which is the best least squares fit. Otherwise, if IDEG < 0 then it is assumed that $m \geqslant 1$ and that the first m basis polynomials are to be used to obtain the least squares fit. When the routine terminates, IDEG = the degree of the polynomial $F(x_1,\ldots,x_n)$ obtained and m = the number of basis polynomials that are actually used.

R is an array of dimension ℓ or larger. $R(i) = z_i - F(x_1^{(i)},\ldots,x_n^{(i)})$ for $i = 1,\ldots,\ell$ when the routine terminates.

---

[1] $\binom{k}{0} = 1$ and $\binom{k}{i} = \frac{k(k-1)\cdots(k-i+1)}{i!}$ for $i = 1,2,\ldots$

IWK is an array of dimension LIWK and WK an array of dimension LWK. When the routine terminates, IWK and WK contain the information needed for computing the polynomial $F(x_1,\dots,x_n)$. Sufficient storage for IWK and WK can be assured by setting LIWK and LWK as follows: If IDEG $\geqslant 0$ then let $N = \min\{\ell, \binom{n+\text{IDEG}}{n}\}$ and δ = IDEG. Otherwise, if IDEG < 0 then let N = m and δ be the smallest nonnegative integer such that $\binom{n+\delta}{n} \geqslant m$. Then set

$$\text{LIWK} \geqslant 4N + \delta n$$
$$\text{LWK} \geqslant 2N + n + 1 + \ell N + \frac{1}{2}(N-1)(N-2).$$

N is the maximum number of basis polynomials that will be used, and δ is the degree of the polynomial $F(x_1,\dots,x_n)$ if N basis polynomials are used.

MIWK and MWK are variables. MIWK is set by the routine to the dimension needed for IWK, and MWK is set to the dimension needed for WK. MIWK and MWK depend only on n, ℓ, IDEG, and m.

If MFIT is called then X, Z, W, R, and WK are single precision real arrays. Otherwise, if DMFIT is called then X, Z, W, R, and WK are double precision arrays.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0	The desired polynomial was obtained.
IERR = −1	Not all the basis polynomials could be used. IDEG is the degree of the polynomial obtained. This setting occurs when the problem is not solvable or is too ill-conditioned for the requested degree.
IERR = 1	Only ℓ basis polynomials were used. A polynomial $F(x_1,\dots,x_n)$ was obtained which solves the equations $F(x_1^{(i)},\dots,x_n^{(i)}) = z_i$ for $i = 1,\dots,\ell$.
IERR = 2	(Input error) IDEG < 0 and $m \leqslant 0$.
IERR = 3	(Input error) $n < 1$ or $\ell < 1$.
IERR = 4	(Input error) LIWK or LWK is too small. Set LIWK $\geqslant$ MIWK and LWK $\geqslant$ MWK.

When an input error is detected, the routine immediately terminates.

Remark. When IERR $\leqslant 1$ then MEVAL or DMEVAL may be used to compute the polynomial obtained.

Algorithm. The revised Gram-Schmidt orthogonalization procedure is used.

Programming. MFIT employs the subroutines ALLOT, BASIZ, MTABLE, GNRTP, INCDG, SCALPM, SCALDN, and DMFIT employs the subroutines ALLOT, BASIZ, MTABLE, DGNRTP, DINCDG, DSCALP, DSCALD. MFIT and DMFIT are modifications by A. H. Morris of CONSTR, written by Richard H. Bartels (University of Waterloo) and John J. Jezioranski (Ontario Cancer Institute).

References

(1) Bartels, R. H and Jezioranski, J. J., "Least Squares Fitting using Orthogonal Multinomials," ***ACM Trans. Math Software 11*** (1985), pp. 201-217.

(2) ————————, "Algorithm 634, CONSTR and EVAL: Routines for Fitting Multinomials in a Least Squares Sense," ***ACM Trans. Math Software 11*** (1985), pp. 218-228.

CALL MEVAL(n,KDEG,$\bar{m}$,$\bar{\ell}$,XI,kxi,ZI,IND,IWK,WK,LIWK,LWK,T)

CALL DMEVAL(n,KDEG,$\bar{m}$,$\bar{\ell}$,XI,kxi,ZI,IND,IWK,WK,LIWK,LWK,T)

MEVAL (DMEVAL) computes the polynomial obtained by MFIT (DMFIT), or a portion thereof. Let IDEG and m be the output values given by MFIT (DMFIT).

The argument $\bar{m}$ is a variable. If KDEG < 0 then it is assumed that $1 \leqslant \bar{m} \leqslant m$ and that the polynomial using the first $\bar{m}$ basis polynomials is to be computed. In this case, the polynomial computed is the best least squares fit for the basis polynomials involved.

If KDEG $\geqslant 0$ then it is assumed that KDEG $\leqslant$ IDEG. In this case, when the routine terminates, $\bar{m}$ = the number of basis polynomials used. If $\bar{m} \leqslant m$ (which will be the case when KDEG $<$ IDEG), then the polynomial computed is the polynomial of degree KDEG which is the best least squares fit.

Usage. If IERR = ± 1 when MFIT (DMFIT) terminates, then the setting KDEG = IDEG normally causes an error to occur since $\bar{m} > m$. Hence, if it is desired that the full polynomial obtained by MFIT (DMFIT) be computed, no matter whether the value for IERR is 0 or ± 1, then KDEG should be assigned a negative value and $\bar{m} = m$.

It is assumed that the polynomial is to be computed at the points $(\bar{x}_1^{(i)},\dots,\bar{x}_n^{(i)})$ for $i = 1,\dots,\bar{\ell}$. XI is an $\bar{\ell} \times n$ matrix whose ith row contains the point $(\bar{x}_1^{(i)},\dots,\bar{x}_n^{(i)})$. The argument kxi is the number of rows in the dimension statement for XI in the calling program. ZI is an array of dimension $\bar{\ell}$ or larger. When the routine terminates, ZI(i) contains the value of the polynomial at the point $(\bar{x}_1^{(i)},\dots,\bar{x}_n^{(i)})$ for $i = 1,\dots,\bar{\ell}$.

IWK and WK are the arrays obtained from MFIT or DMFIT. LIWK is the dimension of IWK and LWK the dimension of WK. T is an array of dimension n or larger that is a work space for the routine.

If MEVAL is called then XI, ZI, WK, and T are single precision real arrays. Otherwise, if DMEVAL is called then XI, ZI, WK, and T are double precision arrays.

IND is a variable that reports the status of the results. When the routine terminates, IND has one of the following values:

IND = 0	The desired computation was performed.
IND = −1	(Input error) $\bar{m} < 1$ or $\bar{m} > m$.
IND = −2	(Input error) $n < 1$ or $\bar{\ell} < 1$.

Programming. MEVAL calls the subroutine MEVAL1 and DMEVAL calls the subroutine DMEVAL1. MEVAL and DMEVAL are modifications by A. H. Morris of EVAL, written by Richard H. Bartels (University of Waterloo) and John J. Jezioranski (Ontario Cancer Institute).

EVALUATION OF INTEGRALS OVER FINITE INTERVALS

QAGS, QSUBA, and DQAGS are available for computing integrals over finite intervals. The subroutine QAGS and function QSUBA yield single precision results, and the subroutine DQAGS yields double precision results. These procedures are adaptive. In such procedures, the selection of the points at which the integrand is evaluated depends on the nature of the integrand.

CALL QAGS(F,a,b,AERR,RERR,z,ERROR,NUM,IERR,ℓ,m,n,IWK,WK)

F(x) is a user defined function whose arguments and values are assumed to be single precision real numbers. The purpose of QAGS is to compute the integral $\int_a^b F(x)dx$. F need not be defined at a and b, and it is not required that $a \leqslant b$. F must be declared in the calling program to be of type EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used, and z is a variable. When QAGS is called, z is assigned the value obtained for $\int_a^b F(x)dx$. The routine attempts to obtain a value z which satisfies $|\int F(x)dx - z| \leqslant \max\{AERR, RERR \cdot |\int F(x)dx|\}$. It is assumed that $AERR \geqslant 0$ and $RERR \geqslant 0$. If one wants accuracy to k significant digits then set $RERR = 10^{-k}$.

ERROR and NUM are variables. When QAGS terminates, ERROR is the estimated absolute error of the result and NUM is the number of points at which F was evaluated.

IWK is an array of dimension ℓ and WK is an array of dimension m. IWK and WK are work spaces for the routine. The input argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geqslant 1$ and $m \geqslant 4\ell$. The argument n is a variable. When QAGS terminates, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The routine is satisfied that the integral has been computed to the desired accuracy.
IERR = 1	The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
IERR = 2	The integral has been computed, but because of roundoff error QAGS is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
IERR = 3	Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.

IERR = 4	The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.
IERR = 5	The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.
IERR = 6	(Input Error) Either $\ell < 1$, $m < 4\ell$, or AERR = 0 and RERR $< 50\tau$ where τ is the smallest number for which $\tau \geqslant 5 \cdot 10^{-15}$ and $1 + \tau > 1$ ($\tau = 2^{-47}$ for the CDC 6700). In this case, the variables z, ERROR, NUM, and n are set to 0.

Note. F may have singularities at a and b. However, it is recommended that no singularities appear in the interior of the interval of integration.

Algorithm. The 21 point Kronrod rule and ϵ-algorithm of P. Wynn are used.

Programming. QAGS employs the subroutines QAGSE, QK21F, QPSRT, and QELG. These routines were developed by Robert Piessens and Elise de Doncker-Kapenga (Katholieke Universiteit Leuven, Heverlee, Belgium). The function SPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK: A Subroutine Package for Automatic Integration,*** Springer-Verlag, 1983.

QSUBA(F,a,b,RERR,MCOUNT,ERROR,IND)

F(x) is a user defined function whose arguments and values are assumed to be single precision real numbers. The purpose of QSUBA is to compute the integral $\int_a^b F(x)dx$. F need not be defined at the points a and b. However, it is assumed that $a \leqslant b$. F must be declared in the calling program to be of type EXTERNAL.

RERR is the relative error tolerance to be satisfied. It is assumed that RERR > 0. If one wants accuracy to k significant digits then set RERR = 10^{-k}.

The input argument MCOUNT is the maximum number of points at which F may be evaluated. It is recommended that MCOUNT $\geqslant 1000$.

ERROR is a variable that is set by QSUBA. If the value of QSUBA is not 0 then ERROR is a rough estimate of the relative error of the computed result. Otherwise, if the value of QSUBA is 0 then ERROR is a rough estimate of the absolute error.

IND is a variable that reports the status of the results. When QSUBA terminates, IND will have one of the following values:

IND = 0	The function QSUBA is satisfied that the integral has been computed to the desired accuracy.
IND = 1	The integral has been computed, but QSUBA is not certain of the accuracy of the result.
IND = 2	F(x) was evaluated at MCOUNT points. More evaluations are needed to complete the computation of the integral.
IND = 3	The function QSUBA cannot compute the integral to the desired accuracy.

If IND = 0 or 1 then the function QSUBA is assigned the value obtained for the integral. If IND = 2 then QSUBA has for its value the most recent acceptable partial estimate made of the integral. Otherwise, if IND = 3, then QSUBA has for its value the best estimate of the value of the integral that it can make.

Note. QSUBA assumes that F and its derivatives have no singularities in the closed interval [a,b]. If this is not the case then QAGS should be used. QSUBA is recommended for computing integrals such as $\int_0^{10} \sin \frac{\pi}{2} x^2\, dx$ whose integrands are finitely oscillatory.

Algorithm. Gaussian quadrature is employed.

Programming. QSUBA calls the subroutine QUAD. QSUBA and QUAD were written by T. N. L. Patterson (Queen's University, Belfast, Northern Ireland), and QSUBA was modified by A. H. Morris.

Reference. Patterson, T. N. L., "Algorithm for Automatic Numerical Integration Over a Finite Interval," ***Comm. ACM 16*** (November 1973), pp. 694-699.

<u>CALL DQAGS(F,a,b,AERR,RERR,z,ERROR,NUM,IERR,ℓ,m,n,IWK,WK)</u>

F(x) is a user defined function whose arguments and values are assumed to be double precision real numbers. The purpose of DQAGS is to compute the integral $\int_a^b F(x)dx$. The arguments a and b are double precision real numbers. F need not be defined at a and b, and it is not required that $a \leqslant b$. F must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL.

AERR and RERR are double precision real numbers and z is a double precision variable. AERR and RERR are the absolute and relative error tolerances to be used. When DQAGS is called, z is assigned the value obtained for $\int_a^b F(x)dx$. The routine attempts to obtain a value z which satisfies $|\int F(x)dx - z| \leqslant \max\{AERR, RERR \cdot |\int F(x)dx|\}$. It is assumed that $AERR \geqslant 0$ and $RERR \geqslant 0$.

ERROR is a double precision variable and NUM an integer variable. When DQAGS terminates, ERROR is the estimated absolute error of the result and NUM is the number of points at which F was evaluated.

IWK is an integer array of dimension ℓ and WK a double precision array of dimension m. IWK and WK are work spaces for the routine. The argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geqslant 1$ and $m \geqslant 4\ell$. The argument n is a variable. When DQAGS terminates, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The routine is satisfied that the integral has been computed to the desired accuracy.
IERR = 1	The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
IERR = 2	The integral has been computed, but because of roundoff error DQAGS is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
IERR = 3	Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.
IERR = 4	The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.
IERR = 5	The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.
IERR = 6	(Input Error) Either $\ell < 1$, $m < 4\ell$, or AERR = 0 and RERR $< 50\tau$ where τ is the smallest number for which $\tau \geqslant 5 \cdot 10^{-29}$ and $1 + \tau > 1$ ($\tau = 2^{-95}$ for the CDC 6700). In this case, the variables z, ERROR, NUM, and n are set to 0.

Remarks. F may have singularities at a and b. However, it is recommended that no singularities appear in the interior of the interval of integration. DQAGS is a double precision version of the routine QAGS.

Algorithm. The 21 point Kronrod rule and ϵ-algorithm of P. Wynn are used.

Programming. DQAGS employs the subroutines DQAGSE, DQK21, DQPSRT, and DQELG. These subroutines are double precision versions of the subroutines QAGSE, QK21F, QPSRT, and QELG, developed by Robert Piessens and Elise de Doncker-Kapenga (Katholieke Universiteit Leuven, Heverlee, Belgium). The function DPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK: A Subroutine Package for Automatic Integration,*** Springer-Verlag, 1983.

EVALUATION OF INTEGRALS OVER INFINITE INTERVALS

The subroutines QAGI and DQAGI are available for computing integrals over infinite intervals. QAGI yields single precision results and DQAGI yields double precision results. QAGI and DQAGI are adaptive routines.

<u>CALL QAGI (F,a,MO,AERR,RERR,z,ERROR,NUM,IERR,ℓ,m,n,IWK,WK)</u>

F(x) is a user defined function whose arguments and values are assumed to be real numbers. The argument a is a real number, z is a variable, and MO may be 1, −1, or 2. When QAGI is called, z is assigned the value $\int_a^\infty F(x)\,dx$ if MO = 1 and the value $\int_{-\infty}^a F(x)\,dx$ if MO = −1. Otherwise, if MO = 2 then z is assigned the value $\int_{-\infty}^\infty F(x)\,dx$. If MO = ±1 then F need not be defined at a. Otherwise, if MO = 2 then a is not used. F must be declared in the calling program to be of type EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used. The routine attempts to obtain a value z which satisfies $|\int F(x)\,dx - z| \leqslant \max\{\text{AERR}, \text{RERR}\cdot|\int F(x)\,dx|\}$. It is assumed that AERR $\geqslant 0$ and RERR $\geqslant 0$. If one wants accuracy to k significant digits then set RERR = 10^{-k}.

ERROR and NUM are variables. When QAGI terminates, ERROR is the estimated absolute error of the result and NUM is the number of points at which F was evaluated.

IWK is an array of dimension ℓ and WK is an array of dimension m. IWK and WK are work spaces for the routine. The input argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that ℓ ⩾ 1 and m ⩾ 4ℓ. The argument n is a variable. When QAGI terminates, n = the number of subintervals that appeared in the partition. Normally n < 100.

IERR is a variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The routine is satisfied that the integral has been computed to the desired accuracy.
IERR = 1	The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
IERR = 2	The integral has been computed, but because of roundoff error QAGI is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
IERR = 3	Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.
IERR = 4	The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.

IERR = 5	The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.
IERR = 6	(Input Error) Either $\ell < 1$. $m < 4\ell$, or AERR = 0 and RERR $< 50\tau$ where τ is the smallest number for which $\tau \geqslant 5 \cdot 10^{-15}$ and $1 + \tau > 1$ ($\tau = 2^{-47}$ for the CDC 6700). In this case, the variables z, ERROR, NUM, and n are set to 0.

Note. F may have a singularity at a when MO = ±1. However, it is recommended that no singularities appear in the interior of the interval of integration.

Algorithm. The integrals are transformed as follows:

$$\int_a^\infty F(x)\,dx = \int_0^1 F(a - 1 + 1/t)\frac{dt}{t^2}$$

$$\int_{-\infty}^a F(x)\,dx = \int_0^1 F(a + 1 - 1/t)\frac{dt}{t^2}$$

$$\int_{-\infty}^\infty F(x)\,dx = \int_0^1 [F(-1 + 1/t) + F(1 - 1/t)]\,\frac{dt}{t^2}$$

The transformed integrals are computed by the 15 point Kronrod rule and the ϵ-algorithm of P. Wynn.

Programming. QAGI employs the subroutines QAGIE, QK15I, QPSRT, and QELG. These routines were developed by Robert Piessens and Elise de Doncker-Kapenga (Katholieke Universiteit Leuven, Heverlee, Belgium). The function SPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK: A Subroutine Package for Automatic Integration,*** Springer-Verlag, 1983.

CALL DQAGI (F, a, MO, AERR, RERR, z, ERROR, NUM, IERR, ℓ, m, n, IWK, WK)

F(x) is a user defined function whose arguments and values are assumed to be double precision real numbers. The argument a is a double precision real number, z is a double precision variable, and MO may be 1, −1, or 2. When DQAGI is called, z is assigned the value $\int_a^\infty F(x)\,dx$ if MO = 1 and the value $\int_{-\infty}^a F(x)\,dx$ if MO = −1. Otherwise, if MO = 2 then z is assigned the value $\int_{-\infty}^\infty F(x)\,dx$. If MO = ±1 then F need not be defined at a. F must be declared in the calling program to be of types DOUBLE PRECISION and EXTERNAL.

AERR and RERR are the absolute and relative error tolerances to be used. The routine attempts to obtain a value z which satisfies $|\int F(x)\,dx - z| \leqslant \max\{\text{AERR}, \text{RERR}\cdot|\int F(x)\,dx|\}$. It is assumed that AERR and RERR are nonnegative double precision real numbers.

ERROR is a double precision variable and NUM an integer variable. When DQAGI terminates, ERROR is the estimated absolute error of the result and NUM is the number of points at which F was evaluated.

IWK is an integer array of dimension ℓ and WK a double precision array of dimension m. IWK and WK are work spaces for the routine. The argument ℓ is the maximum number of subintervals in which the interval of integration may be partitioned. It is assumed that $\ell \geqslant 1$ and $m \geqslant 4\ell$. The argument n is a variable. When DQAGI terminates, n = the number of subintervals that appeared in the partition. Normally $n < 100$.

IERR is a variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The routine is satisfied that the integral has been computed to the desired accuracy.
IERR = 1	The interval of integration was partitioned into ℓ subintervals. More subintervals are needed to compute the integral to the desired accuracy.
IERR = 2	The integral has been computed, but because of roundoff error DQAGI is not certain of the accuracy of the result. The error may be greater than that reported by ERROR.
IERR = 3	Extremely bad integrand behavior occurs in the interval of integration. The routine is not certain of the accuracy obtained.
IERR = 4	The algorithm does not converge. It is assumed that the requested accuracy cannot be achieved and that the result is the best which can be obtained.
IERR = 5	The integral may be divergent or it may converge extremely slowly. In this case, the value for z may be meaningless.
IERR = 6	(Input Error) Either $\ell < 1$, $m < 4\ell$, or AERR = 0 and RERR $< 50\tau$ where τ is the smallest number for which $\tau \geqslant 5\cdot 10^{-29}$ and $1 + \tau > 1$ ($\tau = 2^{-95}$ for the CDC 6700). In this case, the variables z, ERROR, NUM, and n are set to 0.

Remarks. F may have a singularity at a when MO = ±1. However, it is recommended that no singularities appear in the interior of the interval of integration. DQAGI is a double precision version of the routine QAGI.

Programming. DQAGI employs the subroutines DQAGIE, DQK15I, DQPSRT, and DQELG. These subroutines are double precision versions of the subroutines QAGIE, QK15I, QPSRT, and QELG, developed by Robert Piessens and Elise de Doncker-Kapenga (Katholieke Universiteit Leuven, Heverlee, Belgium). The function DPMPAR is also used.

Reference. Piessens, R., de Doncker-Kapenga, E., Uberhuber, C. W., and Kahaner, D. K., ***QUADPACK: A Subroutine Package for Automatic Integration,*** Springer-Verlag, 1983.

EVALUATION OF DOUBLE INTEGRALS OVER TRIANGLES

Let $f(x,y)$ be a real-valued function defined on a triangle T. Then the subroutine CUBTRI is available for computing the integral $\iint_T f(x,y)\,dx\,dy$. CUBTRI is an adaptive routine.

CALL CUBTRI $(F, T, \epsilon, MAX, A, ERR, n, W, \ell, IDATA, RDATA, IERR)$

T is a 2-dimensional real array of dimension 2×3 where $T(1,j)$ and $T(2,j)$ are the x and y coordinates of the jth vertex of the given triangle $(j = 1,2,3)$.

IDATA and RDATA are arrays provided by the user containing any integer or real data needed for computing the integrand $f(x,y)$. The arrays may be of any size. F is a user defined real-valued function having the arguments x, y, IDATA, RDATA. It is assumed that $F(x,y,IDATA,RDATA) = f(x,y)$ for any point (x,y) in the triangle of integration T. F must be declared in the calling program to be of type EXTERNAL.

The input argument ϵ is the error tolerance to be satisfied, and A is a variable. When CUBTRI is called, A is assigned the value obtained for $\iint_T f(x,y)\,dx\,dy$. The routine attempts to obtain a value A which satisfies $|\iint f(x,y)\,dx\,dy - A| < \max\{\epsilon, \epsilon|A|\}$. Thus if $|A| \leqslant 1$ then ϵ is an absolute tolerance, whereas if $|A| > 1$ then ϵ is a relative tolerance. If one wants k digit accuracy then set $\epsilon = 10^{-k}$. ERR is a variable. When CUBTRI terminates, ERR is the estimated error $|\iint f(x,y)\,dx\,dy - A|$ of the result.

The input argument MAX is the maximum number of points (x,y) at which F may be evaluated, and n is a variable. On an initial call to CUBTRI, the user must set $n=0$. When the routine terminates, n will have for its value the number of points at which F was evaluated. (For subsequent calls concerning the same integral, see below.)

W is an array of dimension ℓ for internal use by the routine. The input argument ℓ specifies the maximum number of subtriangles in which the triangle of integration T may be partitioned. The subtriangles are stored in W, each subtriangle requiring 6 storage locations. Thus $\ell/6$ is an estimate of the maximum number of subtriangles that might have to be stored ($\ell \leqslant \max\{1, 3m+1\}$ where $m = (MAX/19 - 1)/4$).

IERR is an integer variable that reports the status of the results. When the routine terminates, IERR will have one of the following values:

IERR = 0	The integral was computed to the desired accuracy.
IERR = 1	MAX is too small. F must be evaluated at more points.
IERR = 2	The storage space W is full. Its dimension ℓ must be increased.

IERR = 3	Further subdivision of the subtriangles impossible. This normally occurs when f(x,y) has a singularity in the region. The situation can frequently be eliminated by placing the singularity at a vertex of the triangle of integration T.
IERR = 4	No further improvement in accuracy is possible because of roundoff error in the computation of F or the irregular behavior of F.
IERR = 5	No further improvement in accuracy is possible because subdivision does not change the estimated integral value A. Machine accuracy has probably been reached.

After an initial call to CUBTRI, the routine may be recalled to continue the computation of $\int\int_T f(x,y)\,dx\,dy$. When the routine is recalled, the value of n obtained on the previous call to CUBTRI is used for the next call. This value for n tells the routine where computation should be resumed (using the information previously stored in W). At least one of the values ϵ, MAX, or ℓ must be modified before CUBTRI is recalled. F, T, n, W, IDATA, and RDATA may not be changed when the routine is recalled.

Remark. F may have a singularity at one of the vertices of T (such as in the case when we are computing $\int_0^1\int_0^x (x^2 + 3y^2)^{-1/2}\,dy\,dx$). However, it is recommended that no singularities appear in the interior of the triangle of integration.

Algorithm. The 7-point degree 5 rule of Radon and a new 19-point degree 8 rule are used.

Programming. CUBTRI calls the function RNDERR and subroutine CUBRUL. Information is saved in labeled common blocks. The block names are CUBSTA and CUBATB. The routines were written by D. P. Laurie (National Research Institute for the Mathematical Sciences, Pretoria, South Africa).

Reference. Laurie, D. P., "Algorithm 584, CUBTRI: Automatic Cubature over a Triangle," *ACM Trans. Math Software 8* (1982), pp. 210-218.

SOLUTION OF FREDHOLM INTEGRAL EQUATIONS OF THE SECOND KIND

If k(s,t) and f(s) are continuous real-valued functions for $a \leqslant s,t \leqslant b$, then the equation to be solved is

$$x(s) - \int_a^b k(s,t)\, x(t)dt = f(s)$$

for $a \leqslant s \leqslant b$. Let K be the operator defined by $(Kx)(s) = \int_a^b k(s,t)\, x(t)dt$ for any real-valued function x continuous on [a,b]. Then (Kx)(s) is continuous for $a \leqslant s \leqslant b$, and k is called the ***kernel*** of K. Also the above integral equation can be written in the form $(I - K)x = f$ where I is the identity operator. This equation has a unique solution if and only if $I - K$ is 1-1, in which case $x = (I - K)^{-1} f$. The subroutine IESLV is available for computing this solution.

Remark. If C[a,b] is the normed space of real-valued functions x continuous on [a,b] and having the norm $\|x\| = \max\{|x(t)| : a \leqslant t \leqslant b\}$, then K is a compact mapping $C[a,b] \to C[a,b]$ having the norm $\|K\| = \max_{a \leqslant s \leqslant b} \int_a^b |k(s,t)|\, dt$.

CALL IESLV (k, f, a, b, EPS, IFLAG, S, X, ℓ, N, M, NF, MF, NORMK, WK, IERR)

It is assumed that $a < b$, and that k(s,t) and f(s) are user defined real-valued functions for $a \leqslant s,t \leqslant b$. It is recommended that k and f be several times continuously differentiable. The functions k and f must be declared in the calling program to be of type EXTERNAL.

EPS is a variable and IFLAG an input argument whose values are 0 and 1. On input EPS is the error tolerance that the solution must satisfy. If IFLAG = 0 then EPS is an absolute tolerance. Otherwise, if IFLAG = 1 then EPS is a relative tolerance. If IESLV successfully solves the equation, then on output EPS will be the estimated error of the result.

Before the remaining arguments s, x, ℓ,... can be described, it is necessary to give a brief outline of the algorithm used. When IESLV is called, the integral equation is approximated by

(*) $$x_n(s) - \sum_{j=1}^{n} w_{jn} k(s,t_{jn})\, x_n(t_{jn}) = f(s)$$

for $a \leqslant s \leqslant b$. Here w_{jn} and t_{jn} are the weights and nodes of Gauss-Legendre quadrature. This equation is treated as an interpolation for x(s) in terms of the values $x_n(t_{jn})$. These values are obtained by solving the equations

(**) $$x_n(t_{in}) - \sum_{j=1}^{n} w_{jn}\, k(t_{in}, t_{jn})\, x_n(t_{jn}) = f(t_{in})$$

for $i = 1,\ldots,n$. This system of equations can be solved directly or iteratively. The following algorithm is used:

(1) Set $n = 2$ and go to (2).

(2) The n equations are solved directly. Then set $m = 2n$ and solve the m equations (**) iteratively. If the rate of convergence is sufficiently rapid or n cannot be increased, then go to (3). Otherwise, set $n = m$, and go to (2).

(3) Here n remains fixed. Repeatedly double the value of m and solve the m equations (**) iteratively until convergence occurs, m cannot be increased, or the iterations diverge.

When the algorithm terminates, values $x_m(t_{im})$ will have been computed for the nodes t_{im} $(i = 1,\dots,m)$. Then from (*), $x(s) \approx x_m(s)$ can be interpolated for $a \leqslant s \leqslant b$.

N and M are input arguments, and WK is an array that is a work space for the routine. N and M are upper limits for n and m in the algorithm, and WK is of dimension $5N^2 + 9(N + M)$ or larger. It is assumed that $M \geqslant N \geqslant 2$. Since n and m are always powers of 2, N and M need only be set to powers of 2. However, this is not required.

S and X are arrays, and ℓ is a variable. On input it is assumed that $\ell \geqslant 0$. If $\ell > 0$ then S is assumed to contain ℓ points $s_1,\dots,s_\ell$ at which the solution x(s) is to be evaluated. Also X is assumed to be an array of dimension ℓ or larger. When IESLV terminates, X contains the values obtained for $x(s_1),\dots,x(s_\ell)$. (This is true irregardless of whether or not the desired accuracy has been achieved.) Otherwise, if $\ell = 0$ then S and X are assumed to be arrays of dimension M or larger. When IESLV terminates ℓ = the final value obtained for m, S contains the Gaussian nodes $t_{i\ell}$ $(i = 1,\dots,\ell)$, and X contains the values obtained for $x(t_{i\ell})$.

NF and MF are variables. When the routine terminates, NF is the final value for n and MF the final value for m.

NORMK is a real variable. If $\ell > 0$ on input, then when IESLV terminates, NORMK is an approximation for $\|K\|$. Otherwise, if $\ell = 0$ then NORMK = 0.

IERR is a variable that reports the status of the results. When the routine terminates, IERR has one of the following values:

IERR = 0	The solution was obtained to the desired accuracy. EPS is the estimated error of the result.
IERR = 1	The solution was not obtained to the desired accuracy. EPS is the estimated error of the result.
IERR = 2	The solution was not obtained to the desired accuracy. It is not clear what accuracy (if any) has been achieved. EPS has been set to 0.
IERR = 3	The input value for EPS was too small. This may be due to ill-conditioning of the integral equation. The value of EPS was reset to a more realistic tolerance, which the solution satisfied.
IERR = 4	The solution x(s) was obtained at the Gaussian nodes to the desired precision. However, the interpolation process may not preserve this accuracy

for the evaluation of x(s) for other points s. EPS is the estimated error of the solution at the Gaussian nodes.

IERR = 5 The solution x(s) was not obtained to the desired accuracy at the Gaussian nodes. EPS is the estimated error at these nodes. The interpolation process may not preserve this accuracy for the evaluation of x(s) for other points s.

IERR = 6 The input value for EPS was too small. This may be due to ill-conditioning of the integral equation. The value of EPS was reset to a more realistic tolerance, which the solution x(s) satisfied at the Gaussian nodes. The interpolation process may not preserve this accuracy for the evaluation of x(s) for other points s.

Difficulties can arise, causing IERR $\geqslant 1$, when the integral equation is ill-conditioned or the kernel k(s,t) is not appropriate for standard Gaussian quadrature. Ill-conditioning can occur when the operator $I - K$ is near singular or the norm $\|K\|$ is exceedingly large. Inappropriate kernels k(s,t) include those which are highly oscillatory or not continuously differentiable for s and t in the open interval (a,b).

Programming. IESLV employs the subroutines IEGS, NSTERP, WANDT, LEAVE, ITERT, LNSYS and functions RMIN, RNRM, CONEW. The routines save and exchange information in labeled common blocks. The block names are XXINFO and XXLIN. The routines were written by Kendall E. Atkinson (University of Iowa), and modified by A. H. Morris. The function SPMPAR is also used.

Reference. Atkinson, K. E., "An Automatic Program for Linear Fredholm Integral Equations of the Second Kind," ***ACM Trans. Math Software 2*** (1976), pp. 154-171.

THE INITIAL VALUE SOLVERS – INTRODUCTORY COMMENTS

Let $y'(t) = f(t,y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y),\ldots,f_n(t,y))$ and $y(t) = (y_1(t),\ldots,y_n(t))$. Assume that y(a) is known. Then for $b \neq a$ the subroutines ODE, RKF45, GERK, SFODE, and SFODE1 are available for computing y(b). These routines are adaptive variable step differential equations solvers. The remaining subroutines (RK and RK8) are fixed order, one step procedures. Given a value y(t) and a step size h, the one step procedures compute a value for $y(t + h)$. The problem of selecting an appropriate step size is left to the user. Given y(a) and b, the one step routines must be repeatedly called to step along the interval from a to b. The situation, however, is considerably different with the adaptive routines. ODE, SFODE, and SFODE1 are variable order, variable step procedures, and RKF45 and GERK are fixed order, variable step procedures. Given y(a), b, and the error tolerances that are to be maintained, these solvers continually adjust their orders and step sizes as they (automatically) step along the interval from a to b.

The adaptive routines differ in their capabilities. While in principle any of the routines can be used for solving a set of differential equations, ODE, RKF45, and GERK are recommended for nonstiff equations, and SFODE and SFODE1 are recommended for stiff equations (see the note at the end of this section). If one does not know whether the equations are stiff, then ODE should be tried. ODE maintains greater accuracy than the other routines, and it will notify the user if the equations appear to be stiff. ODE, RKF45, and GERK should be able to handle mildly stiff problems satisfactorily, but they are decidedly not appropriate for extremely stiff problems. SFODE and SFODE1 are the only routines in the mathematics library that are capable of solving extremely stiff equations.

If the equations to be solved are nonstiff, then the choice between ODE and RKF45 depends on the amount of accuracy needed and the cost of the derivative evaluations. If the accuracy requirements are high then ODE is recommended. However, if the accuracy requirements are low and the derivative evaluations are inexpensive, then RKF45 may be the most efficient routine for the problem. RKF45 frequently requires more derivative evaluations than ODE, but its overhead is considerably less than that for ODE.

When the user specifies the error tolerances to be satisfied, normally he is only interested in the global error (the accuracy of y(b)). However, the adaptive routines employ the tolerances for controlling local error (the error generated at each internal step in the interval). No attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. GERK is the only routine that estimates the global error. This routine employs the same Runge-Kutta-Fehlberg formulae used by RKF45. GERK is 2-3 times slower than RKF45, but it is more accurate.

Output Considerations. Generally, when the user has a system of equations $y'(t) = f(t,y(t))$ to be solved (where $y(a_0)$ is known), he wants its solution at a sequence of points $a_1,...,a_N$. If an adaptive routine is being used, then the routine will be repeatedly called to step along the axis from each point a_i to the next. If ODE, SFODE, or SFODE1 is being employed, then the number and closeness of the output points $a_1,...,a_N$ should be of no concern. These routines partially ignore a_{i+1} in the selection of the step size when going from a_i to a_{i+1}. Instead, they step along the axis using the largest steps that are appropriate (efficiency and accuracy are the prime concerns). Normally a_{i+1} is passed in the process. If a_{i+1} is passed then a quick interpolation yields the desired result at a_{i+1}. Thus the process of solving the equations for a_{i+1} when $y(a_i)$ is known may require that no steps be taken (a_{i+1} may have been bypassed on a previous call to ODE, SFODE, or SFODE1), or it may require that one or more steps be taken.

The situation is considerably different if RKF45 or GERK is used. These routines select their step size so as not to bypass a_{i+1} when going from a_i to a_{i+1}. Thus the output points $a_1,...,a_N$ may be so close to one another as to force inordinately small step sizes (when such step sizes would otherwise not be needed). If this occurs then the efficiency of RKF45 and GERK may deteriorate dramatically. The routines will notify the user of the situation, and the user will be left with the following options:

(1) Switch to an adaptive routine such as ODE which performs interpolation.

(2) Use a nonadaptive one step routine such as RK or RK8.

(3) Use RKF45 or GERK in a one step mode (this capability is permitted).

If option (3) is taken, then the user may just repeatedly call RKF45 or GERK (in the one step mode) until a_N is reached. This will generate a sequence of output points $a'_1,...,a'_m$ where $a'_m = a_N$.

Note. A system of equations $y'(t) = f(t,y(t))$ is stiff if its solution contains a component that decays far more rapidly than the others. This occurs when the real parts of the eigenvalues of the Jacobian matrix $Jf(t) = (\partial f_i/\partial y_j)$ are widely separated and at least one of the eigenvalues has a large negative real part. A set of equations may be stiff in some intervals and nonstiff in other intervals. The difficulty that the traditional procedures have with stiff equations is that the most rapidly decaying components force the step size to become inordinately small, even after these components no longer contribute to the solution.

ADAPTIVE ADAMS SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t,y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y),...,f_n(t,y))$ and $y(t) = (y_1(t),...,y_n(t))$. Assume that y(a) is known. Then for $b \neq a$ the subroutine ODE is available for computing y(b). ODE is recommended for nonstiff equations. The algorithm used is a variable order, variable step Adams predictor-corrector procedure.

CALL ODE(F,n,Y,T,TOUT,RERR,AERR,IFLAG,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Y,DY)

Y and DY are arrays of dimension n or larger. Y(1),...,Y(n) contain the values $y_1(t),...,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),...,y_n'(t)$ using $y'(t) = f(t,y(t))$ and stores the results in DY(1),...,DY(n). F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension 100 + 21n or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to ODE.

It is assumed that $a \neq b$. The argument Y in the call line of ODE is an array of dimension n or larger, and the arguments T,RERR,AERR,IFLAG are variables. (TOUT need not be a variable.) When ODE is initially called, it is assumed that:

T = a
TOUT = b
Y(1),...,Y(n) contain the values $y_1(a),...,y_n(a)$
RERR = the relative error tolerance to be satisfied
AERR = the absolute error tolerance to be satisfied
IFLAG = ± 1

It is preferable, both for efficiency and accuracy, that ODE be permitted to step along the axis from a to b using the largest steps that are appropriate. This is what is done when IFLAG is set to 1. If IFLAG = 1 then ODE will step along the axis, possibly passing b and going as far as the point $a + 10 * (b - a)$. If b is passed, then the solution for the equations at b is obtained by interpolation. However, IFLAG = 1 cannot be used if the equations are not defined at all points between b and $a + 10 * (b - a)$. In a situation such as this, when integration cannot be permitted to automatically step internally past TOUT, IFLAG must be set to -1. If IFLAG = -1 then it is required that the subroutine F be defined at TOUT. However, F need not be defined at points t past TOUT. If the equations $y'(t) = f(t,y(t))$ are not defined at t = TOUT, then it should suffice to let F set each DY(i) = 0.0 when t = TOUT. A solution (if one exists) will be obtained by extrapolation.

If IFLAG is positive (negative), then after ODE terminates IFLAG will have been reset by ODE to one of the values 2,3,4,5,6,7 (2,−3,−4,−5,−6,7). These values have the following meanings:

IFLAG =	2	The equations have been solved at TOUT. T now has the value TOUT and Y contains the solution at TOUT.
IFLAG =	±3	The error tolerances RERR and AERR are too small. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point. RERR and AERR have been reset to larger acceptable values.
IFLAG =	±4	MAXNUM steps were performed.[1] More steps are needed to reach TOUT. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point.
IFLAG =	±5	MAXNUM steps were performed. More steps are needed to reach TOUT. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point. The equations appear to be stiff.
IFLAG =	±6	ODE did not reach TOUT because AERR = 0. T is set to the point closest to TOUT for which the equations were solved and Y contains the solution at the point.
IFLAG =	7	No computation was performed. An input error was detected. The user must correct the error and call ODE again.

If IFLAG = ±3, ±4, ±5 then to continue the integration just call ODE again. Similarly, if IFLAG = ±6 then reset AERR to be positive and call ODE again. In these cases do not modify T,Y,IFLAG. The output values for these parameters are the appropriate input values for the next call to ODE. However, AERR and RERR may always be modified when continuing an integration.

If the equations appear to be stiff (i.e., if IFLAG = ±5) then ODE may not be suitable for solving the equations. In this case it is recommended that a routine designed specifically for stiff equations be used.

Whenever IFLAG = 2 occurs, then the equations have been solved at TOUT = b. WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration, normally one need only reset TOUT to the new value c and call ODE again. Do not modify T,Y,IFLAG. The output values for these parameters are normally the appropriate input values for the next call to ODE. The one exception is when the equations are not defined at points past c. If this

[1] Each step normally requires two calls to the subroutine F. Currently the internal parameter MAXNUM is set at 500.

occurs, then one should also reset the output value IFLAG = 2 (from the last call to ODE) to the input value IFLAG = −2 for the next call to ODE. If IFLAG is reset to −2, then integration will not proceed internally past the new TOUT when ODE is recalled. In this case, the subroutine F need not be defined for points past TOUT. However, it is required that F be defined at TOUT.

If after going from a to b, ODE is recalled to continue the integration and solve the equations at a new point c, then it is important that IFLAG be set to ±2 for the next call to ODE. Setting IFLAG to ±1 causes the integration procedure to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting not only can take more time, but also can lead to less accurate results. If IFLAG is set to ±2, then the integration procedure restarts itself only if

(1) the direction of integration is being reversed or
(2) IFLAG was negative when ODE was last recalled.

The direction of integration is reversed when b does not lie between a and c.

If one has a choice between setting IFLAG to be positive or negative, then always set IFLAG to be positive. Extrapolation is normally involved when IFLAG is negative. The extrapolation can require more time and be less accurate than the procedures employed when IFLAG is positive.

Input Errors. IFLAG = 7 occurs when one of the following conditions is violated:

(1) $n \geqslant 1$
(2) T $\neq$ TOUT
(3) RERR $\geqslant 0$ and AERR $\geqslant 0$
(4) RERR and AERR are not both 0.
(5) $1 \leqslant |\text{IFLAG}| \leqslant 5$
    IFLAG = ± 6 and AERR > 0
(6) When continuing an integration, the input value for T is the output value of TOUT from the previous call to ODE.

The last condition is automatically satisfied if the user has not inadvertently modified T.

Error Control. Assuming that ODE has obtained the correct value for y(t), let e_i denote the error generated in the computation Y(i) of $y_i(t+h)$ for $i = 1,\dots,n$ when ODE steps from t to $t+h$. If EPS = max {AERR,RERR} then ODE attempts at each internal step to maintain the accuracy

$$\sqrt{\sum_{i=1}^{n}\left[\frac{e_i}{\mathrm{WT}(i)}\right]^2} \leqslant \mathrm{EPS}$$

where WT(i) = |Y(i)| * RERR/EPS + AERR/EPS. If the inequality is satisfied then

$$|e_i| \leqslant \text{EPS} * \text{WT(i)} = |\text{Y(i)}| * \text{RERR} + \text{AERR}$$

for i = 1,...,n. This error criterion includes as special cases relative error (AERR = 0) and absolute error (RERR = 0). However, if AERR = 0 and Y(i) = 0 for some i, then WT(i) = 0 and IFLAG = ± 6 will occur.

When going from T to TOUT, ODE continually adjusts and readjusts its order and step size so as to maintain accuracy at each step. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results by rerunning the problem. If this is done, then in the second run ask for greater accuracy.

Programming. ODE employs the subroutines DE1, STEP1, and INTRP. These routines were written by L. F. Shampine and M. K. Gordon (Sandia Laboratories). The function SPMPAR is also used.

Reference. Shampine, L. F., and Gordon, M. K., ***Computer Solution of Ordinary Differential Equations.*** W. H. Freeman and Company, San Francisco, 1975.

ADAPTIVE RKF SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t,y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y),...,f_n(t,y))$ and $y(t) = (y_1(t),...,y_n(t))$. Assume that y(a) is known. Then for $b \neq a$ the subroutine RKF45 is available for computing y(b). RKF45 was designed for solving nonstiff differential equations when derivative evaluations are inexpensive and the accuracy requirements are low (less than 8 significant digits). The routine employs the fourth-fifth order Runge-Kutta-Fehlberg formulae.

CALL RKF45(F,n,Y,T,TOUT,RERR,AERR,IFLAG,WK,IWK)

The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Y,DY)

Y and DY are arrays of dimension n or larger. Y(1),...,Y(n) contain the values $y_1(t),...,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),...,y_n'(t)$ using $y'(t) = f(t,y(t))$ and stores the results in DY(1),...,DY(n). F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension 3 + 6n or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to RKF45.

Y is an array of dimension n or larger, and the arguments T, RERR, IFLAG are variables. (TOUT and AERR need not be variables.) When RKF45 is initially called, it is assumed that:

T = a
TOUT = b
Y(1),...,Y(n) contain the values $y_1(a),...,y_n(a)$
RERR = the relative error tolerance to be satisfied
AERR = the absolute error tolerance to be satisfied
IFLAG = ± 1

Normally IFLAG = 1. However, if only a single step in the direction of TOUT is to be taken, then set IFLAG = − 1.

On output T is set to the point closest to TOUT for which the equations were solved, and Y contains the solution at T. Also IFLAG reports the status of the results. RKF45 sets IFLAG to one of the following values:

IFLAG = 2	The equations were successfully solved at TOUT. T now has the value TOUT.
IFLAG = −2	A single step in the direction of TOUT was taken.
IFLAG = 3	The error tolerance RERR was too small. RERR has been reset to a larger acceptable value.

IFLAG = 4	3000 derivative evaluations were performed. More derivative evaluations are needed to reach TOUT.
IFLAG = 5	RKF45 did not reach TOUT because AERR = 0. AERR must be made positive.
IFLAG = 6	Too much accuracy has been requested. AERR and/or RERR must be increased in value.
IFLAG = 7	The closeness of the output points is restricting the natural step size choice.
IFLAG = 8	No computation was performed. An input error was detected. The user must correct the error and call RKF45 again.

If IFLAG = 2 then the equations have been solved at TOUT = b. The arrays WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration the user need only reset TOUT to the new point c and call RKF45 again.

If IFLAG = −2 then to continue the integration another single step just call RKF45 again. In the single step mode (IFLAG = −1, −2) the user must keep in mind that each step taken is in the direction of the current TOUT. Upon reaching TOUT (which is indicated by IFLAG being set to 2), the user may then define a new TOUT and set IFLAG to ±2 for further integration.

If IFLAG = 3 or 4 then to continue the integration just call RKF45 again. However, if IFLAG = 5 then the user must first reset AERR to be positive before RKF45 can be recalled. If IFLAG = 6 then it is required that IFLAG be reset to ±2 and that AERR and/or RERR be increased in value. If this is not done then the run will be terminated by a STOP instruction!

If IFLAG = 7 then the user should either switch to another routine, or he should use the one step mode. This situation is discussed in the ***Initial Value Solvers – Introductory Comments*** section. If the user insists on continuing the integration with RKF45 in the standard multistep mode, then it is required that IFLAG be reset to 2 before RKF45 is recalled. If this is not done then the run will be terminated by a STOP instruction.

If after going from a to b, RKF45 is recalled to continue the integration and solve the equations at a new point c, then it is important that IFLAG be set to ±2 instead of ±1. Setting IFLAG = ±1 causes the integration process to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting wastes time and is normally not needed. The one exception is when the direction of integration is to be reversed. Then the integration should be restarted.

Notes.

(1) AERR and RERR can be modified each time that RKF45 is called.

(2) When continuing an integration, one may switch from the standard multistep mode (IFLAG = 2) to the one step mode (IFLAG = −2) whenever it is convenient to do so.

Input Errors. IFLAG = 8 occurs when one of the following conditions is violated:

(1) $n \geqslant 1$

(2) T ≠ TOUT when IFLAG ≠ ± 1

(3) RERR ⩾ 0 and AERR ⩾ 0

(4) IFLAG = ±1, ±2, 3, 4,...,8

Error Control. Pure absolute error control is not permitted. If RERR = 0 then RERR is reset to a value slightly greater than 10^{-12}, IFLAG is set to 3, and the routine terminates.

When going from T to TOUT, RKF45 continually adjusts and readjusts its step size so as to maintain accuracy at each step. Assuming that RKF45 has obtained the correct value for y(t), let e_i denote the error generated in the computation of $y_i(t+h)$ for $i = 1,\ldots,n$ when RKF45 steps from t to t + h. Then at each step the error is controlled so that

$$|e_i| \leqslant \frac{|y_i(t)| + |y_i(t+h)|}{2}\,\text{RERR} + \text{AERR}$$

for $i = 1,\ldots,n$. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one way wish to double-check the results. This can best be done by comparing the results obtained by RKF45 with those obtained by ODE or GERK. If ODE is used then ask for greater accuracy. However, if GERK is used then the current error tolerances can be used. GERK is more accurate than RKF45, and it estimates the global error generated.

Programming. RKF45 employs the subroutines RKFS and FEHL. These routines were written by H. A. Watts and L. F. Shampine (Sandia Laboratories).

References. Shampine, L. F., and Allen, R. C., ***Numerical Computing: An Introduction***, W. B. Sanders, Philadelphia, 1973.

ADAPTIVE RKF SOLUTION OF NONSTIFF DIFFERENTIAL EQUATIONS WITH GLOBAL ERROR ESTIMATION

Let $y'(t) = f(t,y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y),...,f_n(t,y))$ and $y(t) = (y_1(t),...,y_n(t))$. Asume that y(a) is known. Then for $b \neq a$ the subroutine GERK is available for computing y(b). GERK was designed for solving nonstiff differential equations when derivative evaluations are inexpensive and the accuracy requirements are low (less than 8 significant digits). The routine employs the fourth-fifth order Runge-Kutta-Fehlberg formulae. GERK estimates the accuracy of the solution y(b).

<u>CALL GERK(F,n,Y,T,TOUT,RERR,AERR,IFLAG,GERROR,WK,IWK)</u>

The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Y,DY)

Y and DY are arrays of dimension n or larger. Y(1),...,Y(n) contain the values $y_1(t),...,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),...,y_n'(t)$ using $y'(t) = f(t,y(t))$ and stores the results in DY(1),...,DY(n). F must be declared in the calling program to be of type EXTERNAL.

WK is an array of dimension 3 + 8n or larger, and IWK is an array of dimension 5 or larger. WK and IWK contain information needed for subsequent calls to GERK.

Y is an array of dimension n or larger, and the arguments T and IFLAG are variables. (TOUT, RERR, AERR need not be variables.) When GERK is initially called, it is assumed that:

T = a
TOUT = b
Y(1),...,Y(n) contain the values $y_1(a),...,y_n(a)$
RERR = the relative error tolerance to be satisfied
AERR = the absolute error tolerance to be satisfied
IFLAG = ±1

Normally IFLAG = 1. However, if only a single step in the direction of TOUT is to be taken, then set IFLAG = −1.

GERROR is an array of dimension n or larger. On output T is set to the point closest to TOUT for which the equations were solved, Y contains the solution at T, and GERROR(i) is an estimate of the error of Y(i) for i = 1,...,n. Also IFLAG reports the status of the results. GERK sets IFLAG to one of the following values:

IFLAG = 2	The equations were successfully solved at TOUT. T now has the value TOUT.

IFLAG = −2	A single step in the direction of TOUT was taken.
IFLAG = 3	9000 derivative evaluations were performed. More derivative evaluations are needed to reach TOUT.
IFLAG = 4	GERK did not reach TOUT because AERR = 0. AERR must be made positive.
IFLAG = 5	Too much accuracy has been requested. AERR and/or RERR must be increased in value.
IFLAG = 6	The closeness of the output points is restricting the natural step size choice.
IFLAG = 7	No computation was performed. An input error was detected. The user must correct the error and call GERK again.

If IFLAG = 2 then the equations have been solved at TOUT = b. The arrays WK and IWK contain information that can often be reused in continuing along the axis and solving the equations at a new point c. To continue the integration the user need only reset TOUT to the new point c and call GERK again.

If IFLAG = −2 then to continue the integration another single step just call GERK again. In the single step mode (IFLAG = −1,−2) the user must keep in mind that each step taken is in the direction of the current TOUT. Upon reaching TOUT (which is indicated by IFLAG being set to 2), the user may then define a new TOUT and set IFLAG to ±2 for further integration.

If IFLAG = 3 then to continue the integration just call GERK again. However, if IFLAG = 4 then the user must first reset AERR to be positive before GERK can be recalled. If IFLAG = 5 then it is required that IFLAG be reset to ±2 and that AERR and/or RERR be increased in value. If this is not done then the run will be terminated by a STOP instruction!

If IFLAG = 6 then the user should either switch to another routine, or he should use the one step mode. This situation is discussed in the ***Initial Value Solvers–Introductory Comments*** section. If the user insists on continuing the integration with GERK in the standard multistep mode, then it is required that IFLAG be reset to 2 before GERK is recalled. If this is not done then the runs will be terminated by a STOP instruction.

If after going from a to b, GERK is recalled to continue the integration and solve the equations at a new point c, then it is important that IFLAG be set to ±2 instead of ±1. Setting IFLAG = ±1 causes the integration process to be restarted, thereby eliminating the information being saved in WK and IWK. Restarting wastes time and is normally not needed. The one exception is when the direction of integration is to be reversed. Then the integration should be restarted.

Notes

(1) AERR and RERR can be modified each time that GERK is called.

(2) When continuing an integration, one may switch from the standard multistep mode (IFLAG = 2) to the one step mode (IFLAG = −2) whenever it is convenient to do so.

Input errors. IFLAG = 7 occurs when one of the following conditions is violated:

(1) $n \geqslant 1$

(2) T $\neq$ TOUT when IFLAG $\neq \pm 1$

(3) RERR $\geqslant 0$ and AERR $\geqslant 0$

(4) IFLAG = $\pm 1, \pm 2, 3, 4, \ldots, 7$

Accuracy Considerations. Error control in GERK is almost identical to that in RKF45. One minor difference is that GERK never employs relative error tolerances less than $3 \cdot 10^{-11}$, whereas RKF45 never employs relative error tolerances less than 10^{-12}.

The only significant difference between GERK and RKF45 is that GERK generates two solutions for the differential equations, whereas RKF45 generates only one. Let $y(t)$ and $\tilde{y}(t)$ denote the solutions generated by GERK at point t. One of these solutions, say $y(t)$, will frequently be identical to the solution computed by RKF45. When going from t to $t + h$, the step size h is selected so that $y(t + h)$ satisfies the local error criterion. After a suitable h is found then GERK takes two steps, each of length $h/2$, to generate $\tilde{y}(t + h)$ from $\tilde{y}(t)$. When GERK terminates, say at point T, then the $\tilde{y}(T)$ solution is stored in the Y array and GERK estimates the error of $\tilde{y}_i(T)$ to be $(y_i(T) - \tilde{y}_i(T))/31$ for $i = 1, \ldots, n$.

Programming. GERK employs the subroutines GERKS and FEHL. These routines were written by H. A. Watts and L. F. Shampine (Sandia Laboratories). The function SPMPAR is also used.

Reference. Shampine, L. F., and Allen, R. C., ***Numerical Computing: An Introduction,*** W. B. Saunders, Philadelphia, 1973.

ADAPTIVE SOLUTION OF STIFF DIFFERENTIAL EQUATIONS

Let $y'(t) = f(t,y(t))$ denote a system of n ordinary first order differential equations where $f(t,y) = (f_1(t,y),\ldots,f_n(t,y))$ and $y(t) = (y_1(t),\ldots,y_n(t))$. Assume that y(a) is known. Then for $b \neq a$ the following subroutines are available for computing y(b). These routines are designed for stiff differential equations. The algorithm used is a variable order, variable step backward differentiation procedure.

CALL SFODE(F,n,Y,T,TOUT,INFO,RERR,AERR,IERR,WK,ℓ,IWK,m,RD,ID)

RD and ID are arrays defined by the user containing any real and integer data that is needed for computing f. These arrays may contain any information that the user desires. The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Y,DY,RD,ID)

Y and DY are arrays of dimension n. On input Y contains the values $y_1(t),\ldots,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),\ldots,y_n'(t)$ using $y'(t) = f(t,y(t))$ and stores the results in DY. F must be declared in the calling program to be of type EXTERNAL.

INFO is an array of dimension 4, WK an array of dimension ℓ, and IWK an array of dimension m. WK and IWK are work spaces for the routine, and INFO is an array defined by the user containing information on how the equations are to be treated.

INFO(1): Set INFO(1) = 0 on an initial call to the routine. On a continuation call INFO(1) = 1.

INFO(2): Normally INFO(2) = 0. However, INFO(2) = 1 when the intermediate output mode is desired (see below).

INFO(3): When INFO(3) = 0, SFODE proceeds from a to b using the largest steps that are appropriate. If b is passed then y(b) is obtained by interpolation. However, for some problems the routine cannot be permitted to step past a point TSTOP because $y'(t) = f(t,y(t))$ is discontinuous or not defined beyond TSTOP. When this is the case set INFO(3) = 1 and WK(1) = TSTOP.

INFO(4): When proceeding from a to b, the $n \times n$ Jacobian matrix $Jf(t) = (\partial f_i/\partial y_j)$ is computed and stored in WK. Normally it is assumed that

INFO(4) = 0
$\ell \geqslant 250 + 10n + n^2$
$m \geqslant 55 + n$.

However, if Jf(t) is banded for all t, having the lower and upper band widths m_ℓ and m_u where $2m_\ell + m_u < n$, then the following setup can be used:

INFO(4) = 1
IWK(1) = m_ℓ
IWK(2) = m_u

$$\ell \geqslant 250 + 10n + (2m_\ell + m_u + 1)n$$
$$m \geqslant 55 + n$$

T, TOUT, RERR, and AERR are variables, and Y is an array of dimension n or larger. On an initial call to the routine it is assumed that

INFO(1) = 0
T = a
TOUT = b
Y(1),...,Y(n) contain the values $y_1(a),\ldots,y_n(a)$
RERR = the relative error tolerance to be satisfied (RERR $\geqslant$ 0)
AERR = the absolute error tolerance to be satisfied (AERR $\geqslant$ 0).

IERR is a variable. When SFODE terminates T is the final point where the equations were solved, Y contains the solution at T, and IERR reports the status of the results. IERR is assigned one of the following values:

IERR = 1	A step was taken in the intermediate output mode. TOUT was not reached. To continue, call the routine again.
IERR = 2	The solution at TOUT was obtained by stepping exactly to TOUT.
IERR = 3	The solution at TOUT was obtained by stepping past TOUT and then interpolating. On output T = TOUT.
IERR = −1	500 steps have been taken. TOUT has not been reached. To continue, call the routine again.
IERR = −2	The tolerances RERR and AERR were too stringent. RERR and AERR have been modified by the routine. The tolerances may be further modified by the user if he desires. To continue, call the routine again.
IERR = −3	In this case AERR = 0. SFODE stopped when y_i became 0. INFO(1) was set to −i. To continue set AERR to be positive, INFO(1) = 1, and call the routine again.
IERR = −6	Convergence failed on the last attempted step. An inaccurate jacobian matrix may be the problem. To continue, restart the routine by setting INFO(1) = 0 and call the routine again.
IERR = −7	Repeated error test failures occurred on the last attempted step. The problem should be reexamined. A singularity may be present in the solution. To continue, restart by setting INFO(1) = 0 and call the routine again.
IERR $\leqslant$ −33	An input error was detected (see below).

When IERR $\geqslant$ −2, then INFO(1) = 1 on output.

When the equations are solved at TOUT (IERR = 2 or 3), integration can be continued along the axis to solve the equations at a new point c beyond TOUT. To continue, one need only set TOUT to the new value c and call the routine again. When continuing an integration where INFO(1) = 1, never modify T, Y, WK, IWK, INFO(3), and INFO(4). However, INFO(2), RERR, AERR, RD, and ID may be modified at any time.

Intermediate Output Mode. If one wishes to study the behavior of the solution y(t) as the routine steps from T to TOUT, then set INFO(2) = 1. Then SFODE will stop after each successful step (yielding IERR = 1) until TOUT is reached. One may switch from the standard mode of operation (INFO(2) = 0) to the intermediate output mode (INFO(2) = 1) or visa versa at any time.

Remark. The diagnostic IERR = −1 does not state that 500 steps have been taken on the current call to SFODE. On an initial call to the routine the step counter is set to 0. On continuation calls, the counter continues to increase until 500 steps have accumulated. When IERR = −1 is reported, the counter is reset to 0, and only then does the step counting begin again.

Input Errors. IERR is set to one of the following values when an input error is detected.

IERR = −33	$n < 1$
IERR = −34	RERR < 0
IERR = −35	AERR < 0
IERR = −36	The routine has been called with TOUT, but it has also been told not to step past the point TSTOP.
IERR = −37	T = TOUT. This is not permitted on continuation calls.
IERR = −38	The user has modified T.
IERR = −39	TOUT is not beyond T. An attempt is being made to change the direction of integration without restarting.
IERR = −40	The jacobian matrix is banded. However, m_ℓ and m_u do not satisfy $0 \leqslant m_\ell < n$ and $0 \leqslant m_u < n$.
IERR = −41	$\ell < 250 + 10n + n^2$
IERR = −42	$\ell < 250 + 10n + (2m_\ell + m_u + 1)n$
IERR = −43	$m < 55 + n$
IERR = −44	INFO(1) is incorrect.

After the error is corrected, set INFO(1) = 0 and call the routine again.

Error Control. Assuming that SFODE has the correct value for y(t), let e_i denote the error generated in computing $y_i(t + h)$ for $i = 1,\ldots,n$ when SFODE steps from t to t + h. The routine attempts at each step to maintain the accuracy $\frac{1}{n} \Sigma_i (e_i/w_i)^2 \leqslant 1$ where w_i = RERR $|y_i(t)|$ + AERR. When this criterion is satisfied, $|e_i| \leqslant \sqrt{n}\, w_i$ for $i = 1,\ldots,n$. This criterion includes as special cases relative error (AERR = 0) and absolute error (RERR = 0). However, if AERR = 0 and $y_i(t) = 0$ for some i, then this criterion cannot be applied and IERR = −3 occurs.

When proceeding from T to TOUT, the routine continually readjusts its order and step size so as to maintain accuracy at each step. However, no attempt is made to control the progressive erosion of accuracy that can occur when the steps accumulate. Since the erosion of accuracy can be significant, at times one may wish to double-check the results. If the problem is nonstiff or mildly stiff for an interval, then the best procedure is to compare the results obtained by SFODE with those obtained by ODE for the interval. ODE normally maintains greater accuracy than SFODE. However, if the problem is extremely stiff then rerun the problem with SFODE. On the second run, request greater accuracy.

Programming. SFODE calls the subroutines STFODE and ZZZJAC. STFODE employs the subroutines LSOD1, HSTART, INTYD, STOD, CFOD, PJAC, SLVS, SGBFA, SGBSL, SGEFA, SGESL, SAXPY, and SSCAL, and the functions VNORM, VNWRMS, ISAMAX, SDOT, and SPMPAR. The routines save and exchange information in a labeled common block having the block name DEBDF1. STFODE is a modification by A. H. Morris of the subroutine DEBDF, designed by L. F Shampine and H. A. Watts (Sandia Laboratories). DEBDF appears in the SLATEC library. STFODE is a driver for a modification of the code LSODE, written by A. C. Hindmarsh (Lawrence Livermore Laboratory).

CALL SFODE1(F,n,Y,T,TOUT,INFO,RERR,AERR,IERR,WK,ℓ,IWK,m,RD,ID)

SFODE1 differs from SFODE only in the treatment of RERR and AERR. In SFODE1, RERR and AERR are arrays of dimension n. RERR(i) and AERR(i) are relative and absolute error tolerances to control the accuracy of the i^{th} solution component $y_i(t)$ for $i = 1,...,n$. Let e_i denote the error generated in the computation of $y_i(t+h)$ from $y_i(t)$ when SFODE1 steps from t to t + h. Then SFODE1 attempts at each step to maintain the accuracy $\frac{1}{n}\Sigma_i(e_i/w_i)^2 \leqslant 1$ where $w_i =$ RERR(i) $|y_i(t)|$ + AERR(i). When this criterion is satisfied $|e_i| \leqslant \sqrt{n}\, w_i$ for $i = 1,...,n$. However, if AERR(i) = 0 and $y_i(t) = 0$ for some i, then the criterion cannot be applied and IERR = −3 occurs.

When IERR references RERR and AERR, the settings for IERR provide the following information:

IERR = −2	The accuracy required by RERR and AERR is too stringent. RERR and AERR have been modified by the routine. RERR and AERR may be further modified by the user if he desires. To continue, call SFODE1 again.
IERR = −3	SFODE1 stopped when y_i became 0 and AERR(i) = 0. INFO(1) was set to −i. To continue set AERR(i) to be positive, INFO(1) = 1, and call the routine again.
IERR = −34	(Input error) RERR(i) < 0 for some i.
IERR = −35	(Input error) AERR(i) < 0 for some i.

RERR and AERR may be modified on any continuation call to SFODE1.

Programming. SFODE1 calls the subroutines STFODE and ZZZJAC.

FOURTH-ORDER RUNGE-KUTTA

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $y(t) = (y_1(t),\dots,y_n(t))$. Assume that $y(t_0)$ is known. Then for a small real number h, the subroutine RK is available for computing $y(t_0 + h)$. RK employs the standard fourth-order Runge-Kutta procedure.

<u>CALL RK(n,T,h,A,F)</u>

T is a variable having the value t_0 and A is an array of dimension 3n or larger. It is assumed that A(1),...,A(n) contain the values $y_1(t_0),\dots,y_n(t_0)$. If $h = 0$ then RK computes the derivatives $y_1'(t_0),\dots,y_n'(t_0)$ and stores them in A(n + 1),...,A(2n). If $h \neq 0$ then it is assumed that the derivatives $y_1'(t_0),\dots,y_n'(t_0)$ have already been computed and stored in A(n + 1),...,A(2n). In this case, when RK is called, the values $y_1(t_0 + h),\dots,y_n(t_0 + h)$ and derivatives $y_1'(t_0 + h),\dots,y_n'(t_0 + h)$ are computed and stored in A(1),...,A(2n). Also T is reset to the value $t_0 + h$.

The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Z)

Z is an array of dimension n or larger where Z(1),...,Z(n) contain the values $y_1(t),\dots,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),\dots,y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in Z(1),...,Z(n). F must be declared in the calling program to be of type EXTERNAL.

Note. The area A(2n + 1),...,A(3n) serves as a work space for the routine.

Programmer. A. H. Morris

Example. Consider the equations

$$x'(t) = y(t)$$
$$y'(t) = -x(t)$$

where $x(0) = 0$ and $y(0) = 1$. The following code may be used for solving these equations at the points .01, .02,..., 1.00, and storing the results in the arrays X and Y.

```
DIMENSION A(6), X(100), Y(100)
EXTERNAL FUN
T = 0.0
H = .01
A(1) = 0.0
A(2) = 1.0
A(3) = 1.0
A(4) = 0.0
```

```
      DO 10 I = 1, 100
      CALL RK(2,T,H,A,FUN)
      X(I) = A(1)
   10 Y(I) = A(2)
```

Here FUN may be defined by:

```
      SUBROUTINE FUN(T,Z)
      DIMENSION Z(2)
      X = Z(1)
      Y = Z(2)
      Z(1) = Y
      Z(2) = -X
      RETURN
      END
```

Note that the statements A(3) = 1.0 and A(4) = 0.0, which store the derivatives $x'(0)$ and $y'(0)$ in A(3) and A(4), can be replaced with CALL RK(2,T,0.0,A,FUN).

EIGHTH-ORDER RUNGE-KUTTA

Let $y'(t) = f(t, y(t))$ denote a system of n ordinary first order differential equations where $y(t) = (y_1(t),...,y_n(t))$. Assume that $y(t_0)$ is known. Then for a small real number h, the subroutine RK8 is available for computing $y(t_0 + h)$.

CALL RK8(n,T,h,Y,DY,WK,F)

T is a variable having the value t_0, and Y and DY are arrays of dimension n or larger. It is assumed that Y(1),...,Y(n) contain the values $y_1(t_0),...,y_n(t_0)$. If $h = 0$ then RK8 computes the derivatives $y_1'(t_0),...,y_n'(t_0)$ and stores them in DY(1),...,DY(n). If $h \neq 0$ then it is assumed that the derivatives $y_1'(t_0),...,y_n'(t_0)$ have already been computed and stored in DY(1),...,DY(n). In this case, when RK8 is called, the values $y_1(t_0 + h),...,$ $y_n(t_0 + h)$ and derivatives $y_1'(t_0 + h),...,y_n'(t_0 + h)$ are computed and stored in Y and DY. Also T is reset to the value $t_0 + h$.

WK is an array of dimension 8n or larger that is used for a work space by the routine.

The argument F is the name of a user defined subroutine that has the format:

CALL F(t,Z)

Z is an array of dimension n or larger where Z(1),...,Z(n) contain the values $y_1(t),...,y_n(t)$ for the argument t. F computes the derivatives $y_1'(t),...,y_n'(t)$ using $y'(t) = f(t, y(t))$ and stores the results in Z(1),...,Z(n). F must be declared in the calling program to be of type EXTERNAL.

Algorithm. The routine employs formulae (8–12) given on p. 34 of the reference.

Remarks. RK8 is used in the same manner as the routine RK. RK8 takes more time and storage than RK, but may be more accurate.

Reference. Shanks, E. B., "Solutions of Differential Equations By Evaluations of Functions," ***Math. Comp. 20***, 1966, pp. 21-38.

Programmer. A. H. Morris

SEPARABLE SECOND-ORDER ELLIPTIC EQUATIONS ON RECTANGULAR DOMAINS

Given a separable elliptic equation

$$a(x)\, u_{xx} + b(x)\, u_x + c(x)\, u + d(y)\, u_{yy} + e(y)\, u_y + f(y)\, u = g(x,y)$$

on the rectangle $a_1 \leqslant x \leqslant a_2$, $b_1 \leqslant y \leqslant b_2$, where u is periodic in x or y, or u or its normal derivative $\partial u/\partial n$ is given on each of the edges. For m, n > 1 let $x_i = a_1 + (i-1)h$ and $y_j = b_1 + (j-1)k$ where $h = (a_2 - a_1)/(m-1)$, $k = (b_2 - b_1)/(n-1)$, $i = 1,\dots,m$, and $j = 1,\dots,n$. Then the subroutine SEPDE is available for computing u at the points (x_i,y_j).

<u>CALL SEPDE(COFX,COFY,g,ITYPE,BVAL,IORD,a_1,a_2,m,b_1,b_2,n,
U,ku,W,ℓ,IERR)</u>

It is assumed that $m \geqslant 7$ and $n \geqslant 6$. U is an $m \times n$ matrix. The argument ku is the number of rows in the dimension statement for U in the calling program. When SEPDE is called, if the elliptic equation is solved then $U(i,j) = u(x_i,y_j)$ for $i = 1,\dots,m$ and $j = 1,\dots,n$.

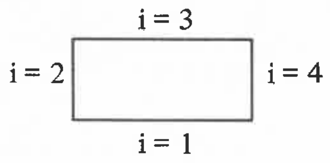
The input argument IORD is the order of the approximation procedure to be used. IORD may have the values 2 or 4.

The argument COFX is the name of a user defined subroutine that has the format:
CALL COFX(x,A,B,C)
A, B, and C are variables. COFX sets A = a(x), B = b(x), and C = c(x) for the argument x. COFX must be declared in the calling program to be of type EXTERNAL.

The argument COFY is the name of a user defined subroutine that has the format:
CALL COFY(y,D,E,F)
D, E, and F are variables. COFY sets D = d(y), E = e(y), and F = f(y) for the argument y. COFY must be declared in the calling program to be of type EXTERNAL.

The argument g is the name of a user defined function, where g(x,y) gives the right hand side of the elliptic equation for all $a_1 \leqslant x \leqslant a_2$, $b_1 \leqslant y \leqslant b_2$. The argument g must be declared in the calling program to be of type EXTERNAL.

Boundary conditions. The edges of the rectangular domain are labeled in a clockwise manner as follows:



$$\begin{aligned} \text{edge } 1 &= \{(x,b_1) : a_1 \leqslant x \leqslant a_2\} \\ \text{edge } 2 &= \{(a_1,y) : b_1 \leqslant y \leqslant b_2\} \\ \text{edge } 3 &= \{(x,b_2) : a_1 \leqslant x \leqslant a_2\} \\ \text{edge } 4 &= \{(a_2,y) : b_1 \leqslant y \leqslant b_2\} \end{aligned}$$

ITYPE is an array of dimension 4. For edge i (i = 1,...,4), ITYPE(i) specifies the type of boundary condition on the edge. ITYPE(i) must be set by the user to one of the following values:

ITYPE(i) = 0	It is assumed that u is given on the edge.
ITYPE(i) = 1	If i = 1 or i = 3 then u_y is given on the edge. Otherwise, if i = 2 or i = 4 then u_x is given on the edge.
ITYPE(i) = −1	If i = 1 or i = 3 then it is assumed that u is periodic in y; i.e., $u(x, y + b_2 - b_1) = u(x,y)$ for all x,y. In this case ITYPE(i) must be −1 for both i = 1 and i = 3. If i = 2 or i = 4 then it is assumed that u is periodic in x; i.e., $u(x + a_2 - a_1, y) = u(x,y)$ for all x,y. In this case ITYPE(i) must be −1 for both i = 2 and i = 4.

The argument BVAL is the name of a user defined function. BVAL(i,x,y) is defined for any point (x,y) on edge i when ITYPE(i) = 0 or 1, where

$$\mathrm{BVAL}(i,x,y) = \begin{cases} u(x,y) & \text{if } \mathrm{ITYPE}(i) = 0 \\ u_y(x,y) & \text{if } \mathrm{ITYPE}(i) = 1 \quad (i = 1 \text{ or } i = 3) \\ u_x(x,y) & \text{if } \mathrm{ITYPE}(i) = 1 \quad (i = 2 \text{ or } i = 4) \end{cases}$$

The function BVAL(i,x,y) is ignored when ITYPE(i) = −1. BVAL must be declared in the calling program to be of type EXTERNAL.

W is an array of dimension ℓ that is a work space. The argument ℓ is a variable whose value depends on IORD, m, n, and the types of boundary conditions used. Let ν be the largest integer $\leqslant \log_2 n$ and $\ell_1 = (\nu - 1)\, 2^{\nu+2} + \nu + 14m + 12n + 6$. Then

$\ell \geqslant \ell_1$ if IORD = 2

$\ell \geqslant \ell_1 + mn$ if IORD = 4.

When the routine terminates, ℓ will have been reset to the actual amount of storage needed.

IERR is a variable that reports the status of the results. When SEPDE terminates, IERR has one of the following values:

IERR = 0	The solution U was obtained.
IERR = −1	A constant (which is stored in W(1)) was subtracted from the right hand side of the equation in order to obtain a solution U. The solution is a weighted minimal least squares solution of the original problem.
IERR = 1	(Input error) $a_1 \geqslant a_2$ or $b_1 \geqslant b_2$.
IERR = 2	(Input error) ITYPE(i) $\neq 0, \pm 1$ for some edge i.
IERR = 4	The approximating linear system of equations is not diagonally dominant. This cannot occur when m and n are sufficiently large. Increase m and n, and reset ℓ.
IERR = 5	(Input error) ku < m
IERR = 6	(Input error) m < 7

IERR = 7	(Input error) $n < 6$
IERR = 8	(Input error) IORD $\neq$ 2, 4
IERR = 10	(Input error) $a(x)\,d(y) \leqslant 0$ for some interior point (x,y) of the rectangle. This violates the assumption that the equation is elliptic.
IERR = 11	(Input error) ℓ was too small. ℓ has been reset to the minimum amount of storage needed for W.
IERR = 12	(Input error) ITYPE(i) = -1 for edge 1 or 3, but not for both edges.
IERR = 13	(Input error) ITYPE(i) = -1 for edge 2 or 4, but not for both edges.

Precision. If IORD = 2 then the elliptic equation is approximated by a set of linear equations using finite differences. Otherwise, if IORD = 4 then the approximating equations are obtained by deferred corrections. The most accuracy is achieved when ITYPE(i) = 1 boundary conditions are not involved. For $m, n \geqslant 100$, 3-4 digit accuracy may be attained when IORD = 2 and 7-8 digit accuracy when IORD = 4. When ITYPE(i) = 1 boundary conditions are used, then for $m, n \geqslant 100$, 2-3 digits may be attained when IORD = 2 and 5-6 digits when IORD = 4.

Programming. SEPDE is an interface by A. H. Morris for SEPELL, a modification of the routine SEPELI described in the reference. SEPELI was developed by John C. Adams, being supported (in part) by codes written by Paul Swarztrauber and Roland Sweet (National Center for Atmospheric Research, Boulder, Colorado). SEPDE employs the subroutines PDEDGE, SEPELL, SEPEL1, CHKPRM, CHKSNG, ORTHG, MINSOL, TRISP, DEFER, DXFN, DYFN, BLKTRI, BLKTR1, COMPB, PROD0, PRODP, CPROD0, CPRODP, INDXA, INDXB, INDXC, PPADD, TQLRT0 and functions PSGF, BSRH, PPSGF, PPSPF, SPMPAR. The routines exchange information in the labeled common blocks having block names CBLKT and SPLP.

Example. Consider $(1 + x)^2\, u_{xx} - 2(1 + x)\, u_x + u_{yy} = 3(1 + x)^4 \sin y$ for $0 \leqslant x \leqslant 1$ and $|y| \leqslant \pi$

where $u_x(0,y) = 4 \sin y \quad |y| \leqslant \pi$
$u(1,y) = 16 \sin y$

and u is periodic in y. This problem has the solution $u = (1 + x)^4 \sin y$. Let

ITYPE(1) = -1
ITYPE(2) = 1
ITYPE(3) = -1
ITYPE(4) = 0.

Then the following routines and functions may be used for describing the problem. (Here g = GVAL.)

```
      SUBROUTINE COFX (X, A, B, C)
      T =  1.0 + X
      A =  T*T
      B =  -2.0*T
      C =  0.0
      RETURN
      END

      SUBROUTINE COFY (Y, D, E, F)
      D =  1.0
      E =  0.0
      F =  0.0
      RETURN
      END

      REAL FUNCTION GVAL (X, Y)
      GVAL = 3.0*(1.0 + X)**4*SIN(Y)
      RETURN
      END

      REAL FUNCTION BVAL (I, X, Y)
      BVAL = 4.0*SIN(Y)
      IF (I .EQ. 4) BVAL = 4.0*BVAL
      RETURN
      END
```

COFX, COFY, GVAL, and BVAL must be declared in the calling program to be of type EXTERNAL.

Reference. Adams, J., Swarztrauber, P., and Sweet, R., ***FISHPAK: Efficient FORTRAN Subprograms for the Solution of Separable Elliptic Partial Differential Equations, Version 3.*** National Center for Atmospheric Research, Boulder, Colorado, 1978.

UNIFORM RANDOM NUMBER GENERATOR

The following subroutine is available for generating a sequence of uniform variates in the interval (0,1).

CALL URNG(ix,A,n,IERR)

The argument n is the number of variates to be generated. A is an array of dimension n or larger, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leqslant ix < 2^{31} - 1$. When URNG is called, if no input errors are detected then IERR is set to 0 and n uniform variates are stored in A. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leqslant 0$ and IERR = 2 if ix is not a proper seed.

Usage. A given seed always initiates the same set of variates. Thus, the following two sets of instructions

```
(1)         IX = 103
            CALL URNG(IX,A,30,IERR)

            IX = 103
(2)         CALL URNG(IX,A,20,IERR)
            CALL URNG(IX,A(21),10,IERR)
```

generate the same 30 variates.

Remark. It is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leqslant 2^{31} - 1$.

Programming. Written by Linus Schrage (University of Chicago). Adapted by A. H. Morris.

Reference. Schrage, Linus, "A More Portable Fortran Random Number Generator," ***ACM Trans. Math Software*** *5* (1979), pp. 132-138).

GAUSSIAN RANDOM NUMBER GENERATOR USING THE BOX-MULLER TRANSFORMATION

The following subroutine is available for generating a sequence of normal variates from a normal distribution with mean 0 and standard deviation 1.

CALL NRNG(ix,A,n,IERR)

The argument n is the number of variates to be generated. A is an array of dimension n or larger, and ix and IERR are variables. On input, ix is an integer (called a ***seed***) for initializing the sequence of variates. It is assumed that $1 \leqslant ix < 2^{31} - 1$. When NRNG is called, if no input errors are detected then IERR is set to 0 and n normal variates are stored in A. On output, ix is a new seed for generating more variates.

Error Return. IERR = 1 if $n \leqslant 0$ and IERR = 2 if ix is not a proper seed.

Algorithm. When NRNG is called, an even number of uniform variates $u_1,\ldots,u_m$ is generated ($m = n$ if n is even and $m = n + 1$ if n is odd). Then the Box-Muller transformation

$$a_k = \sqrt{-2 \ln u_k}\ \cos 2\pi u_{k+1}$$

$$a_{k+1} = \sqrt{-2 \ln u_k}\ \sin 2\pi u_{k+1} \qquad (k = 1, 3, 5, \ldots)$$

is applied to obtain n normal variates $a_1,\ldots,a_n$. This transformation generates pairs of normal variates (a_1,a_2), (a_3,a_4), ... from the corresponding pairs of uniform variates. If n is odd then only the first variate of the final pair (a_n,a_{n+1}) is computed.

Usage. A given seed always generates the same sequence of ***uniform*** variates. Thus, if we consider the following three sets of instructions

```
(1)     IX = 73
        CALL NRNG(IX,A,30,IERR)

        IX = 73
(2)     CALL NRNG(IX,A,10,IERR)
        CALL NRNG(IX,A(11),20,IERR)

        IX = 73
(3)     CALL NRNG(IX,A,9,IERR)
        CALL NRNG(IX,A(10),20,IERR),
```

then we note that (1) and (2) generate the same 30 normal variates. Also (3) produces 29 of these 30 variates, skipping the 10th normal variate. The reason for this is that the request in (3) for 9 normal variates requires 10 uniform variates to be generated. The 10 uniform variates could be used for computing 10 normal variates (as is done in (2)). However, since only 9 normal variates are requested, computation of the 10th normal variate that would usually be generated is skipped.

Remark. NRNG calls the subroutine URNG for the uniform variates. Thus, it is assumed that the integer arithmetic being used handles all integers i in the interval $|i| \leqslant 2^{31} - 1$.

Programmer. A. H. Morris

DISTRIBUTION

	Copies
Polaroid Corporation 2 Osborn Street Attn: M. Abdulwahab Mail Stop — 1M Cambridge, MA 02139	1
Virginia Polytechnic Institute and State University Department of Computer Science Attn: Dr. Donald C.S. Allison Blacksburg, VA 24061	1
E.T.S. Ingenieros Industriales Dept. de Energia Nuclear Attn: Rosario Arroyo P. de la Castellana, 80 Madrid 6, Spain	1
Old Dominion University Department of Mathematics Attn: Dr. Howard W. Baeumler Norfolk, VA 23508	1
Aerojet ElectroSystems P.O. Box 296 Attn: Maryann Becker 160/4343 Azusa, CA 91702	1
System Development Corporation Satellite Programs Branch Attn: Dr. John R. Berg 2500 Colorado Avenue Santa Monica, CA 90406	1
IBM Federal Systems Division Attn: Dr. Terry D. Boldt 2625 Townsgate Rd. Westlake Village, CA 91361	1
Unilever Australia Ltd. Technical Information Department Attn: Vivien Bowman P.O. Box 9 Balmain 2041 Reynolds Street Balmain NSW, Australia	1
Westinghouse Defense and Electronics Center Attn: Angela M. Brusca P.O. Box 746 MS 1297 Baltimore, MD 21203	1
Naval Coastal Systems Laboratory Attn: Code 760 (C.M. Callahan) Panama City, FL 32401	1
General Mills, Inc. James Ford Bell Technical Center Attn: Mark Chatterton 9000 Plymouth Avenue, North Minneapolis, MN 55427	1
U.S. Army Engineer Waterways Experiment Station Coastal Engineering Research Center Attn: H.S. Chen Vicksburg, MS 39180	1
Georgia Institute of Technology Office of Computing Services Attn: Rand H. Childs M.C. Trauner Atlanta, GA 30332	1 1
Vandenberg Air Force Base ITT Federal Electric Corporation Attn: Chuck Converse DS 450 P.O. Box 1886 Lompox, CA 93437	1
System Development Corporation Attn: Elaine Denton (72-31) 2500 Colorado Ave. Santa Monica, CA 90406	1
Naval Coastal Systems Center Attn: Code 4210 (Joyce Elliott) Panama City, FL 32407	1
Naval Underwater Systems Center Attn: Code 714 (Gerald J. Elias) Newport, RI 02840	1

	Copies
Naval Underwater Systems Center Attn: Code PA-4 (Tom Galib) New London, CT 06320	1
Network Analysis Associates, Inc. Attn: J.D. Gaski P.O. Box 8007 Fountain Valley, CA 92728	1
Control Data Corporation Attn: William E. Glass Mail Station HQC01P 8100 34th Avenue, South Bloomington, MN 55431	1
Webb Institute of Naval Architecture Attn: Martin A. Goldberg Crescent Beach Road Glen Cove, NY 11542	1
Queen's University Computing and Communications Services Attn: Donna Hamilton Kingston K7L 3N6, Ontario Canada	1
Polaroid Corporation Attn: Cheryl Healy 750 Main Street 1-B Cambridge, MA 02139	1
Coastal Engineering Research Center Department of the Army Attn: Barry E. Herchenroder Kingman Bldg. Fort Belvoir, VA 22060	1
Litton Systems Inc. Amecon Division Attn: John L. Houser MS 2-23 5115 Calvert Road College Park, MD 20740	1
Norton Company Chemical Process Products Division Attn: Alex Hsia P.O. Box 350 Akron, OH 44309	1

	Copies
Westinghouse Defense and Electronics Center Attn: Joshua C. Hung P.O. Box 746 Baltimore, MD 21203	1
Rijksuniversiteit Utrecht Academisch Computer Centrum Utrecht Attn: Jan van Kats Budapestlaan 6 Postbus 80.011 3508 TA Utrecht, Netherlands	1
Diamond Shamrock Corporation Computer Services Attn: Ted E. Keller P.O. Box 348 Painesville, OH 44077	1
Special Products Development Systems Engineering and Applications Division TRW Defense Systems Group Attn: Manfred Kory 7600 Colshire Drive McLean, VA 22102	1
Case Western Reserve University School of Medicine Department of Biometry Attn: Dr. Robbin B. Lake Cleveland, OH 44106	1
Acadia University Computer Centre Attn: Kim Leonard Wolfville, Nova Scotia, Canada BOP 1X0	1
Mary Washington College Mathematics Department Attn: Dr. Stephen Lipscomb Fredericksburg, VA 22401	1
Naval Weapons Station Attn: NQEC — Code 302 (Jim Liverman) Yorktown, VA 23691	1

Copies

Internationale Atomreaktorbau GmbH
Interatom
Attn: Mr. Luigs 1
9130 EDV und Mathematik
Postfach 5060
Bergisch Gladbach 1
West Germany

Georgia Institute of Technology
Electronics Research Bldg.
GTRI/MCSF
Attn: Gerald F. Mackey 1
Lee Gantt 1
Atlanta, GA 30332

Mr. Eugene Maguin 1
234 S. Magnolia
Lansing, MI 48912

Massachusetts Institute of Technology
Lincoln Laboratory
Attn: Dr. Eugene J. Mallove 1
P.O. Box 73
Lexington, MA 02173

Mayo Foundation
Medical Sciences Computer Facility
Attn: John M. McKinley 1
3-16 Medical Science (Wes von Nurden) 1
200 First Street SW
Rochester, MN 55905

Naval Ocean Systems Center
Attn: Code 9122 (Chuck Messinger) 1
San Diego, CA 92152

Naval Research Laboratory
Attn: Code 2303 (Alvin B. Owens) 1
Code 2303.1 (Allen R. Miller) 1
Washington, DC 20375

Virginia Polytechnic Institute and
State University
Department of Computer Science
Attn: Dr. Richard E. Nance 1
Blacksburg, VA 24061

Copies

Easams LTD
Attn: K.D. Needham 1
Lyon Way, Frimley Road
Camberley GU16 5EX, Surrey
England

Westinghouse Electric Corporation
Attn: Dr. Steve Orbon 1
Box 158
Madison, PA 15663

Pratt and Whitney Aircraft
Attn: Code EB2C (Scott M. Ramsey) 1
400 Main Street
East Hartford, CT 06108

Mr. John Reed 1
Mail Code A1
16441 Space Center Blvd.
Houston, TX 77058

Correios e Telecomunicacoes de Portugal
R.S. Jose, 10-70
Attn: Vitor Rodrigues 1
1198 Lisboa Codex
Portugal

Universitat Dortmund
FB Informatik, Systemanalyse
Attn: Dr. Ing. H.P. Schwefel 1
Postfach 500500
4600 Dortmund 50
West Germany

Merck and Company, Inc.
Attn: Jerome Starr 1
P.O. Box 2000
Rahway, NJ 07065

Science Applications Inc.
Attn: Dr. Fred Stern 1
134 Holiday Court
Suite 318
Annapolis, MD 21401

	Copies		Copies
Department of the Army Waterways Experiment Station Corps of Engineers Attn: Connie A. Stirgus P.O. Box 631 Vicksburg, MS 39180	1	Norwegian Contractors Attn: Per Urvik P.O. Box 40 Ankertorvet Oslo 1 Norway	1
Westinghouse Electric Corporation Attn: Vish Subramaniam Avenue A and West Street, Forrest Hills Pittsburg, PA 15221	1	Queen's University Dept. of Mathematics and Statistics Jeffrey Hall Attn: James H. Verner Kingston K7L 3N6, Ontario Canada	1
The Analysts Schlumberger Attn: Jerry Sychra 200 Macco Blvd. Sugar Land, TX 77478	1	Virginia Polytechnic Institute and State University Department of Computer Science Attn: Dr. Layne T. Watson Blacksburg, VA 24061	1
Naval Air Development Center Attn: Code 8511 (Dan Tarrant) Warminster, PA 18974	1	Norton Company Chemical Process Products Division Attn: Jan York P.O. Box 350 Akron, OH 44309	1
Energy, Mines, and Resources Canada Mining Research Laboratories Attn: Neil Toews 555 Booth Street Ottawa, Ontario K1A0G1 Canada	1	Library of Congress Attn: Gift and Exchange Division Washington, DC 20540	4
Aerojet ElectroSystems Company Attn: D.S. Toomb 1100 West Hollyvale Street P.O. Box 296-III Azusa, CA 91702	1		

Internal Distribution:

C	1	E211 (Sullivan)	1	H02	1	N20	1
D	1	F	1	H10	1	N30	1
D1	1	F01	1	H20	1	N40	1
D101	1	F10	1	H30	1	R	1
D2	1	F20	1	K	1	R01	1
D21	1	F30	1	K02	1	R10	1
D22	1	F40	1	K10	1	R30	1
D23	1	G	1	K20	1	R40	1
D24	1	G01	1	K30	1	U	1
D25	1	G10	1	K33	500	U01	1
E	1	G20	1	K40	1	U10	1
E02	1	G30	1	K50	1	U20	1
E10	1	G40	1	N	1	U30	1
E20	1	G60	1	N01	1	U40	1
E30	1	H	1	N10	1	X	1