Programs for Solving Linear Equations in the PORT Library

Linda Kaufman

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This paper describes the subroutines that have recently been inserted into the PORT library for solving linear systems. Some of the subroutines are high-level drivers which solve

$$AX = B$$

and indicate the sensitivity of the solution to perturbations in the problems. Others are low level subroutines designed for complicated problems such as solving a sequence of problems with the same matrix but with different right-hand sides, which depend on previous solutions. The subroutines are classified on the basis of the structure of the A matrix, e.g. whether it is symmetric, banded, sparse, etc.

February 11, 1993

Programs for Solving Linear Equations in the PORT Library

Linda Kaufman

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

The linear algebra chapter in the PORT library contains many routines for solving linear equations and linear least squares problems. Some are high-level drivers (see section 2.1) designed for those users who simply want to solve a system of equations

$$AX = B$$

Others are lower-level routines (see section 2.2) designed for users with slightly more complicated problem. e.g., users who wish to solve a sequence of linear systems with the same matrix but different right-hand sides.

Presently, the package is divided into the following categories, with the first two letters in the name of each subroutine specifying the type of matrix.

Table 1

Category	2 letter prefix	Meaning
General	GE	no known specific properties
Symmetric	SY	$A(i,j) = A(j,i)$
Banded	BA	all non-zero elements are near the main diagonal
Banded, symmetric, positive definite	BP	banded and symmetric, as above, and all eigenvalues positive
Sparse	SP	the placements of all nonzero elements are known and they occupy not more than about 1% of the matrix

Associated with each category is a distinct data structure. For example, for symmetric matrices, the upper triangular portion only is stored in one long vector. In section 3 the data structures are discussed in detail.

Note that we have not implemented a subroutine for computing A^{-1}. If necessary, the matrix A^{-1} can be found by solving $AX = I$.

The subroutines in this package have been implemented in single precision, double precision, and complex arithmetic.

2. Structure of the Package

2.1. Higher-Level Drivers

Associated with each category in Table 1 are two high-level subroutines — the Linear Equation solver (LE) and the System Solver (SS) which call lower-level subroutines.

Both the SS subroutines (GESS, SYSS, BASS, etc.) and the LE subroutines (GELE, SYLE, BALE, etc.) solve systems

$$AX = B \tag{2.1}$$

using variants of Gaussian elimination adapted to the structure of the matrix[8]. For example, partial pivoting is used for general matrices but no pivoting is used for band symmetric positive definite matrices. The matrix B in (2.1) may have several columns. In all the subroutines the matrix A is overwritten, and the solution replaces the right-hand side matrix B.

Besides solving (2.2), each of the SS subroutines returns an estimate of the condition number (COND) of the matrix A. The condition number provides a measure of the sensitivity of X to errors in A and B. The larger it is, the more ill conditioned the matrix. If one is on a machine having d decimal digits of precision, then normally one may expect the elements of X to have at most $d - \log_{10}(\mathrm{COND})$ correct decimal digits, although the accuracy also depends on the right-hand sides. (See section 2.2 for further details.)

Users are generally urged to use the SS subroutines rather than the LE subroutines. If the matrix is known to be well conditioned, some computation time can be saved by using the LE subroutine. However, the added cost for computing the condition number with an SS subroutine should rarely be a deterrent. In general, if solving (2.1) is expensive, the cost of obtaining the condition estimate will be relatively inconsequential. If solving (2.1) is rather inexpensive, the cost of obtaining the condition estimate will indeed be noticeable, but rarely worrisome. Since the time required to compute the condition estimate is roughly equivalent to the additional time that would be required by the LE subroutines if B were augmented with two more columns, the overhead decreases as the number of columns in B increases. Of course the ratio is dependent on the structure of the matrix. For general and symmetric systems, the added cost decreases as the size of the system increases. With small systems (say 10×10) with one right-hand side, the LE subroutine might execute in half the time for SS, while for larger systems (say 100×100) there might be only a 10% saving. For banded and sparse systems the added cost of SS over LE depends on the sparsity of the matrix, i.e. the ratio of zero to nonzero elements in A. The operation counts given in the example in the Port user sheet for BALE suggest that the penalty for BASS for narrow banded matrices with one right hand side is quite noticeable.

2.2. Lower-Level Subroutines

The lower-level subroutines are the building blocks for the higher level subroutines. Casual users will probably not use them directly and may safely skip this section. However some people have complicated problems which cannot be handled by the high-level drivers, while others may wish to have more control over the process.

The solving of a linear system is composed of two phases:

(1) The decomposition (DC) of the coefficient matrix or a permutation thereof into the product of two or three matrices which have simple structures. For example, one might factor A as

$$PA = LU \tag{2.2}$$

where P is a permutation matrix, L is a lower triangular matrix and U is an upper triangular matrix.

(2) The solving of the decomposed system. Usually the matrices have been decomposed into the product of 2 triangular matrices and one usually thinks of forward solving (FS) with the first matrix and back solving (BS) with the second matrix. Thus if $PA = LU$ and we wish to solve $Ax = b$

we will forward solve

$$Ly = Pb \tag{2.3}$$

and then back solve

$$Ux = y \tag{2.4}$$

A typical LE routine would call a DC subroutine, an FS subroutine, and then a BS subroutine. For example, GELE calls first GEDC, then GEFS, and finally GEBS.

During the decomposition phase, the matrix A is often permuted to promote stability. The permutation matrix P can be obtained by unraveling the n-vector INTER, an output parameter of the DC

subroutines. For an $n \times n$ matrix Gaussian elimination proceeds in $n-1$ stages. At stage k a particular row is interchanged with row k and elements below the diagonal in the k^{th} column are zeroed out. In our subroutines at the k^{th} stage we interchange rows k and i for some $i \geq k$ and set INTER(k) to i. Thus if pivoting is never done, INTER(k)=k for $1 \leq k \leq n-1$. Because interchanges after the k^{th} stage of the elimination process are not applied to the first k columns of the L matrix, in the FS subroutines the exact order in which the interchanges were performed must be readily accessible. Note that we do not use the common alternative of initializing INTER(i) to i and interchanging the elements of that vector. In INTER(n) we return +1 if there has been an even number of interchanges and −1 if there has been an odd number. This information is useful in determining the sign of the determinant (see the Port user sheet for GELU).

During the decomposition phase we decide whether the matrix is singular (or nearly singular). In exact arithmetic if a matrix were singular the DC subroutines would generate a U matrix with some zero diagonal elements. However, due to roundoff error,when A is singular generally no diagonal element of U will be exactly zero, but some diagonal element will be very small relative to the elements of A.

Since choosing a cutoff criterion for singularity is very difficult, a still lower level subroutine (GELU for general matrices) is provided which allows the user to specify his own criterion for singularity. These lowest level subroutines have an additional parameter EPS and they declare singular any matrix which produces a diagonal element of U whose magnitude is less than or equal to EPS. Because the choice of EPS often demands more expertise in numerical analysis than can reasonably be expected of a user, the DC subroutines compute a default value for EPS, namely

$$\text{EPS} = \max_j \left\{ \sum_i |a_{ij}| \right\} \varepsilon$$

where ε is the machine precision, and then call the lowest level subroutines.

In the lowest level subroutine, if for some index i

$$|u_{ii}| \leq \text{EPS}$$

then we set $u_{ii} = sign\ (u_{ii}) \times \text{EPS}$ and computation continues. If EPS is 0, the i^{th} column of L is set to the i^{th} column of the identity matrix and numerical problems are avoided. In general, users who legitimately wish to solve singular or near singular problems (e.g. those arising from the eigenvalue inverse iteration algorithm) may do so without encountering avoidable overflow and underflow. However, if the matrix is complex and the compiler implements complex division naively, users may encounter an overflow condition if the modulus of a diagonal element of U underflows.

Another group of lower level subroutines are the condition estimators (CE). As mentioned in section (2.1), the condition number measures the sensitivity of the solution to errors in the system. If the condition number is rather large, one should be wary of the solution to the linear system. Some people may be under the impression that the determinant is a good, cheap measure of conditioning. However, some very ill conditioned problems have very reasonable determinants. For example the determinant of the n by n matrix

$$\begin{matrix} 1 & -1 & \cdot & \cdot & \cdot & -1 \\ & 1 & -1 & \cdot & \cdot & -1 \\ & & 1 & -1 & \cdot & -1 \\ & & & & \cdot & \\ & & & & 1 & -1 \\ & & & & & 1 \end{matrix}$$

is 1, but its condition number is approximately 2^n. The condition number is defined as $||A||\ ||A^{-1}||$, where $||A|| = \max_{||x||=1} ||Ax||$. Our condition estimator does not compute A^{-1} explicitly, but estimates the size of its largest elements. It uses the algorithm defined in [2] which solves

$$A^T \mathbf{x} = \mathbf{b} \tag{2.4}$$

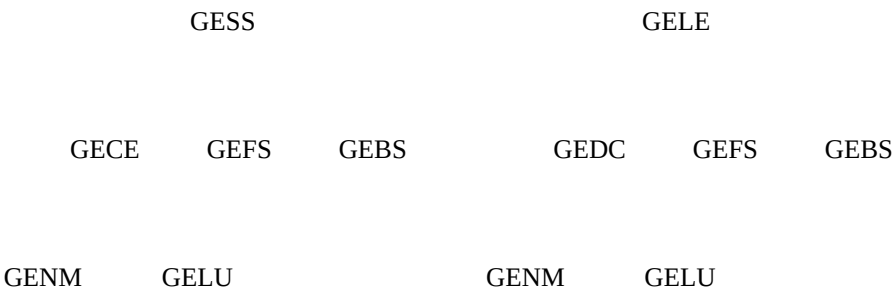
and

$$A\mathbf{y} = \mathbf{x}$$

with the vector **b** suitably chosen so that **y** looks like the column of A^{-1} with the largest elements. Since this algorithm requires a decomposition of *A*, each CE subroutine returns that decomposition along with the condition estimate.

Several additional families of subroutines have been included in the package. The NM (NorM) subroutines (GENM, BANM, BPNM, etc) return the norm of a matrix. The ML (MuLtiply) subroutines (GEML, BAML, BPML, etc.) form the product $y = Ax$, where *A* is a matrix having the structure indicated by the first 2 letters of the subroutine.

The following figure shows the hierarchy of subroutines for general matrices.



3. Data Structures for Special Cases

3.1. Symmetric Matrices

A real matrix A is symmetric if $a_{ij} = a_{ji}$. Separate subroutines for symmetric matrices are included in PORT because a recent paper [1] has shown it is possible to decompose a symmetric matrix with elementary transformations twice as efficiently as a nonsymmetric matrix.

The algorithm in [1] for symmetric matrices also requires the storage of only the upper triangular portion of the matrix. We store the upper triangle portion of a symmetric matrix A by rows in a vector C. For example the 4×4 matrix

$$\begin{matrix} 1 & 3 & 5 & 7 \\ 3 & 2 & 4 & 6 \\ 5 & 4 & 8 & 10 \\ 7 & 6 & 10 & 9 \end{matrix}$$

is stored as the vector

C = (1 3 5 7 2 4 6 8 10 9)

In FORTRAN, one can pack an $n \times n$ symmetric matrix A into a vector C using the following scheme:

```
      L=1
      DO 10 I=1,N
         DO 5 J=I,N
            C(L)=A(I,J)
            L=L+1
 5       CONTINUE
10    CONTINUE
```

Reading a matrix into a packed array is slightly more complicated. Of course the exact code depends on the information contained in the input file. If each record of the input file contains one row of the upper triangular portion of the matrix, the matrix can be read in as follows.

```
      LBEGIN=1
      DO 10 I=1,N
         LEND=LBEGIN+N–I
         READ(5,1)(C(L),L=LBEGIN,LEND)
1        <appropriate FORMAT statement>
         LBEGIN=LEND+1
10    CONTINUE
```

If each record of the file contains one row of the original matrix (both its upper and lower triangular portion), the following FORTRAN program fragment which uses an N-vector TEMP, can be used.

```
      L=1
      DO 20 I=1,N
           READ(5,1)(TEMP(J),J=1,N)
 1         <appropriate FORMAT statement>
           DO 10 J=I,N
                C(L)=TEMP(J)
                L=L+1
10         CONTINUE
20    CONTINUE
```

If each record of the file contains one row of only the lower triangular portion, the following FORTRAN program can be used.

```
      DO  20  I=1,N
           READ(5,1)(TEMP(J),J=1,I)
  1        < appropriate  FORMAT  statement)
C         TEMP(J)  NOW  CONTAINS  A(J,I)
           L=I
C         C(I)  WILL  CONTAIN  A(1,I)
           DO  10  J=1,I
                C(L)=TEMP(J)
                L=L+N–J
 10        CONTINUE
 20   CONTINUE
```

A complex matrix is considered complex symmetric if $a_{ij} = a_{ji}$, and complex Hermitian if $a_{ij} = \bar{a}_{ji}$, i.e. the imaginary parts have opposite signs across the diagonal. Subroutines beginning with CSY are designed for complex symmetric matrices and those beginning with CHE are designed for complex Hermitian matrices.

In the symmetric decomposition subroutines one decomposes A as

$$PAP^T = MDM^T$$

where P is a permutation matrix, M is a unit lower triangular matrix and D is a block diagonal matrix with blocks of order 1 and 2. The vector INTER, an output parameter of SYCE, SYDC, and SYMD, indicates not only the permutation array as in the GE decomposition subroutines but also the block structure of D. If D contains a 2×2 block beginning at row I, INTER(I+1) will be set to -1 and during the decomposition phase row INTER(I) was interchanged with row I+1. If D contains a 1×1 block beginning at row I, during the decomposition phase row INTER(I) was interchanged with row I.

3.2. Banded Matrices

A matrix A is banded if all its nonzero elements lie on diagonals close to the main diagonal as in

```
x x              x x
x x x            x x x
  x x x          x x x x
    x x x          x x x x
      x x            x x x
```

Fig. 1 **Fig. 2**

Separate programs have been included in PORT to cover general banded matrices because most

problems with banded matrices are very large and it is possible to save time and space by taking advantage of the structure.

In our subroutines the user must specify two quantities:

$$\begin{aligned} m_l,&\quad \text{the number of diagonals in the lower triangular} \\ &\quad \text{portion of } A \\ m,&\quad \text{the total number of diagonals.} \end{aligned} \tag{3.1}$$

In Figure 1 $m_l = 2$ and $m = 3$, and in Figure 2, $m_l = 3$ and $m = 4$.

Our subroutines and data structures are based on those in Martin and Wilkinson[6]. In those subroutines each diagonal of A occupies a column of an array; in ours each diagonal is stored in a *row* of an array, with the leftmost (lowest) diagonal occupying the first row. Basically a banded matrix A will be packed into an $m \times n$ array G according to the formula

$$\begin{aligned} G(m_l + j - i, i) &= a_{ij} \\ i &= 1,2,\ldots,n, \\ j &= i - m_l + 1,\ldots,i + m - m_l \end{aligned} \tag{3.2}$$

Since the expression $i - j$ is a constant on any diagonal, equation (3.2) turns a diagonal of A into a row of G. Also the nonzero portions of rows of A are turned into columns of G. For example the banded matrix

$$\begin{array}{cccccc} 11 & 12 & 13 & & & \\ 21 & 22 & 23 & 24 & & \\ & 32 & 33 & 34 & 35 & \\ & & 43 & 44 & 45 & 46 \\ & & & 54 & 55 & 56 \\ & & & & 65 & 66 \end{array}$$

will be stored as

$$\begin{array}{cccccc} & 21 & 32 & 43 & 54 & 65 \\ 11 & 22 & 33 & 44 & 55 & 66 \\ 12 & 23 & 34 & 45 & 56 & \\ 13 & 24 & 35 & 46 & & \end{array}$$

This storage scheme is admittedly complicated. Since the basic step of our algorithm adds a multiple of one row of the matrix A (i.e. a column of G) to another row and since FORTRAN stores 2 dimensional arrays by columns, this arrangement should prevent excessive paging because in any such step the elements referenced in G will be close together.

The user does not have to 'zero' the unused corners in the G matrix, since our subroutines will not use them.

To solve a linear system $AX = B$ with an $n \times n$ banded matrix A one first decomposes A as

$$PA = LU \tag{3.3}$$

where L is lower triangular and U is upper triangular as in (2.2) and P is a permutation matrix chosen to promote stability. The matrix U will have at most m diagonals (see (3.1)) and L will have at most m_l diagonals. Thus U can be stored in the space which A once occupied but extra space is needed for L. Since the diagonal of L contains all 1's, the actual storage space reserved for L need only be $(m_l - 1) \times n$ locations. The information needed to form P requires only $n - 1$ integer locations.

Typically after forming (3.3) one forward solves

$$LY = PB \tag{3.4}$$

and then back solves

$$UX = Y. \tag{3.5}$$

In our BALE subroutine, steps (3.3) and (3.4) are combined and there is no need to store either P or L. However lower level subroutines are provided for steps (3.3) and (3.4) separately for those users whose problems cannot be solved by BALE. Moreover, the driver BASS, which solves $AX = B$ and determines the condition number of A demands the separation of the two steps. Thus BASS requires more storage than BALE.

Our subroutine for computing U in (3.3) also keeps track of the number of diagonals in U. Because of pivoting it can have as many as m diagonals; but in the absence of pivoting it will have only $m - m_l + 1$ diagonals. Since there is no need during backsolving to worry about diagonals which are all zero, the variable MU determined in the decomposition subroutines tells BABS the number of nonzero diagonals U contains.

Users with problems involving matrices with narrow bands and using only a few right-hand sides, should acquaint themselves with the relative costs of BASS and BALE. The example in the Port user sheet for BALE gives some computational times which may be used as a guide.

3.3. Banded Symmetric Positive Definite Matrices

The subroutines beginning with the letters BP are designed for matrices A with three special properties:

(1) They are banded, i.e. their nonzero elements lie close to the main diagonal.

(2) They are symmetric, i.e. $a_{ij} = a_{ji}$.

(3) They are positive definite, i.e. all their eigenvalues are positive.

A matrix satisfying the property that the sum of the absolute values of the off diagonal elements in any row is less than the diagonal, is certain to be positive definite.

The matrix

$$\begin{matrix} 4 & 1 & & \\ 1 & 4 & 1 & \\ & 1 & 4 & 1 \\ & & 1 & 4 \end{matrix}$$

satisfies all these properties. We single out these matrices because they are often encountered in physical problems and the BP subroutines requires one third the storage and one third the time for the decomposition phase of the BA subroutines..

Our subroutines assume that A has been packed into the $m_l \times n$ matrix G according to the following rule:

$$G(j-i+1,i) = a_{ij} \quad \text{for } i = 1,...,n \; ; j = i,..,i+m_l-1.$$

where m_l is the number of nonzero diagonals on and below the diagonal of A. Hence the diagonal of A becomes the first row of G, the first subdiagonal becomes the second row, etc.

Thus the matrix

```
8  2  1
2  7  2  1
1  2  7  2  1
   1  2  7  2
      1  2  8
```

will be stored as

```
8  7  7  7  8
2  2  2  2
1  1  1
```

The BP subroutines do not touch the lower right-hand corner.

Our subroutines for banded symmetric positive definite matrices use Gaussian elimination without pivoting. Symmetric pivoting, which ruins the band structure of the matrix, is not needed for stability because the matrix is positive definite.

4. Sparse Matrices

A matrix is considered sparse if not more than about 1% of its elements are nonzero.

5. Relation to PFORT, PORT, and LINPACK

The linear algebra package achieves portability by using the PFORT subset of ANSI FORTRAN, as defined and enforced by the PFORT Verifier [8], and by using the machine constants module from the PORT Mathematical Subroutine Library[4] to characterize the host computer. This package also depends on PORT's dynamic storage allocation module to provide temporary working space, and on PORT's error handling module to respond to errors, which may be "fatal" or "recoverable". These three modules constitute the PORT kernel[5], which is in the public domain.

Concurrently with the development of the linear algebra subroutines in PORT, the author tested the programs in LINPACK[3], and greatly benefited from the experience. In fact, the idea of the condition estimator can be attributed directly to that effort. The major differences between the two libraries stem from the author's adherence to the philosophy of the PORT library. As a result, the PORT library includes high-level drivers, checks arguments when possible, and uses the PORT kernel when applicable. In some cases the two libraries use different data structures but both use essentially the same algorithms, which are elegantly explained in the LINPACK User's Guide[3].

6. References

1. J. R. Bunch and L. Kaufman, *Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems*, Math. Comp. 31, January 1977, pp. 163-179.
2. A. K. Cline, C. B. Moler, G. W. Stewart and J. H. Wilkinson, *An Estimate for the Condition Number of a Matrix*, SIAM J. Numer. Anal.16 April 1979, 368-375.
3. J.J. Dongarra, J. R. Bunch, C. B. Moler, G. W. Stewart, *Linpack Users' Guide*, SIAM, Philadelphia. 1979.
4. P.A. Fox, A.D. Hall, and N.L. Schryer, *The PORT Mathematical Subroutine Library*, ACM Transactions on Mathematical Software 4, pp. 104-126, June 1978.
5. P.A. Fox, A.D. Hall, and N.L. Schryer, *Algorithm 528:Framework for a Portable Library*, ACM Transactions on Mathematical Software 4, pp. 177-188, June 1978.
6. C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krough, *Basic Linear Algebra Modules*, ACM Transactions on Mathematical Software 5, pp. 308-325, Sept. 1979.

7. R.S. Martin and J.H. Wilkinson, *Solutions of Symmetric and Unsymmetric Band Equations and the Calculations of Eigenvectors of Band Matrices*, Numer. Math. 9, pp. 279-301, 1967.

8. B.G. Ryder, *The PFORT Verifier*, Software Practice and Experience 4, pp.359-377, October 1974.

9. J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

Appendix 1

GENERAL MATRICES

GEBS	-	Back Solve
GECE	-	Condition Estimation
GEDC	-	DeComposition
GEFS	-	Forward Solve
GELE	-	Linear Equation solution
GELU	-	LU decomposition
GEML	-	MuLtiplication
GENM	-	NorM
GESS	-	System Solution

Purpose: GEBS (GEneral matrix Back-Solve) solves AX = B where A is an upper triangular matrix. It can be used for the back solution phase of a general linear system solution. (It is used in this way by the routines GESS and GELE.)

Usage: CALL GEBS (N, A, IA, B, IB, NB)

N	→	the number of equations
A	→	the array, dimensioned (IA, KA) in the calling program, where IA ≥ N and KA ≥ N, containing the N × N upper triangular matrix The strictly lower triangular portion of A is not used or changed.
IA	→	the row (leading) dimension of A, as dimensioned in the calling program
B	→	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB ≥ N and KB ≥ NB
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides

Note: GEFS and GEBS can be used directly on the output matrix produced by GEDC, GELU, or GECE to solve a general linear system.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
3	IB < N
4	NB < 1
10 + k*	singular matrix with k^{th} diagonal element 0.0

Double-precision version: DGEBS with A and B declared double precision

Complex version: CGEBS with A and B declared complex

Storage: None

Time: $(N^2/2 - N/2) \times$ NB additions
$(N^2/2 - N/2) \times$ NB multiplications
$N \times$ NB divisions

See also: GECE, GEDC, GEFS, GELE, GELU, GESS

Author: Linda Kaufman

Example: Usually the subroutine GEBS is used as part of a package designed for general matrices as in the example for GECE. However it may also be used to solve a linear system with a triangular matrix as in the following example. In this example the last column of the inverse of the triangular matrix

$$\begin{matrix} 1 & -1 & -1 & . & . & . & -1 \\ & 1 & -1 & . & . & . & -1 \\ & & 1 & . & . & . & -1 \\ & & & . & . & . & . \\ & & & & . & . & . \\ & & & & & 1 & -1 \\ & & & & & & 1 \end{matrix}$$

is computed by setting the right-hand side to the vector (0, 0, . . . , 0, 1).

Although the determinant of the matrix is 1, the inverse can have large off-diagonal elements. In fact as the size of this matrix increases, the off-diagonal elements of the inverse increase and the matrix becomes more ill-conditioned.

```
      INTEGER N, I, J, IWRITE, I1MACH
      REAL A(15,15), B(15)
      N=15
C
C FORM THE MATRIX AND SET THE RIGHT-HAND SIDE
C TO THE LAST COLUMN OF THE IDENTITY MATRIX
      DO 20 I=1,N
         DO 10 J=I,N
            A(I,J) = -1.0
10       CONTINUE
         A(I,I) = 1.0
         B(I)   = 0.0
20    CONTINUE
      B(N)=1.0
C FIND THE LAST COLUMN OF THE INVERSE MATRIX
      CALL GEBS(N,A,15,B,N,1)
      IWRITE=I1MACH(2)
      WRITE(IWRITE,21)(I,B(I),I=1,N)
21    FORMAT(3H B(,I3,3H )=,F15.4)
      STOP
      END
```

Execution on the Honeywell 6000 computer at Bell Labs yields the result:

```
B(  1 )=     8192.0000
B(  2 )=     4096.0000
B(  3 )=     2048.0000
B(  4 )=     1024.0000
B(  5 )=      512.0000
B(  6 )=      256.0000
B(  7 )=      128.0000
B(  8 )=       64.0000
B(  9 )=       32.0000
B( 10 )=       16.0000
B( 11 )=        8.0000
B( 12 )=        4.0000
B( 13 )=        2.0000
B( 14 )=        1.0000
B( 15 )=        1.0000
```

GECE — LU decomposition of a general matrix with condition estimation

Purpose: GECE (GEneral matrix Condition Estimation) gives a lower bound for the condition number of a real general matrix A. It also supplies the LU decomposition of the matrix A using partial pivoting and may be used to replace GEDC or GELU in a linear equation package.

Usage: CALL GECE (N, A, IA, INTER, COND)

N	$\rightarrow$	the order of the matrix A
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA $\geq$ N and KA $\geq$ N, containing the N $\times$ N coefficient matrix
	$\leftarrow$	the LU decomposition of A (see **Note 2**)
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
INTER	$\leftarrow$	an integer vector of length N recording row interchanges performed during the decomposition (see **Note 2**)
COND	$\leftarrow$	an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{\text{Bd}P}$, there may be no correct digits.

Note 2: INTER and the LU decomposition returned in A are suitable for input into GEFS and GEBS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. On return from GECE, U occupies the upper triangular portion of A, P can be obtained from INTER (see the introduction to this chapter), and the elements of L appear permuted in the strictly lower triangular portion of A. Since the diagonal elements of L are all 1, they are not stored.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
10 + k*	singular matrix whose rank is at least k

Double-precision version: DGECE with A and COND declared double precision

Complex version: CGECE with A declared complex

Storage: N real (double precision for DGECE, complex for CGECE) locations of scratch storage in the dynamic storage stack

Time: $\frac{N^3}{3} + \frac{9}{2}N^2 + \frac{19}{6}N$ additions

$\frac{N^3}{3} + \frac{5}{2}N^2 + \frac{7}{6}N$ multiplications

$\frac{N^2}{2} + \frac{3}{2}N$ divisions

Method: Gaussian elimination with partial pivoting.
See the reference below for the method used to estimate the condition number.
GECE calls GELU after setting EPS to 0.

See also: GEBS, GEDC, GEFS, GELE, GELU, GESS

Authors: Doris Ryan and Linda Kaufman

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: The example below is an encoding of the iterative refinement algorithm which can be used to obtain a highly accurate solution to a system of linear equations with an ill-conditioned coefficient matrix. If the condition number is not excessively high, the program usually returns a solution that is accurate to the working precision of the machine.

The iterative refinement algorithm is essentially:

(1) Solve $Ax = b$
(2) Set tol = $\varepsilon \sum |x_i|$
where ε is the precision of the machine
(3) Compute in double precision the residual
$r = Ax - b$
(4) Solve $A\,\delta x = r$
(5) Compute norm = $\sum |\delta x_i|$
(6) Set x to $x + \delta x$
(7) If norm $\leq$ tol stop, else return to step 3

In our code, step (1) is accomplished using the three lower-level subroutines GECE, GEFS, and GEBS. The subroutine GECE factors A into LU where L is lower triangular and U is upper triangular. Then GEFS forward solves with L and GEBS back solves with U. Since A is overwritten by GECE and needed in step (3) of the algorithm, a copy of the A matrix is saved. In step (4) the decomposition created earlier in GECE is reused and only GEFS and GEBS are called. Since it is possible that the matrix is so ill-conditioned that the iterative refinement algorithm will diverge, steps (3) through (7) in our code are performed only a finite number of times. This number is chosen to be an upper bound on the number of bits in the mantissa of the floating-point number supported by the machine.

This algorithm is not yet included in PORT because the double-precision version of the program would require the residuals to be computed in extended precision.

```
      INTEGER N, IA, IB, NB, INTER(5), IREAD, I1MACH
      INTEGER I, J, IWRITE, ITER, IEND
      REAL A(5, 5), SAVEA(5, 5), B(5), SAVEB(5), R(5)
      REAL COND, BNORM, R1MACH, ABS, RNORM
      DOUBLE PRECISION DSDOT
C
      N=5
      IA=5
      IB=5
      NB=1
      IREAD=I1MACH(1)
C
      DO 10 I=1,N
   10    READ(IREAD,11) (A(I,J),J=1,N)
   11    FORMAT(1X,5F8.0)
      DO 20 I=1,IB
   20    READ(IREAD,21) B(I)
   21    FORMAT(F8.0)
```

```
C
C SAVE THE MATRIX AND RIGHT-HAND SIDE (WHICH WILL BE OVERWRITTEN)
C
      DO 40 I=1,N
        SAVEB(I)=B(I)
        DO 30 J=1,N
   30      SAVEA(I,J)=A(I,J)
   40 CONTINUE
C
C SOLVE AX = B USING SEPARATE CALLS TO GECE, GEFS, GEBS
C
      CALL GECE(N,A,IA,INTER,COND)
      IWRITE=I1MACH(2)
      IF (COND.GE.1.0/R1MACH(4)) WRITE(IWRITE,41)
   41 FORMAT(49H CONDITION NUMBER HIGH,ACCURATE SOLUTION UNLIKELY)
C
      CALL GEFS(N,A,IA,B,IB,NB,INTER)
C
      CALL GEBS(N,A,IA,B,IB,NB)
      WRITE(IWRITE,42)
   42 FORMAT(44H ESTIMATED CONDITION NUMBER OF THE MATRIX A,)
      WRITE(IWRITE,43) COND
   43 FORMAT(27H  USING ONE CALL TO GECE = ,E15.7)
      BNORM=0.0
      WRITE(IWRITE,44)
   44 FORMAT(/22H THE FIRST SOLUTION X,)
      WRITE(IWRITE,45)
   45 FORMAT(41H (USING CALLS TO GECE, GEFS, AND GEBS) = )
C
C COMPUTE NORM OF SOLUTION
C
      DO 50 I=1,N
         BNORM=BNORM + ABS(B(I))
   50    WRITE(IWRITE,51) B(I)
   51    FORMAT(1X, 5F20.7)
C
C REFINE THE SOLUTION DEPENDING ON THE LENGTH OF THE MANTISSA
C
            IEND=I1MACH(11)*IFIX(R1MACH(5)/ALOG10(2.0) + 1.0)
      DO 90 ITER=1,IEND
C  COMPUTE RESIDUAL R = B - AX, IN DOUBLE PRECISION
C
         WRITE(IWRITE,52)
   52    FORMAT(/27H THE RESIDUAL R = B - AX = )
         DO 70 I=1,IA
               DSDOT=0.0
               DO 60 J=1,N
   60             DSDOT = DSDOT  +  DBLE(SAVEA(I,J))*B(J)
               R(I) = SAVEB(I) - DSDOT
   70          WRITE(IWRITE,51) R(I)
C
C  SOLVE LU*(DELTA X) = R USING SEPARATE CALLS TO GEFS AND GEBS
C
         CALL GEFS(N,A,IA,R,IB,NB,INTER)
         CALL GEBS(N,A,IA,R,IB,NB)
C
C  THE NEW SOLUTION X = X  +  DELTA X
```

```
C
         WRITE(IWRITE,71)
   71    FORMAT(/36H THE NEW SOLUTION X = X  +  DELTA X = )
C
C DETERMINE NORM OF CORRECTION AND ADD IN CORRECTION
C
         RNORM=0.0
         DO 80 I=1,N
               B(I) = B(I)  +  R(I)
               RNORM=RNORM + ABS(R(I))
   80          WRITE(IWRITE,51) B(I)
C
C TEST FOR CONVERGENCE
C
         IF(RNORM.LT.R1MACH(4)*BNORM) GO TO 100
   90  CONTINUE
       WRITE(IWRITE,91)
   91  FORMAT(/29H ITERATIVE IMPROVEMENT FAILED)
  100  CONTINUE
       STOP
       END
```

For the input matrix

1.	–2.	3.	7.	–9.
–2.	8.	–6.	2.	50.
3.	–6.	18.	–15.	–18.
7.	2.	–15.	273.	174.
–9.	50.	–18.	173.	1667.

with the following right-hand side:

78.
–320.
–81.
215.
–10856.

the following results were obtained on the Honeywell 6000 computer at Bell Labs:

```
ESTIMATED CONDITION NUMBER OF THE MATRIX A,
 USING ONE CALL TO GECE =   0.7263499E 07

THE FIRST SOLUTION X,
(USING CALLS TO GECE, GEFS, AND GEBS) =
```

−5.9872369
−4.9984942
−8.0019742
4.9995200
−6.9999477

```
THE RESIDUAL R = B − AX =
```

0.0000020
−0.0000190
0.0000215
−0.0000073
−0.0000473

```
THE NEW SOLUTION X = X + DELTA X =
```

−6.0000002
−5.0000000
−7.9999999
5.0000000
−7.0000000

```
THE RESIDUAL R = B − AX =
```

0.0000001
−0.0000001
−0.0000004
0.0000026
−0.0000011

```
THE NEW SOLUTION X = X + DELTA X =
```

−6.0000000
−5.0000000
−8.0000000
5.0000000
−7.0000000

The first solution above is inaccurate, as would have been expected from the estimate of the condition number for the matrix. The iterative refinement algorithm successfully improved the solution to this problem because the matrix and the right-hand side could be represented exactly in the machine. (Also the condition number was not high.) Often the input matrix cannot be represented exactly and the iterative refinement algorithm produces a very accurate, but worthless, solution to a slightly incorrect problem.

GEDC — LU decomposition of a general matrix

Purpose: GEDC (GEneral matrix DeComposition) computes the LU decomposition of a dense general matrix using partial pivoting. It is called by GELE as the first step of the solution of a general linear system.

Usage: CALL GEDC (N, A, IA, INTER)

N	$\rightarrow$	the order of the matrix A
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA $\geq$ N and KA $\geq$ N, containing the N $\times$ N coefficient matrix
	$\leftarrow$	the LU decomposition of A (see **Note**)
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
INTER	$\leftarrow$	an integer vector of length N recording row interchanges performed during the decomposition (see **Note**)

Note: INTER and the LU decomposition returned in A are suitable for input into GEFS and GEBS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. On return from GEDC, U occupies the upper triangular portion of A, P can be obtained from INTER (see the introduction to this chapter), and the elements of L appear permuted in the strictly lower triangular portion of A. Since the diagonal elements of L are all 1, they are not stored.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
10 + k*	singular matrix whose rank is at least k

Double-precision version: DGEDC with A declared double precision

Complex version: CGEDC with A declared complex

Storage: None

Time: $\frac{N^3}{3} + \frac{N^2}{2} + \frac{N}{6}$ additions

$\frac{N^3}{3} - \frac{N^2}{2} + \frac{N}{6}$ multiplications

$\frac{(N^2 - N)}{2}$ divisions

Method: Gaussian elimination with partial pivoting. GEDC calls GELU after setting EPS = $||A||\varepsilon$, where ε is machine precision, i.e. the value returned by R1MACH(4) (or, for double precision, by D1MACH(4)).

See also: GEBS, GECE, GEFS, GELE, GELU, GESS

Author: Linda Kaufman

Example: In this example, we illustrate, for a special case, how the building blocks, GEDC, GEFS and GEBS of our linear equation solver can be used to circumvent a limitation in memory space.

We consider a matrix *A* which has the special form

$$A = \begin{bmatrix} C & D \\ E & F \end{bmatrix}$$

where *C* and *F* are dense $n \times n$ matrices, and *D* and *E* are $n \times n$ diagonal matrices. (Thus *D* and *E* can be stored in vectors of length *n*.) If *n* is 200, and the problem is to be solved on a computer with only 100K words of data space, the set of linear equations cannot be solved by either the general subprogram GELE or the band package, BALE, because too much additional storage would be required. The sparse matrix package is also eliminated because those subroutines demand additional storage for additional column indices for each nonzero element.

However, if C^{-1} exists then one can solve $AX = B$ using the fact that

$$\begin{bmatrix} C & D \\ E & F \end{bmatrix} = \begin{bmatrix} I & 0 \\ EC^{-1} & I \end{bmatrix} \begin{bmatrix} C & D \\ 0 & F - EC^{-1}D \end{bmatrix} .$$

If B is partitioned into

$$B = \begin{bmatrix} B_U \\ \hline B_L \end{bmatrix}$$

where B_U has length n, then, applying the inverse,

$$\begin{bmatrix} I & 0 \\ EC^{-1} & I \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -EC^{-1} & I \end{bmatrix} ,$$

to $AX = B$, we see that X is the solution to

$$\begin{bmatrix} C & D \\ 0 & F - E\ C^{-1}D \end{bmatrix} X = \begin{bmatrix} B_U \\ B_L - E\ C^{-1}B_U \end{bmatrix}$$

Finally, if X is partitioned into

$$X = \begin{bmatrix} X_U \\ \hline X_L \end{bmatrix}$$

where X_U has length n, then X can be found by the following algorithm:

(1) replace F by $F - EC^{-1}D$
(2) replace B_L by $B_L - EC^{-1}B_U$
(3) solve $FX_L = B_L$
(4) replace B_U by $B_U - DX_L$
(5) solve $CX_U = B_U$

Of course in steps (1) and (2) we do not form C^{-1} explicitly, but solve a system of equations with C as the coefficient matrix. Thus steps (1),(2), and (5) solve systems with the same coefficient matrix but different right-hand sides. Moreover, the right-hand side for step (5) is not known until after the first two systems have been solved so that GELE, the general linear equation solver, cannot be used to solve all three simultaneously. The correct approach is to compute the LU factorization of C once, and to use it three times. Then subroutine GEFS forward solves with the L portion and GEBS back solves with the U portion.

In the code below, X_U is left in the array B_U and X_L is left in the array B_L.

```
      INTEGER INTER(200), I, J, N
      REAL C(200, 200), D(200), E(200), F(200, 200)
      REAL TEMP(200), BL(200), BU(200)
      N=200
      .
      code for filling in C,D,E,F,BL, and BU belongs here
      .
C DO AN LU DECOMPOSITION OF C
      CALL GEDC(N,C,200,INTER)
C
C FORM F - EC(INVERSE)D IN F
C
      DO 30 J=1,N
         DO 10 I=1,N
            TEMP(I)=0
  10     CONTINUE
         TEMP(J)=D(J)
         CALL GEFS(N,C,200,TEMP,200,1,INTER)
         CALL GEBS(N,C,200,TEMP,200,1)
C TEMP CONTAINS THE JTH COLUMN OF
C C(INVERSE)D
         DO 20 I=1,N
            F(I,J)=F(I,J) - E(I)*TEMP(I)
  20     CONTINUE
  30  CONTINUE
C
C FORM BL - EC(INVERSE)BU
C
      DO 40 I=1,N
         TEMP(I)=BU(I)
  40  CONTINUE
      CALL GEFS(N,C,200,TEMP,200,1,INTER)
      CALL GEBS(N,C,200,TEMP,200,1)
      DO 50 I=1,N
         BL(I)=BL(I) - E(I)*TEMP(I)
  50  CONTINUE
C
C SOLVE FOR LOWER PART OF X
C
      CALL GELE(N,F,200,BL,200,1)
C
C FORM RIGHT HAND SIDE TO SOLVE FOR UPPER PART OF X
C
      DO 60 I=1,N
         BU(I)=BU(I) - D(I)*BL(I)
  60  CONTINUE
C
C SOLVE FOR UPPER PART OF X
C
      CALL GEFS(N,C,200,BU,200,1,INTER)
      CALL GEBS(N,C,200,BU,200,1)
      .
      .
```

GEFS — lower (unit) triangular linear system solution

Purpose: GEFS (GEneral matrix Forward-Solve) solves AX = PB where A is a unit lower triangular matrix, (i.e. 1's on the diagonal), and P is a permutation matrix. It can be used for the forward solution phase of a general linear system solver. (It is used in this way by the routines GESS and GELE.)

Usage: CALL GEFS (N, A, IA, B, IB, NB, INTER)

N	$\rightarrow$	the number of equations
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA $\geq$ N and KA $\geq$ N, containing the N $\times$ N coefficient matrix The upper triangular portion of A(including the main diagonal) is not used or changed.
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
B	$\rightarrow$	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB $\geq$ N and KB $\geq$ NB
	$\leftarrow$	the solution X
IB	$\rightarrow$	the row (leading) dimension of B, as dimensioned in the calling program
NB	$\rightarrow$	the number of right-hand sides
INTER	$\rightarrow$	the integer vector of length N recording interchanges performed in GELU, GEDC, or GECE. To solve a unit lower triangular system, set INTER(J) = J, J = 1,...,N.

Note 1: GEFS and GEBS can be used directly on the output matrix produced by GEDC, GELU, or GECE to solve a general linear system.

Note 2: Users who have to solve a sequence of problems with the same coefficient matrix, but different right-hand sides, *not all known in advance*, should not call GESS or GELE repeatedly, but should use the sequence shown in the example on page 3.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
3	IB < N
4	NB < 1
5	elements of INTER out of range

Double-precision version: DGEFS with A and B declared double precision

Complex version: CGEFS with A and B declared complex

Storage: None

Time: $(N^2 - N) \times NB/2$ additions
$(N^2 - N) \times NB/2$ multiplications

See also: GEBS, GECE, GEDC, GELE, GELU, GESS

Author: Linda Kaufman

Example: In this example we give a general outline of a program to solve a sequence of problems with the same coefficient matrix but different right-hand sides.

The call to GEDC computes the LU decomposition of the matrix A. This decomposition can be then used repeatedly (and efficiently) for the sequence of forward solutions (using GEFS) and back solutions (using GEBS) for each set of right-hand sides.

```
      declare matrix with leading dimension IA
      declare right-hand side vector
      declare integer vector INTER
      assign appropriate values to N and IA
      .
      compute or read in the coefficient matrix
      .
      CALL GEDC(N,A,IA,INTER)

10    compute a right-hand side

      CALL GEFS(N,A,IA,B,N,1,INTER)
      CALL GEBS(N,A,IA,B,N,1)

      if sequence of problems is not finished go to 10
      .
      .
      .
```

GELE — general linear system solution

Purpose: GELE (GEneral Linear Equation solution) solves the system AX = B where A is a dense general matrix.

Usage: CALL GELE (N, A, IA, B, IB, NB)

N	$\rightarrow$	the number of equations
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA $\geq$ N and KA $\geq$ N, containing the N $\times$ N coefficient matrix A is overwritten during the solution.
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
B	$\rightarrow$	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB $\geq$ N and KB $\geq$ NB
	$\leftarrow$	the solution X
IB	$\rightarrow$	the row (leading) dimension of B, as dimensioned in the calling program
NB	$\rightarrow$	the number of right-hand sides

Note 1: Unless the given matrix, A, is known in advance to be well-conditioned, the user should use GESS instead of GELE.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should not use GELE, but should call subprograms GEDC, GEFS and GEBS. (See the example in GEDC.) GEDC is called once to get the LU decomposition (see the introduction to this chapter) and then the pair, GEFS (forward solve) and GEBS (back solve), is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
3	IB < N
4	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DGELE with A and B declared double precision

Complex version: CGELE with A and B declared complex

Storage: N integer locations of scratch storage in the dynamic storage stack

Time: $N^3/3 + N^2/2 + N/6 + NB\times(N^2-N)$ additions
$N^3/3-N^2/2 + N/6 + NB\times(N^2-N)$ multiplications
$(N^2-N)/2 + N\times NB$ divisions

Method: Gaussian elimination with partial pivoting.
GELE calls GEDC, GEFS, and GEBS.

See also: GEBS, GECE, GEDC, GEFS, GELU, GESS

Authors: Linda Kaufman and Doris Ryan

Example: Two things are illustrated in the following example. First, the relative efficiencies of GELE and GESS are compared as a function of the size of the system. The example indicates that the extra cost associated with computing the condition number (GESS) decreases as the size of the system increases. When N = 90, GELE is only about 10% faster that GESS. For small systems with one right-hand side, however, GESS takes almost twice the time required by GELE.

Secondly, the example illustrates the increase of the error in the solution as the condition number grows. Notice that as the condition number increases from 1.3×10^3 to 1×10^6, the error in the solution grows from 3×10^{-7} to 6×10^{-4}.

The N × N matrix used in the example

$$a_{ij} = \begin{cases} j - i & \text{for } i < j \\ i - j + 1 & \text{for } i \geq j \end{cases}$$

becomes more ill-conditioned as N increases. The right-hand side was chosen to make the solution vector all 1's, and the maximum error is computed as $\max_I |X(I) - 1.0|$ for the solution, X.

The timing subroutine, ILAPSZ, on the Honeywell 6000 system has about 1% accuracy.

```
      INTEGER IA, IB, I1MACH, N, I, J, IT, ILAPSZ, IWRITE
      REAL A(100, 100), AA(100, 100), B(100), BB(100)
      REAL SUM, ERR, COND, ABS, TIME, TIMES, AMAX1
      IA=100
      IB =100
C
C GENERATE THE MATRIX AND RIGHT-HAND SIDE
C
      DO 40 N=10,90,40
         DO 20 I=1,N
            SUM=0.0
            DO 10 J=1,N
               A(I,J)=ABS(I-J)
               IF (I.GE.J) A(I,J)=A(I,J) + 1.0
               AA(I,J)=A(I,J)
               SUM=SUM + AA(I,J)
10          CONTINUE
            B(I)=SUM
            BB(I)=SUM
20       CONTINUE
C
C CALL  GELE AND TIME IT
         IT =ILAPSZ(0)
         CALL GELE(N,A,IA,B,IB,1)
         TIME=FLOAT(ILAPSZ(0)-IT)/64.0
C
C COMPUTE THE MAXIMUM ERROR
C
         ERR=0.0
         DO 30 I=1,N
            ERR=AMAX1(ERR, ABS(B(I)-1.0))
30       CONTINUE
C
C CALL GESS
C
         IT =ILAPSZ(0)
```

```
         CALL GESS(N,AA,IA,BB,IB,1,COND)
         TIMES=FLOAT(ILAPSZ(0)-IT)/64.0
         IWRITE=I1MACH(2)
         WRITE(IWRITE,31)N,COND
31       FORMAT(8H FOR N= ,I4,20H CONDITION NUMBER = ,E15.7)
         WRITE(IWRITE,32)ERR
32       FORMAT(30H MAXIMUM ERROR IN SOLUTION IS ,F15.7)
         WRITE(IWRITE,33)TIME
33       FORMAT(34H TIME IN MILLISECONDS FOR GELE IS ,F10.2)
         WRITE(IWRITE,34)TIMES
34       FORMAT(34H TIME IN MILLISECONDS FOR GESS IS ,F10.2)
40    CONTINUE
      STOP
      END
```

When the program above was run on the Honeywell 6000 machine at Bell Laboratories, the following was printed.

```
FOR N=   10 CONDITION NUMBER =   0.1349031E 04
MAXIMUM ERROR IN SOLUTION IS       0.0000003
TIME IN MILLISECONDS FOR GELE IS      13.27
TIME IN MILLISECONDS FOR GESS IS      22.70

FOR N=   50 CONDITION NUMBER =   0.1674143E 06
MAXIMUM ERROR IN SOLUTION IS       0.0000704
TIME IN MILLISECONDS FOR GELE IS     499.30
TIME IN MILLISECONDS FOR GESS IS     597.75

FOR N=   90 CONDITION NUMBER =   0.9745692E 06
MAXIMUM ERROR IN SOLUTION IS       0.0005805
TIME IN MILLISECONDS FOR GELE IS    2602.00
TIME IN MILLISECONDS FOR GESS IS    2919.06
```

GELU — LU decomposition of a general matrix

Purpose: GELU (GEneral matrix LU decomposition) finds the LU decomposition of a dense general matrix using partial pivoting. It allows the user to specify a threshold for considering a matrix singular. GELU is called by the LU decomposition routines GECE and GEDC.

Usage: CALL GELU (N, A, IA, INTER, EPS)

N	$\rightarrow$	the order of the matrix A
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA $\geq$ N and KA $\geq$ N, containing the N $\times$ N coefficient matrix
	$\leftarrow$	the LU decomposition of A (see **Note 2**)
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
INTER	$\leftarrow$	an integer vector of length N recording row interchanges performed during the decomposition (see **Note 2**)
EPS	$\rightarrow$	if A = LU and $\lvert u_{kk} \rvert$ < EPS, for some $1 \leq k \leq N$, the matrix is considered singular.

Note 1: After the execution of GELU, (if the matrix has not been found singular), the value of the determinant is INTER(N) $\times$ A(1,1) $\times$ A(2,2) $\times \cdots \times$ A(N,N) where INTER(N) contains the sign of the permutation (the number of row interchanges).

Note 2: INTER and the LU decomposition returned in A are suitable for input into GEFS and GEBS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. On return from GELU, U occupies the upper triangular portion of A, P can be obtained from INTER (see the introduction to this chapter), and the elements of L appear permuted in the strictly lower triangular portion of A. Since the diagonal elements of L are all 1, they are not stored.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
10 + k*	singular matrix whose rank is at least k

Double-precision version: DGELU with A and EPS declared double precision

Complex version: CGELU with A declared complex

Storage: None

Time: $\frac{N^3}{3} - \frac{N^2}{2} + \frac{N}{6}$ additions

$\frac{N^3}{3} - \frac{N^2}{2} + \frac{N}{6}$ multiplications

$\frac{(N^2 - N)}{2}$ divisions

Method: Gaussian elimination with partial pivoting

See also: GEBS, GECE, GEDC, GEFS, GELE, GESS

Author: Linda Kaufman

Example: The following subroutine uses GELU to compute a determinant. The subroutine uses the stack to obtain space for the integer vector INTER. Care is taken to avoid overflow and underflow during the calculation. The subroutine UMKFL is used to decompose a floating point number, F, into a mantissa, M, and an exponent E such that

$$F = Mb^E$$

where b is the base of the machine and $1/b \leq M < 1$.

```
      SUBROUTINE DET(N,A,IA,DETMAN,IDETEX)
C
C THIS SUBROUTINE COMPUTES THE DETERMINANT OF A
C THE RESULT IS GIVEN BY DETMAN*BETA**IDETEX
C WHERE BETA IS THE BASE OF THE MACHINE
C AND DETMAN IS BETWEEN 1/BETA AND 1
C
      INTEGER N, IA, IDETEX
      INTEGER E, IPOINT, ISTKGT, I1MACH, ISIGN, I
      INTEGER IN(1000)
      REAL A(IA, N), DETMAN, BETA, FLOAT, ONOVBE, M, ABS
      DOUBLE PRECISION D(500)
      COMMON /CSTAK/ D
      EQUIVALENCE(D(1),IN(1))
C
C ALLOCATE SPACE FROM THE STACK FOR THE PIVOT ARRAY
C
      IPOINT=ISTKGT(N,2)
      CALL GELU(N,A,IA,IN(IPOINT),0.0)
C
C THE DETERMINANT IS THE PRODUCT OF THE DIAGONAL ELEMENTS
C AND THE LAST ELEMENT OF THE INTERCHANGE ARRAY
C WE TRY TO COMPUTE THIS PRODUCT IN A WAY THAT WILL
C AVOID UNDERFLOW AND OVERFLOW
C
      BETA=FLOAT(I1MACH(10))
      ONOVBE=1.0/BETA
      ISIGN=IPOINT + N-1
      DETMAN=IN(ISIGN)*ONOVBE
      IDETEX=1
      DO 10 I=1,N
          CALL UMKFL(A(I,I),E,M)
          DETMAN=DETMAN*M
          IDETEX=IDETEX+E
          IF(ABS(DETMAN).GE.ONOVBE) GO TO 10
             IDETEX=IDETEX-1
             DETMAN=DETMAN*BETA
 10   CONTINUE
      RETURN
      END
```

GEML — matrix - vector multiplication

Purpose: GEML (GEneral matrix MuLtiplication) forms the product Ax where A is a general matrix.

Usage: CALL GEML(N, A, IA, X, B)

N	$\rightarrow$	the length of x
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA ≥ N and KA ≥ N, containing the N × N coefficient matrix
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
X	$\rightarrow$	the vector x to be multiplied
B	$\leftarrow$	the vector Ax

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N

Double-precision version: DGEML with A, X, and B declared double precision.

Complex version: CGEML with A, X, and B declared complex

Time: N^2 additions
N^2 multiplications

See also: GECE, GEDC, GELU, GELE, GESS

Author: Linda Kaufman

Example: This example checks the consistency of GEML and GESS, the linear system solver.

First the example uses GEML to compute for a given vector x and matrix A, the vector

$$b = Ax.$$

Then the problem is inverted, i.e., GESS is used to find the vector x which satisfies

$$Ax = b$$

This x is then compared with the original vector. The 10×10 matrix A is chosen so that

$$a_{ij} = \begin{cases} j-i & \text{for } i<j \\ i-j+1 & \text{for } i \geq j \end{cases}$$

The vector x is chosen randomly.

```
      INTEGER I, J, IWRITE, I1MACH, N
      REAL A(10, 10), X(10), B(10)
      REAL ERR, SASUM, UNI, COND
      N=10
C
C CONSTRUCT A MATRIX
C
      DO 20 I=1,N
         DO 10 J=I,N
            A(I,J)=J-I
            A(J,I)=J-I + 1
 10       CONTINUE
 20    CONTINUE
C
C CONSTRUCT A RANDOM VECTOR X
C
      DO 30 I=1,N
         X(I)=UNI(0)
 30    CONTINUE
C
C FIND THE VECTOR B=AX
C
      CALL GEML(N,A,10,X,B)
C
C SOLVE THE SYSTEM AX=B
C
      CALL GESS(N,A,10,B,N,1,COND)
C
C PRINT THE COMPUTED AND TRUE SOLUTION
C
      IWRITE=I1MACH(2)
      WRITE(IWRITE,31)
 31   FORMAT(34H TRUE SOLUTION   COMPUTED SOLUTION)
      WRITE(IWRITE,32)(X(I),B(I),I=1,N)
 32   FORMAT(1H ,2E17.8)
C
```

```
C COMPUTE THE RELATIVE ERROR
C
      ERR=0.0
      DO 40 I=1,N
         ERR=ERR + ABS(B(I)-X(I))
 40   CONTINUE
      ERR=ERR/SASUM(N,X,1)
      WRITE(IWRITE,41)ERR
 41   FORMAT(19H RELATIVE ERROR IS ,1PE15.7)
      WRITE(6,42)COND
 42   FORMAT(21H CONDITION NUMBER IS ,1PE15.7)
      STOP
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
TRUE SOLUTION COMPUTED SOLUTION
0.22925607E 00 0.22925687E 00
0.76687502E 00 0.76687336E 00
0.68317685E 00 0.68317838E 00
0.50919111E 00 0.50918986E 00
0.87455959E 00 0.87456071E 00
0.64464101E 00 0.64463982E 00
0.84746840E 00 0.84746962E 00
0.35396343E 00 0.35396226E 00
0.39889160E 00 0.39889258E 00
0.45709422E 00 0.45709377E 00
RELATIVE ERROR IS 1.9705190E-06
CONDITION NUMBER IS 1.3490306E 03
```

The condition number of the matrix and the precision of the Honeywell computer suggest that even in the absence of roundoff error in GEML, a relative error of 10^{-5} would not be surprising. The value computed above is quite reasonable.

GENM — norm of a general matrix

Purpose: GENM (GEneral matrix NorM) computes the norm of a general matrix A. The norm is defined as $\max_{1 \le j \le n} \sum_{i=1}^{n} | a_{ij} |$

Type: Real function

Usage: <answer> = GENM (N, A, IA)

N	$\rightarrow$	the number of rows in A
A	$\rightarrow$	the array, dimensioned (IA, KA) in the calling program, where IA ≥ N and KA ≥ N, containing the N × N coefficient matrix
	$\leftarrow$	the LU decomposition of A (see **Note 2**)
IA	$\rightarrow$	the row (leading) dimension of A, as dimensioned in the calling program
<answer>	$\leftarrow$	$\max_{1 \le j \le n} \sum_{i=1}^{n} | a_{ij} |$

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N

Double precision version: DGENM with A and DGENM declared double precision

Complex version: CGENM with A declared complex

Storage: None

Time: N^2 additions
N comparisons

See also: GEDC, GELU, GELE, GESS, GECE

Author: Linda Kaufman

Example: The subroutines in the PORT library for solving $Ax = b$ are designed to return computed solutions x such that the residual $r = Ax - b$ satisfies

$$\frac{||r||}{||A|| \; ||x||} \leq \varepsilon$$

where ε is the machine precision. In this example we show that if A is ill-conditioned, then the computed solution need not be very close to the true solution even though equation (1.1) is satisfied. The subroutine GENM is used to compute the left-hand side of (1.1). The matrix in this example is given by

$$a_{ij} = \begin{cases} j-i & \text{for } i<j \\ i-j+1 & \text{for } i \geq j \end{cases}$$

and the true solution is $x_i = i$. The right hand side is generated using GEML and the computed solution is obtained using GELE. The function SAMAX is used to compute the 1-norm of a vector; i.e. $\max_{1 \leq i \leq n} |x_i|$

```
      INTEGER I, J, L, N, IA, IWRITE, I1MACH
      REAL A(50, 50), AA(50, 50), B(50), X(50)
      REAL RELERR, RELRES, XNORM, RNORM, ERR, R(50)
      REAL GENM, SAMAX
      IA = 50
C
C GENERATE MATRIX
C
      N=50
      DO 20 I=1,N
         DO 10 J=I,N
            A(I,J)=J-I
            A(J,I)=J-I + 1
            AA(I,J)=A(I,J)
            AA(J,I)=A(J,I)
 10      CONTINUE
         B(I)=I
 20   CONTINUE
C
C GENERATE RIGHT HAND SIDE
C
```

```
      CALL GEML(N,A,IA,B,X)
C
C MAKE COPY OF RIGHT HAND SIDE
C
      CALL MOVEFR(N,X,B)
C
C SOLVE THE SYSTEM
C
      CALL GELE(N,A,IA,B,N,1)
C
C COMPUTE THE RELATIVE ERROR AND THE RELATIVE RESIDUAL
C
      CALL GEML(N,AA,IA,B,R)
      ERR=0.0
      DO 30 I=1,N
         ERR=AMAX1(ERR,ABS(B(I)-FLOAT(I)))
         R(I)=R(I)-X(I)
 30   CONTINUE
      XNORM=SAMAX(N,X,1)
      RNORM=SAMAX(N,R,1)
      RELERR=ERR/XNORM
      RELRES=RNORM/(XNORM*GENM(N,AA,IA))
      IWRITE=I1MACH(2)
      WRITE(IWRITE,31)RELERR,RELRES
 31   FORMAT(16H RELATIVE ERROR=,E15.5,19H RELATIVE RESIDUAL=,
     1   E15.5)
      STOP
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
 RELATIVE ERROR=    0.13554E-06 RELATIVE RESIDUAL=    0.22987E-10
```

The condition number of the matrix(see the example in GELE) is about 10^5, and the machine precision on the Honeywell computer is about 10^{-8}. Thus even in the absence of roundoff error in GEML, a relative error of 10^{-3} would not be surprising. The relative error given above is quite within reason. The relative residual, as promised, satisfies (1.1) even though the problem is ill-conditioned.

GESS — general linear system solution with condition estimation

Purpose: GESS (GEneral System Solution) solves the system AX = B where A is a general matrix. An estimate of the condition of A is provided.

Usage: CALL GESS (N, A, IA, B, IB, NB, COND)

N	→	the number of equations
A	→	the array, dimensioned (IA, KA) in the calling program, where IA ≥ N and KA ≥ N, containing the N × N coefficient matrix A is overwritten during the solution.
IA	→	the row (leading) dimension of A, as dimensioned in the calling program
B	→	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB ≥ N and KB ≥ NB
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides
COND	←	an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{\mathrm{Bd}P}$, there may be no correct digits.

If the given matrix, A, is known in advance to be well-conditioned, then the user may wish to use the routine GELE, which is a little faster than GESS. Ordinarily, however, the user is strongly urged to choose GESS, and to follow it by a test of the condition estimate.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should not use GESS, but should call subprograms GECE, GEFS and GEBS. (See the example of GEDC.) GECE is called once to get the LU decomposition (see the introduction to this chapter) and then the pair, GEFS (forward solve) and GEBS (back solve), is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IA < N
3	IB < N
4	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DGESS with A, B, and COND declared double precision

Complex version: CGESS with A and B declared complex

Storage: N integer locations and
N real (double precision for DGESS, complex for CGESS) locations of scratch storage in the dynamic storage stack

Time: $\frac{N^3}{3} + N^2 \times (\frac{9}{2} + NB) + N \times (\frac{19}{6} + NB)$ additions
$\frac{N^3}{3} + N^2 \times (\frac{5}{2} + NB) + N \times (\frac{7}{6} + NB)$ multiplications
$\frac{N^2}{2} + N \times (\frac{3}{2} + NB)$ divisions

Method: Gaussian elimination with partial pivoting.
See the reference below for the method used to estimate the condition number.
GESS calls GECE, GEFS, and GEBS.

See also: GEBS, GECE, GEDC, GEFS, GELE, GELU

Authors: Linda Kaufman and Doris Ryan

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: The following program solves a 5×5 system with two right-hand sides

```
      INTEGER N, IREAD, I1MACH, I, NB, IWRITE, J
      REAL A(5,5), B(5,2), COND
      N=5
      IREAD=I1MACH(1)
C
      DO 10 I=1,N
          READ(IREAD,1) (A(I,J),J=1,N)
    1     FORMAT(1X,5F10.0)
   10  CONTINUE
C
      NB=2
      DO 20 I=1,N
          READ(IREAD,11) (B(I,J),J=1,NB)
   11     FORMAT(1X,2F10.3)
   20  CONTINUE
C
C SOLVE AX = B  BY CALLING GESS
C
      CALL GESS(N,A,N,B,N,NB,COND)
      IWRITE=I1MACH(2)
      WRITE(IWRITE,21) COND
   21  FORMAT(52H AN ESTIMATE OF THE CONDITION NUMBER OF THE MATRIX =,
     1         E14.7)
C
      WRITE(IWRITE,22)
   22  FORMAT(27H THE COMPUTED SOLUTION X IS,//)
      DO 30 I=1,N
          WRITE(IWRITE,23) (B(I,J),J=1,NB)
   23     FORMAT(1H,5F20.7)
   30  CONTINUE
C
      STOP
      END
```

For the input matrix given by:

1.	−2.	3.	7.	−9.
−2.	8.	−6.	9.	50.
11.	−6.	18.	−15.	−18.
7.	2.	−15.	273.	173.
−9.	50.	−18.	6.	1667.

and the following right-hand sides:

30.	29.419
−191.	−190.994
133.	133.072
−986.	−985.775
−6496.	−6495.553

the following results were obtained on the Honeywell 6000 computer at Bell Labs:

```
AN ESTIMATE OF THE CONDITION NUMBER OF THE MATRIX = 0.2759414E 04
THE COMPUTED SOLUTION X IS
```

2.0000004	2.4800003
4.9999970	4.8709986
2.9999988	2.6439993
−1.0000001	−1.0320001
−3.9999999	−3.9970000

The true solution to this problem is:

2.	2.48
5.	4.871
3.	2.644
−1.	−1.032
−4.	−3.997

Notice that a seemingly slight change in the right-hand side causes the solution to change noticeably. Furthermore, the relative error in the solution is about 2×10^{-7}. On the Honeywell computer, which has about 8 decimal digits for single-precision numbers, this represents the loss of about 1.5 decimal digits. A loss of up to 2×10^{-5} could be expected in light of the analysis given below.

Let Δb represent a perturbation in the right-hand side of a linear system.
If $Ax = b$ then

$$A(x + \Delta x) = b + \Delta b$$

where

$$\frac{||\Delta x||}{||x||} \leq K(A) \left[\frac{||\Delta b||}{||b||} \right]$$

where $K(A)$ is the condition number of A, $K(A) = ||A||\ ||A^{-1}||$ and $||\cdot||$, is some norm, e.g., $||x||_1 = \sum_{i=1}^{n} |x_i|$ if x is a vector.

The methods used in our linear equation package are guaranteed to provide an accurate answer to a slightly perturbed problem. If we assume that our method produces the correct answer to a problem where $||\Delta b|| \leq \varepsilon||b||$, where ε is the machine precision, then on the Honeywell 6000 where ε is about 10^{-8}, a relative error for the above example of 2×10^{-5} would not be surprising.

In our example one may consider the first column of B as b in (1.1), and the second column of B as $b + \Delta b$, so that $||\Delta b||/||b||$ is approximately .00015 using the $||\cdot||_1$ norm. If we look at the second solution as $x + \Delta x$ in (1.1) and the first solution as x, then $||\Delta x||/||x||$ is approximately .07. Thus equation (1.1) indicates that the condition number is at least 400, and the condition estimate, which at first appeared to be conservative, was in fact quite realistic.

SYMMETRIC MATRICES

SYCE - Condition Estimation
SYDC - DeComposition
SYFBS - Forward and Back Solve
SYLE - Linear Equation solution
SYMD - MDM^T decomposition
SYML - MuLtiplication
SYNM - NorM
SYSS - System Solution

Purpose: SYCE (SYmmetric matrix Condition Estimation) gives a lower bound for the condition number of a symmetric matrix A, which need not be positive definite. It also supplies the MDM^T decomposition of A and may be used in a linear equation package.

Usage: CALL SYCE (N, C, INTER, COND)

N $\longrightarrow$ the number of rows in A

C $\longrightarrow$ a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \longrightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

$\longleftarrow$ the D and M matrices of the decomposition for SYFBS (see **Note 2**)

INTER $\longleftarrow$ an integer vector of length N containing a record of the interchanges, i.e. the matrix P, described in **Note 2** below.

COND $\longleftarrow$ an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as **d** – $\log_{10}$(COND) correct decimal digits. Thus if COND is greater than $10^{\mathrm{Bd}P}$, there may be no correct digits.

Note 2: The MDM^T decomposition of a symmetric matrix A satisfies $P^TAP = MDM^T$, where P is a permutation matrix, M is a unit lower triangular matrix, and D is a block diagonal matrix with blocks of order 1×1 and blocks of order 2×2. Whenever $d_{i+1,i}$ is nonzero (in a 2×2 block of D), $m_{i+1,i}$ is zero. On return from SYCE, d_{ii}, the diagonal of D, occupies the position of C which contained a_{ii} on entry, and the elements of the strictly lower portions of M and D appear permuted in the remaining positions of C. Since the diagonal elements of M are all 1, they are not stored. The positive elements of INTER contain information for constructing P (see the introduction to this chapter). The negative elements of INTER, if any, indicate the presence of 2 × 2 blocks in D. If INTER(I) is negative, then D contains a 2×2 block beginning at row I–1. In this case, $d_{i,i-1}$ directly follows $d_{i-1,i-1}$ in C.

Note 3: For complex Hermitian matrices (A = A*, where A* is the conjugate transpose of A), the complex Hermitian version of this subroutine computes the MDM* decomposition and returns the conjugate of M rather than M in C.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYCE with C declared double precision.

Complex symmetric version: CSYCE with C declared complex

Complex Hermitian version: CHECE with C declared complex (see **Note 3**).

Storage: N real (double precision for DSYCE, complex for CSYCE) locations of scratch storage in the dynamic storage stack

Time: $\frac{N^3}{6} + \frac{19}{4}N^2 + \frac{25}{6}N$ additions

$\frac{N^3}{6} + \frac{13}{4}N^2 + \frac{5}{6}N$ multiplications

$\frac{N^2}{2} + \frac{5}{2}N$ divisions

at most $N^2 + N$ comparisons

Method: The Bunch - Kaufman algorithm, described in reference [1] below, is used to determine the decomposition. The algorithm in [2] is used to get the condition estimate.

SYCE calls SYMD after setting EPS to 0.0

See also: SYLE, SYFBS, SYMD, SYDC, SYSS

Author: Linda Kaufman

References: [1] Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

[2] Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: This example is an encoding of the iterative refinement algorithm which may be used to obtain a highly accurate solution to a system of linear equations with an ill-conditioned coefficient matrix. If the condition number is not excessively high, the program usually returns a solution that is accurate to the working precision of the machine.

The iterative refinement algorithm is essentially:

(1) Solve $Ax = b$
(2) Set $tol = \varepsilon \sum |x_i|$
where ε is the precision of the machine
(3) Compute in double precision the residual
$r = Ax - b$
(4) Solve $A\,\delta x = r$
(5) Compute $norm = \sum |\delta x_i|$
(6) Set x to $x + \delta x$
(7) If $norm \leq tol$ stop, else return to step 3

In the program below, step (1) is accomplished using the two lower-level subroutines SYCE and SYFBS. SYCE decomposes A into several factors and SYFBS solves the system using these factors. Since A is destroyed by SYCE and needed in step (3) of the algorithm, a copy of the A matrix is saved. In step (4) the decomposition created earlier in SYCE is reused and only the forward and back solver SYFBS is required. Since it is possible that the matrix is so ill-conditioned that the iterative refinement algorithm will diverge, steps (3) through (7) in the program are performed only a finite number of times. This number is chosen to be an upper bound on the number of bits in the mantissa of the floating-point number supported by the machine.

This algorithm is not included in PORT because for double-precision matrices part of the computation would have to be done in extended precision.

```
      INTEGER N, JEND, IREAD, I1MACH, I, JBEGIN, J, IWRITE
      INTEGER INTER(6), IEND, ITER, L, IFIX
      REAL C(20), SAVEC(36), B(6), SAVEB(6), R(6)
      REAL COND, R1MACH, BNORM, RNORM, ABS, ALOG10
      DOUBLE PRECISION D(6)
      N=5
C
C READ IN A SYMMETRIC MATRIX WHOSE UPPER TRIANGULAR
C PORTION IS STORED ONE ROW PER CARD. MAKE A
C COPY OF THE MATRIX SO THAT IT CAN BE USED LATER
C
       JEND=0
       IREAD=I1MACH(1)
       DO 20 I=1,N
          JBEGIN=JEND+1
          JEND=JBEGIN+N - I
          READ(IREAD,1)(C(J),J=JBEGIN,JEND)
 1        FORMAT(5F8.0)
          DO 10 J=JBEGIN,JEND
               SAVEC(J)=C(J)
10        CONTINUE
20     CONTINUE
C READ IN RIGHT HAND SIDE AND SAVE IT
       DO 30 I=1,N
          READ(IREAD,1)B(I)
          SAVEB(I)=B(I)
30     CONTINUE
C
C  SOLVE AX = B USING SEPARATE CALLS TO SYCE AND SYFBS
C
      CALL SYCE(N,C,INTER,COND)
      CALL SYFBS(N,C,B,6,1,INTER)
      IWRITE=I1MACH(2)
      IF(COND.GE.1.0/R1MACH(4))WRITE(IWRITE,31)
31    FORMAT(49H CONDITION NUMBER HIGH,ACCURATE SOLUTION UNLIKELY)
      WRITE(IWRITE,32) COND
32    FORMAT(21H CONDITION NUMBER IS ,1PE16.8)
C     COMPUTE NORM OF SOLUTION
      BNORM=0.0
      WRITE(IWRITE,33)
33    FORMAT(43H THE FIRST SOLUTION X, FROM SYCE AND SYFBS=)
      DO 40 I=1,N
         BNORM=BNORM+ABS(B(I))
40       WRITE(IWRITE,41)B(I)
41    FORMAT(1H ,F20.7)
C
C IEND IS THE UPPER BOUND ON THE NUMBER OF BITS PER WORD
C
      IEND=I1MACH(11)*IFIX(R1MACH(5)/ALOG10(2.0)+1.0)
C
C REFINE SOLUTION
C
      DO 90 ITER=1,IEND
C
C COMPUTE RESIDUAL R = B - AX, IN DOUBLE PRECISION
C
         DO 50 I=1,N
```

```
  50         D(I)=DBLE(SAVEB(I))
             L=1
             DO 70 I=1,N
                DO 60 J=I,N
                   IF (I.NE.J) D(J)=D(J) - DBLE(SAVEC(L))*B(I)
                   D(I) = D(I) - DBLE(SAVEC(L))*B(J)
  60            L=L+1
                R(I) = D(I)
  70         CONTINUE
C
C SOLVE A(DELTAX) =R
C
           CALL SYFBS(N,C,R,8,1,INTER)
C
C DETERMINE NORM OF CORRECTION AND ADD IN CORRECTION
C
           WRITE(IWRITE,71)ITER
  71       FORMAT(30H THE SOLUTION AFTER ITERATION ,I5)
           RNORM=0.0
           DO 80 I=1,N
                 B(I) = B(I) + R(I)
                 RNORM=RNORM+ABS(R(I))
                 WRITE(IWRITE,41)B(I)
  80       CONTINUE
        IF(RNORM.LT.R1MACH(4)*BNORM) STOP
  90    CONTINUE
        WRITE(IWRITE,91)
  91    FORMAT(29H ITERATIVE IMPROVEMENT FAILED)
        STOP
        END
```

The above program was applied to a problem in which the upper triangular portion of the symmetric matrix A was given by

–4.0	0.0	–16.	–32.	28.0
	1.0	5.0	10.0	–6.0
		–37.0	–66.0	64.0
			–85.0	53.0
				–15.0

Of course when the matrix was read in to the C array, to conform to FORTRAN conventions each of the above lines had to be left justified.

When the following right hand side was read in

448.
–111.
1029.
1207.
–719.

the following results were obtained on the Honeywell 6000 computer at Bell Labs:

```
CONDITION NUMBER IS   6.71523875E 05
THE FIRST SOLUTION X, FROM SYCE AND SYFBS=
         -8.0002890
         -3.0003689
         -1.9998967
         -5.0000131
          8.0000029
THE SOLUTION AFTER ITERATION     1
         -8.0000000
         -3.0000000
         -2.0000000
         -5.0000000
          8.0000000
THE SOLUTION AFTER ITERATION     2
         -8.0000000
         -3.0000000
         -2.0000000
         -5.0000000
          8.0000000
```

As in most iterative algorithms, the algorithm implemented above stops when the change in the solution is sufficiently small. Although the solution at the end of iteration 1 is correct, the change from the original solution was large and hence the program decided to take one more step.

The first solution above is inaccurate, as would have been expected from the estimate of the condition number for the matrix. The iterative refinement algorithm successfully improved the solution to this problem because the matrix and the right-hand side could be exactly represented in the machine. (Also the condition number was not high.) Often the input matrix cannot be represented exactly and the iterative refinement algorithm produces a very accurate, but worthless, solution to a slightly incorrect problem.

SYDC — decomposition of a symmetric matrix

Purpose: SYDC (SYmmetric matrix DeComposition) forms the MDM^T decomposition of a symmetric matrix A, which need not be positive definite. It is called by SYLE as the first step of the solution of a symmetric linear system.

Usage: CALL SYDC (N, C, INTER)

N $\longrightarrow$ the order of the matrix A

C $\longrightarrow$ a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \longrightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

$\longleftarrow$ the D and M matrices of the decomposition for SYFBS (see **Note 1**)

INTER $\longleftarrow$ an integer vector of length N containing a record of the interchanges, i.e. the matrix P, described in **Note 1** below.

Note 1: The MDM^T decomposition of a symmetric matrix A satisfies $P^TAP = MDM^T$ where P is a permutation matrix, M is a unit lower triangular matrix, and D is a block diagonal matrix with blocks of order 1×1 and blocks of order 2×2. Whenever $d_{i+1,i}$ is nonzero (in a 2×2 block of D), $m_{i+1,i}$ is zero. On return from SYDC, d_{ii}, the diagonal of D, occupies the position of C which contained a_{ii} on entry, and the elements of the strictly lower portions of M and D appear permuted in the remaining positions of C. Since the diagonal elements of M are all 1, they are not stored. The positive elements of INTER contain information for constructing P (see the introduction to this chapter). The negative elements of INTER, if any, indicate the presence of 2×2 blocks in D. If INTER(I) is negative, D contains a 2×2 block beginning at row I–1. In this case, $d_{i,i-1}$ directly follows $d_{i-1,i-1}$ in C.

Note 2: For complex Hermitian matrices (A = A*, where A* is the conjugate transpose of A), the complex Hermitian version of this subroutine computes the MDM* decomposition and returns the conjugate of M rather than M in C.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYDC with C declared double precision.

Complex symmetric version: CSYDC with C declared complex

Complex Hermitian version: CHEDC with C declared complex (see **Note 2** above).

Storage: None

Time: $\frac{N^3}{6} + \frac{3}{4}N^2 + \frac{7}{6}N$ additions
$\frac{N^3}{6} + \frac{N^2}{4} + \frac{11}{6}N$ multiplications
$\frac{N^2}{2} + \frac{N}{2}$ divisions

at most $N^2 - 1$ comparisons

Method: The Bunch - Kaufman algorithm, described in reference [1] below, is used.

SYDC calls SYMD after setting EPS = $||A|| \varepsilon$, where ε is the machine precision, i.e. the value returned by R1MACH(4) (or, for double precision, by D1MACH(4)).

See also: SYLE, SYFBS, SYMD, SYCE, SYSS

Author: Linda Kaufman

Reference: Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

Example: The program fragment below determines how many positive eigenvalues a symmetric matrix, A, has.

According to the reference above the signs of the eigenvalues of A correspond to the signs of the eigenvalues of D in the MDM^T decomposition of A. Moreover, each 2×2 block in D corresponds to a positive-negative eigenvalue pair. Thus the number of positive eigenvalues of D is equal to the number of 2×2 blocks of D plus the number of positive 1×1 blocks.

The program first counts the number of 2×2 blocks of D by counting the number of negative elements in the array INTER, since each negative element (see **Note 1** above) signals the existence of a 2×2 block. It adds to this count the number of positive 1×1 blocks of D in order to find the total number of positive eigenvalues of A.

```
        CALL SYDC(N,C,INTER)
        NPOS=0
C
C COUNT THE NUMBER OF POSITIVE EIGENVALUES OF D AND HENCE
C OF THE MATRIX WHICH HAD ORIGINALLY BEEN PACKED INTO C
C
C THE INDEX K PICKS OUT THE DIAGONAL OF THE MATRIX D OF
C THE DECOMPOSITION
C
        K=1
        I=1
  10    IF (I-N) 20,30,40
  20    (INTER(I+1) .GT. 0) GO TO 30
C
C OTHERWISE WE HAVE A 2×2 BLOCK
C
          NPOS=NPOS+1
          K=K+2*(N-I)+1
          I=I+2
          GO TO 10
C
C WE HAVE A 1×1 BLOCK
C
  30      IF(C(K) .GT. 0.0) NPOS=NPOS+1
          K = K + N - I
          I=I+1
          GO TO 10
  40    IWRITE=I1MACH(2)
        WRITE(IWRITE,41)NPOS
  41    FORMAT(38H THE NUMBER OF POSITIVE EIGENVALUES IS,I5)
```

SYFBS — forward and back solve for symmetric matrices

Purpose: SYFBS (SYmmetric matrix Forward and Back Solution) solves AX = B where A is a symmetric matrix using the decomposition of A computed by SYCE, SYDC, or SYMD. It is called by both SYSS and SYLE to solve symmetric linear systems.

Usage: CALL SYFBS (N, C, B, IB, NB, INTER)

N	→	the number of equations
C	→	a one-dimensional array of length N(N+1)/2 containing the matrices M and D computed by SYDC, SYCE, or SYMD
B	→	the matrix of right-hand sides, dimensioned (IB,KB) in the calling program, where IB≥N and KB≥NB
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides
INTER	→	an integer vector of length N containing a record of the interchanges performed by SYDC, SYCE, or SYMD

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IB < N
3	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYFBS with C and B declared double precision.

Complex version: CSYFBS with C and B declared complex

Complex Hermitian version: CHEFBS with C and B declared complex

Storage: None

Time: $NB\times(N^2-N)$ additions
$NB\times(N^2+N)$ multiplications
$NB\times N$ divisions

Method: The Bunch - Kaufman algorithm, described in the reference below, is used.

See also: SYLE, SYDC, SYMD, SYSS, SYCE

Author: Linda Kaufman

Reference: Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

Example: The program fragment below replaces the K×N matrix B with BA^{-1} where A is a symmetric matrix packed into C according to the scheme given in the parameter list of SYDC. Note that A^{-1} is not formed explicitly since forming BA^{-1} is equivalent to solving XA = B for the matrix X. Because A is symmetric, solving XA = B is, in turn, equivalent to solving $AX^T = B^T$. Thus the problem reduces to solving a linear system with 'K' right-hand sides, each of which unfortunately resides in a 'row' of the array B, rather than a column of an array. In the program fragment we chose not to transpose the matrix B but to invoke SYFBS K times and store each row of B temporarily in the one-dimensional array Y.

```
      CALL SYDC(N,C,INTER)
      DO 30 I=1,K
         DO 10 J=1,N
            Y(J)=B(I,J)
10       CONTINUE
         CALL SYFBS(N,C,Y,N,1,INTER)
         DO 20 J=1,N
            B(I,J)=Y(J)
20       CONTINUE
30    CONTINUE
```

SYLE — symmetric linear system solution

Purpose: SYLE (SYmmetric Linear Equation solution) solves AX = B where A is a symmetric matrix. A does not have to be positive definite.

Usage: CALL SYLE (N, C, B, IB, NB)

N → the number of equations

C → a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \rightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

C is overwritten during the solution.

B → the matrix of right-hand sides, dimensioned (IB,KB) in the calling program, where IB≥N and KB≥NB

← the solution X

IB → the row (leading) dimension of B, as dimensioned in the calling program

NB → the number of right-hand sides

Note 1: Unless the given matrix A, is known in advance to be well-conditioned, the user should use the routine SYSS instead of SYLE.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should not use SYLE, but should call subprograms SYDC and SYFBS. (See the example in SYCE.) SYDC is called once to get the MDM^T decomposition (see the introduction to this chapter) and then SYFBS is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IB < N
3	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYLE with C and B declared double precision.

Complex symmetric version: CSYLE with C and B declared complex

Complex Hermitian version: CHELE with C and B declared complex

Storage: N integer locations of scratch storage in the dynamic storage stack

Time: $\frac{N^3}{6} + (\frac{3}{4}+NB)\times N^2 + (\frac{7}{6}+NB)$ N additions

$\frac{N^3}{6} + (\frac{1}{4}+NB)\times N^2 + (\frac{11}{6}+NB)\times$N multiplications

$\frac{N^2}{2} + (\frac{1}{2}+NB)\times$N divisions

at most $N^2 - 1$ comparisons

Method: The Bunch - Kaufman algorithm, described in the reference below, is used.

See also: SYDC, SYFBS, SYMD, SYSS, SYCE

Author: Linda Kaufman

Reference: Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

Examples:

1. The following call to SYLE replaces the $N \times K$ matrix B with A^{-1} B where A is a symmetric matrix packed into a vector C. Note that A^{-1} is not computed

```
CALL SYLE (N, C, B, N, K)
```

2. On the other hand, to compute the inverse A^{-1}, one may set B to the identity matrix and call SYLE. The following program fragment leaves A^{-1} in the matrix B. It does not use the fact that A^{-1} is symmetric.

```
      DO 20 I=1,N
         DO 10 J=1,N
            B(I,J)=0.0
10       CONTINUE
         B(I,I)=1.0
20    CONTINUE
      CALL SYLE(N, C, B, N, N)
```

SYMD — MDM^T decomposition of a symmetric matrix

Purpose: SYMD (SYmmetric MDM^T decomposition) forms the decomposition $PMDM^TP^T$ of a symmetric matrix A, where P is a permutation matrix, M is unit lower triangular matrix, and D is block diagonal. The matrix A need not be positive definite. This subroutine allows the user to specify a threshold for considering the matrix singular. It is called by the decomposition routines SYDC and SYCE.

Usage: CALL SYMD (N, C, INTER, EPS)

N $\longrightarrow$ the order of the matrix A

C $\longrightarrow$ a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \longrightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

$\longleftarrow$ the D and M matrices of the decomposition for SYFBS (see **Note 1**)

INTER $\longleftarrow$ a vector of length N containing a record of the interchanges, i.e. the matrix P, described in **Note 1** below.

EPS $\longrightarrow$ if $| d_{kk} | \leq$ EPS and d_{kk} corresponds to a 1×1 block of D, then C is considered a singular matrix whose rank is at least k–1

Note 1: The MDM^T decomposition of a symmetric matrix A satisfies $P^TAP = MDM^T$ where P is a permutation matrix, M is a unit lower triangular matrix, and D is a block diagonal matrix with blocks of order 1×1 and blocks of order 2×2. Whenever $d_{i+1,i}$ is nonzero (in a 2×2 block of D), $m_{i+1,i}$ is zero. On return from SYMD, d_{ii}, the diagonal of D, occupies the position of C which contained a_{ii} on entry, and the elements of the strictly lower portions of M and D appear permuted in the remaining positions of C. Since the diagonal elements of M are all 1, they are not stored. The positive elements of INTER contain information for constructing P (see the introduction to this chapter). The negative elements of INTER, if any, indicate the presence of 2×2 blocks in D. If INTER(I) is negative, D contains a 2×2 block beginning at row I–1. In this case, $d_{i,i-1}$ directly follows $d_{i-1,i-1}$ in C.

Note 2: For complex Hermitian matrices(A = A*, where A* is the conjugate transpose of A), the complex Hermitian version of this subroutine computes the MDM* decomposition and returns the conjugate of M rather than M in C.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYMD with C and EPS declared double precision.

Complex symmetric version: CSYMD with C declared complex

Complex Hermitian version: CHEMD with C declared complex (see **Note 2**).

Storage: None

Time: $\frac{N^3}{6} + \frac{N^2}{4} + \frac{7}{6}N$ additions
$\frac{N^3}{6} + \frac{N^2}{4} + \frac{11}{6}N$ additions
$\frac{N^2}{2} + \frac{N}{2}$ divisions
at most $N^2 - 1$ comparisons

Method: The Bunch - Kaufman algorithm, described in the reference below, is used.

See also: SYLE, SYDC, SYFBS, SYSS, SYCE

Author: Linda Kaufman

Reference: Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

Example: The following program fragment determines whether a matrix is positive definite. According to the theory given in the reference above, a symmetric matrix is positive definite only if D in the decomposition computed by SYMD is diagonal with positive diagonal elements. If D is diagonal, all the elements of INTER are positive and the elements of D are packed into C in the same positions that the diagonal of A had originally occupied.

```
      CALL SYMD(N,C,INTER,0.0)
      IWRITE=I1MACH(2)
C
C DETERMINE IF THE MATRIX PACKED INTO C IS POSITIVE DEFINITE.
C THE INDEX K PICKS OUT THE DIAGONAL OF THE MATRIX D
C OF THE DECOMPOSITION
       K=1
       DO 10 I=1,N
          IF(INTER(I).LT.0.OR.C(K).LE.0.0) GO TO 20
C FIND NEXT DIAGONAL ELEMENT
          K = K + N - I
10     CONTINUE
       WRITE(IWRITE,11)
11     FORMAT(32H THE MATRIX IS POSITIVE DEFINITE)
       GO TO 30
20     WRITE(IWRITE,21)
21     FORMAT(36H THE MATRIX IS NOT POSITIVE DEFINITE)
30     CONTINUE
```

SYML — symmetric matrix - vector multiplication

Purpose: SYML (SYmmetric matrix MuLtiplication) forms the product *Ax* where *A* is a general symmetric matrix stored in packed form.

Usage: CALL SYML(N, C, X, B)

N → the length of *x*

C → a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \rightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

X → the vector *x* to be multiplied

B ← the vector A*x*

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1

Double-precision version: DSYML with C, X, and B declared double precision.

Complex version: CSYML with C, X, and B declared complex

Complex Hermitian version: CHEML with C, X, and B declared complex

Time: N^2 additions
N^2 multiplications

See also: SYFBS, SYCE, SYDC, SYMD, SYLE, SYSS

Author: Linda Kaufman

Example: This example checks the consistency of SYML and SYLE, the symmetric linear equation solver.

First the example uses SYML to compute for a given vector x and matrix A, the vector

$$b = Ax.$$

Then the problem is inverted, i.e., SYLE is used to find the vector x which satisfies

$$Ax = b$$

This x is then compared with the original vector. The 10×10 symmetric matrix A is chosen so that

$$a_{ij} = |i - j|.$$

The vector x is chosen randomly.

```
      INTEGER N, L, I, J, IWRITE, I1MACH
      REAL C(55), X(10), B(10)
      REAL UNI, ERR, SASUM, ABS
      N=10
C
C CONSTRUCT THE MATRIX A(I,J)=ABS(J-I) AND PACK INTO C
C
      L=0
      DO 20 I=1,N
         DO 10 J=I,N
            L=L+1
            C(L)=J-I
 10       CONTINUE
 20    CONTINUE
C
C CONSTRUCT A RANDOM VECTOR X
C
      DO 30 I=1,N
         X(I)=UNI(0)
 30    CONTINUE
C
C FIND THE VECTOR B=AX
C
      CALL SYML(N,C,X,B)
C
C SOLVE THE SYSTEM AX=B
C
```

```
      CALL SYLE(N,C,B,N,1)
C
C PRINT THE COMPUTED AND TRUE SOLUTION
C
      IWRITE=I1MACH(2)
      WRITE(IWRITE,31)
  31  FORMAT(34H TRUE SOLUTION   COMPUTED SOLUTION)
      WRITE(IWRITE,32)(X(I),B(I),I=1,N)
  32  FORMAT(1H ,2E17.8)
C
C COMPUTE THE RELATIVE ERROR
C
      ERR=0.0
      DO 40 I=1,N
         ERR=ERR+ABS(B(I)-X(I))
  40  CONTINUE
      ERR=ERR/SASUM(N,X,1)
      WRITE(IWRITE,41)ERR
  41  FORMAT(19H RELATIVE ERROR IS ,1PE15.7)
      STOP
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
   TRUE SOLUTION         COMPUTED SOLUTION
   0.22925607E 00          0.22925607E 00
   0.76687502E 00          0.76687498E 00
   0.68317685E 00          0.68317696E 00
   0.50919111E 00          0.50919100E 00
   0.87455959E 00          0.87455969E 00
   0.64464101E 00          0.64464090E 00
   0.84746840E 00          0.84746833E 00
   0.35396343E 00          0.35396342E 00
   0.39889160E 00          0.39889174E 00
   0.45709422E 00          0.45709418E 00
RELATIVE ERROR IS   1.2794319E-07
```

The condition number of the matrix (see the example in SYSS) is about 54. and the machine precision, on the Honeywell computer is about 10^{-8}. Thus even in the absence of roundoff error in SYML, a relative error of 5×10^{-7} would not be surprising. The value computed above is quite reasonable.

SYNM — norm of a symmetric matrix

Purpose: SYNM (SYmmetric matrix NorM) computes the norm of a symmetric matrix A stored in packed form. The infinity norm is defined as $\max_{1 \le i \le n} \sum_{j=1}^{n} | a_{ij} |$

Type: Real function

Usage: <answer> = SYNM (N, C)

N $\longrightarrow$ the number of rows in A

C $\longrightarrow$ a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \longrightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

<answer> $\longleftarrow \max_{1 \le i \le n} \sum_{j=1}^{n} | a_{ij} |$

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1

Double precision version: DSYNM with C and DSYNM declared double precision

Complex version: CSYNM with C declared complex

Storage: None

Time: N^2 additions
N comparisons

See also: SYDC, SYMD, SYLE, SYSS, SYCE

Author: Linda Kaufman

Example: The subroutines in the library for solving $Ax = b$ are designed to return computed solutions x such that the residual $r = Ax - b$ satifies

$$\frac{||r||}{||A|| \, ||x||} \leq \varepsilon \qquad (1.1)$$

where ε is the machine precision. In this example we show that if A is ill-condtioned, then the computed solution need not be calose to the true solution even though equation (1.1) is satisfied. The subroutine SYNM is used to compute the left-hand side of (1.1). The matrix in this example is given by

$$a_{ij} = |\, i - j \,|$$

and the true solution is $x_i = i$. The right hand side is generated using SYML and the computed solution is obtained using SYLE. The subroutine SAMAX is used to compute the 1-norm of a vector.i.e. $\max_{1 \leq i \leq n} |\, x_i \,|$

```
      INTEGER I, J, L, N, I1MACH, IWRITE
      REAL C(1300), CC(1300), B(50), X(50)
      REAL RELERR, RELRES, XNORM, RNORM, ERR, R(50)
      REAL SYNM, SAMAX
      L=0
C
C GENERATE MATRIX
C
      N=50
      DO 20 I=1,N
         DO 10 J=I,N
            L=L+1
            C(L)=J-I
            CC(L)=C(L)
 10      CONTINUE
         B(I)=I
 20   CONTINUE
C
C GENERATE RIGHT HAND SIDE
C
      CALL SYML(N,C,B,X)
C
C MAKE COPY OF RIGHT HAND SIDE
```

```
C
        CALL MOVEFR(N,X,B)
C
C SOLVE THE SYSTEM
C
        CALL SYLE(N,C,B,N,1)
C
C COMPUTE THE RELATIVE ERROR AND THE RELATIVE RESIDUAL
C
        CALL SYML(N,CC,B,R)
        ERR=0.0
        DO 30 I=1,N
           ERR=AMAX1(ERR,ABS(B(I)-FLOAT(I)))
           R(I)=R(I)-X(I)
 30     CONTINUE
        XNORM=SAMAX(N,X,1)
        RNORM=SAMAX(N,R,1)
        RELERR=ERR/XNORM
        RELRES=RNORM/(XNORM*SYNM(N,CC))
        IWRITE=I1MACH(2)
        WRITE(IWRITE,31)RELERR,RELRES
 31     FORMAT(16H RELATIVE ERROR=,E15.5,19H RELATIVE RESIDUAL=,
     1   E15.5)
        STOP
        END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
RELATIVE ERROR=  0.10544E-07 RELATIVE RESIDUAL=  0.95702E-11
```

The condition number of the matrix (see the example in SYSS) is about 1300, and the machine precision on the Honeywell computer is about 10^{-8}. Thus even in the absence of roundoff error in SYML, a relative error of 1.3×10^{-5} would not be surprising. The relative error given above is quite within reason. The relative residual, as promised, satisfies (1.1) even though the problem is slightly ill-conditioned.

SYSS — symmetric linear system solution with condition estimation

Purpose: SYSS (SYmmetric System Solution) solves the system AX = B where A is a symmetric matrix. It also provides an estimate of the condition number of A. A does not have to be positive definite.

Usage: CALL SYSS (N, C, B, IB, NB, COND)

N $\longrightarrow$ the number of equations

C $\longrightarrow$ a one-dimensional array of length N(N+1)/2 into which the lower triangular part of the matrix A is packed by columns as illustrated in the following 4×4 example:

$$\begin{matrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{matrix} \longrightarrow \begin{matrix} c_1 & & & \\ c_2 & c_5 & & \\ c_3 & c_6 & c_8 & \\ c_4 & c_7 & c_9 & c_{10} \end{matrix}$$

C is overwritten during the solution.

B $\longrightarrow$ the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB

$\longleftarrow$ the solution X

IB $\longrightarrow$ the row (leading) dimension of B, as dimensioned in the calling program

NB $\longrightarrow$ the number of right-hand sides

COND $\longleftarrow$ an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{\text{Bd}P}$, there may be no correct digits.

If the given matrix, A, is known in advance to be well-conditioned, then the user may wish to use the routine SYLE, which is a little faster than SYSS. Ordinarily, however, the user is strongly urged to choose SYSS, and to follow it by a test of the condition estimate.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should not use SYSS, but should call subprograms SYCE and SYFBS. (See the example of SYCE.) SYCE is called once to get the MDM^T decomposition (see the introduction to this chapter) and then SYFBS is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IB < N
3	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DSYSS with C, B, and COND declared double precision.

Complex symmetric version: CSYSS with C and B declared complex

Complex Hermitian version: CHESS with C and B declared complex

Storage: N integer locations and
N real (double precision for DSYSS, complex for CSYSS and CHESS) locations of scratch storage in the dynamic storage stack

Time:
$$\frac{N^3}{6} + \left(\frac{19}{4} + NB\right) \times N^2 + \left(\frac{25}{6} + NB\right) \times N \text{ additions}$$
$$\frac{N^3}{6} + \left(\frac{13}{4} + NB\right) \times N^2 + \left(\frac{5}{6} + NB\right) \times N \text{ multiplications}$$
$$\frac{N^2}{2} + \left(\frac{5}{2} + NB\right) \times N \text{ divisions}$$

at most $N^2 + N$ comparisons

Method: The Bunch - Kaufman algorithm described in reference [1] below, is used. See reference [2] below for the method used to estimate the condition number.
SYSS calls SYCE and SYFBS.

See also: SYDC, SYFBS, SYMD, SYLE, SYCE

Author: Linda Kaufman

References:

[1] Bunch, J. R., Kaufman, L., and Parlett, B., Decomposition of a symmetric matrix, *Numer. Math 27* (1976), 95-109.

[2] Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: The following program packs the matrix

$$a_{i,j} = |\, i - j \,|$$

into the vector C and sets up the right-hand side so that the solution will be all ones. It then calls SYSS to solve the problem and calculates the error in the solution. This example was included to show that seemingly innocent looking problems may be ill-conditioned and to show the effect of ill-conditioning on a solution. It also demonstrates how to pack a symmetric matrix into a vector.

```
      INTEGER N, L, I, IWRITE, I1MACH
      REAL C(5000), B(100)
      REAL SUM, FLOAT, ABS, ERR, COND
      DO 40  N=10,90,40
C
C CREATE THE MATRIX A(I,J)=ABS(I-J), PACK IT INTO
C THE VECTOR C AND FORM THE RIGHT-HAND SIDE SO THE
C SOLUTION HAS ALL ONES.
C
        L=1
        SUM=(N*(N-1))/2
        DO 20 I=1,N
           DO 10 J=I,N
              C(L)=J-I
              L=L+1
 10        CONTINUE
           B(I)=SUM
           SUM=SUM+FLOAT(I-(N-I))
 20     CONTINUE
C
C SOLVE THE SYSTEM AND GET THE CONDITION NUMBER OF THE MATRIX
        CALL SYSS(N,C,B,100,1,COND)
```

```
C
C COMPUTE THE ERROR IN THE SOLUTION
          ERR=0.0
          DO 30 I=1,N
  30          ERR=ERR+ABS(B(I)-1.0)
          ERR=ERR/FLOAT(N)
          IWRITE=I1MACH(2)
          WRITE(IWRITE,31)N
  31      FORMAT(/8H FOR N= ,I5)
          WRITE(IWRITE,32)COND
  32      FORMAT(23H CONDITION ESTIMATE IS 1PE15.7)
          WRITE(IWRITE,33)ERR
  33      FORMAT(30H RELATIVE ERROR IN SOLUTION IS,1PE15.7)
  40    CONTINUE
        STOP
        END
```

The output from the above program run on the Honeywell 6000 at Bell Labs was

```
FOR N=    10
CONDITION ESTIMATE IS   5.4831256E 01
RELATIVE ERROR IN SOLUTION IS  7.3015689E-08

FOR N=    50
CONDITION ESTIMATE IS   1.3551582E 03
RELATIVE ERROR IN SOLUTION IS  4.9673021E-06

FOR N=    90
CONDITION ESTIMATE IS   4.4305656E 03
RELATIVE ERROR IN SOLUTION IS  1.2567225E-05
```

As the output indicates, for small values of N, the matrix is well-conditioned and the solution is very accurate, but as N increases, the matrix becomes more ill-conditioned and the error in the solution increases. Although the relative error for N = 90 appears large, it is not unreasonable as the following analysis indicates:

Let Δb represent a perturbation in the right-hand side of a linear system.
If $Ax = b$ then

$$A(x+\Delta x) = b+\Delta b$$

where

$$\frac{||\Delta x||}{||x||} \leq K(A)\left[\frac{||\Delta b||}{||b||}\right] \qquad (1.1)$$

where $K(A)$ is the condition number of A, $K(A) = ||A||\ ||A^{-1}||$ and $||\cdot||$ is some norm e.g., $||x||_1 = \sum_{i=1}^{n} |x_i|$ if x is a vector.

The methods used in our linear equation package are guaranteed to provide an accurate answer to a slightly perturbed problem. If we assume that our method produces the correct answer to a problem where $||\Delta b|| \leq \varepsilon||b||$, where ε is the machine precision, then on the Honeywell 6000 where ε is about 10^{-8}, a relative error of 4×10^{-5} for the above problem with N of 90 would not be surprising.

Appendix 3

BANDED MATRICES

BABS - Back Solve
BACE - Condition Estimation
BADC - DeComposition
BAFS - Forward Solve
BALE - Linear Equation solution
BALU - LU decomposition
BAML - MuLtiplication
BANM - NorM
BASS - System Solution

Purpose: BABS (BAnded matrix Back Solution) solves AX = B where A is a banded upper triangular matrix. It can be used for the back solution phase of a banded linear system solution. (It is used in this way by the routines BASS and BALE.)

Usage: CALL BABS (N, G, IG, B, IB, NB, MU)

N → the number of equations

G → a matrix (which may have been created by the routines BACE, BADC, or BALU) into which A has been packed as follows:

$$G(j-i+1, i) = a_{ij}$$

i.e. the diagonal is the first row of G,
(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.

IG → the row (leading) dimension of G, as dimensioned in the calling program

B → the matrix of right-hand sides, dimensioned (IB,KB) in the calling program, where IB≥N and KB≥NB

← the solution X

IB → the row (leading) dimension of B, as dimensioned in the calling program

NB → the number of right-hand sides

MU → the number of nonzero bands in A

Notes: BAFS and BABS can be used directly on the output matrix produced by BADC, BALU, or BACE to solve a general linear system.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	IG < MU
3	IB < N
4	NB < 1
5	MU < 1
10 + k*	singular matrix with 0.0 in the kth position on the diagonal

Double-precision version: DBABS with G and B declared double precision.

Complex version: CBABS with G and B declared complex

Storage: None

Time: NB×N×(MU – 1) additions
NB×N×(MU – 1) multiplications
NB×N divisions

See also: BAFS, BADC, BALU, BACE, BASS, BALE

Author: Linda Kaufman

Reference: Martin, R. S., and Wilkinson, J. H., Solution of Symmetric and Unsymmetric Band Equations and the Calculation of Eigenvectors of Band Matrices, *Numer. Math. 9* (1967) 279-301.

Example: In this example we present a subroutine which implements the inverse iteration for finding the eigenvector x corresponding to a specified real eigenvalue λ of a banded matrix A. The algorithm is essentially

Determine x_0, an initial approximation to the eigenvector
Until convergence
Solve $(A-\lambda I)\, x_{k+1} = \alpha_k\, x_k$

where $1/\alpha_k$ is the element of x_k of maximum modulus.

Given the LU decomposition of $A-\lambda I$, the starting vector x_0 is usually set to U^{-1}e, where e is the vector whose elements are all unity. As the above reference indicates, a suitable stopping criteria is $||A||\ \varepsilon \geq ||(\alpha_{k-1}x_{k-1} - \beta_k x_k)\,\alpha_k||_\infty$

where ε is the machine precision given by R1MACH(4) and $1/\beta_k$ is the element of x_k in the same position as the unit element of $\alpha_{k-1}x_{k-1}$.

In the subroutine EIGVEC below, BABS is referenced twice, once to obtain the initial approximation and once within the iterative loop. If λ is an eigenvalue of A, $A-\lambda I$ is theoretically a singular matrix and hence when BALU is invoked, one would expect the subroutine to terminate with a recoverable error. Thus after calling BALU the error flag is turned off if necessary. If BALU determines that a diagonal element of the U matrix is not greater than EPS in magnitude, that diagonal element is set to EPS and the computation continues. Al-

though the computed vectors x_k will not be necessarily close to the true solutions of the linear systems, after these vectors are scaled, they will approach an eigenvector of the matrix.

```
        SUBROUTINE EIGVEC(N,M,ML,G,IG,EVAL,EVEC,LIMIT)
C
C GIVEN A BANDED MATRIX PACKED INTO G WITH
C N ROWS, M NONZERO DIAGONALS AND ML NONZERO DIAGONALS
C ON AND BELOW THE DIAGONAL AND GIVEN AN EIGENVALUE OF THE
C MATRIX IN EVAL, THIS SUBROUTINE USES INVERSE ITERATION TO
C DETERMINE THE CORRESPONDING EIGENVECTOR AND RETURNS IT
C IN EVEC.
C LIMIT IS A BOUND ON THE NUMBER OF ITERATIONS
C
       INTEGER N, M, ML, IG, LIMIT
       INTEGER I, JAL, ISTKGT, JINTER, JX, MU, IERR, NERROR
       INTEGER LIM, JJ, ISAMAX, JXI, IST(1000)
       REAL G(IG, N), EVEC(N), EVAL
       REAL BANM, SIZE, R1MACH, EPS, SC, BET, D1, SC2, ABS
       REAL R(1000)
       DOUBLE PRECISION D(500)
       COMMON /CSTAK/ D
       EQUIVALENCE (D(1),IST(1)),(R(1),D(1))
       CALL ENTER(1)
C DETERMINE ITERATION TOLERANCE
       CALL BANM(N,ML,M,G,IG,SIZE)
       EPS=SIZE*R1MACH(4)
C SUBTRACT EIGENVALUE FROM DIAGONAL OF G
       DO 10 I=1,N
          G(ML,I)=G(ML,I) - EVAL
  10   CONTINUE
C GET SPACE FROM STACK FOR AL,INTER, AND SCRATCH VECTOR
       JAL =ISTKGT(N*(ML-1),3)
       JINTER=ISTKGT(N,2)
       JX=ISTKGT(N,3)
C GET LU DECOMPOSITION OF MATRIX
       CALL BALU(N,ML,M,G,IG,R(JAL),ML-1,IST(JINTER),MU,EPS)
C OBTAIN INITIAL RIGHT HAND SIDE
       IF (NERROR(IERR).NE.0) CALL ERROFF
       DO 20 I=1,N
          EVEC(I)=1.0
  20   CONTINUE
       CALL BABS(N,G,IG,EVEC,N,1,MU)
       LIM=0
       JJ=ISAMAX(N,EVEC,1)
       SC=1.0/EVEC(JJ)
C SCALE FIRST RHS TO HAVE INFINITY NORM OF 1
       CALL SSCAL(N,SC,EVEC,1)
C ITERATIVE PHASE BEGINS HERE
  30   LIM=LIM+1
C MAKE A COPY OF OLD APPROXIMATION
       CALL MOVEFR(N,EVEC,R(JX))
C GET NEW APPROXIMATION OF EIGNVECTOR
       CALL BAFS(N,ML,R(JAL),ML-1,IST(JINTER),EVEC,N,1)
       CALL BABS(N,G,IG,EVEC,N,1,MU)
       BET=1.0/EVEC(JJ)
```

```
      JJ=ISAMAX(N,EVEC,1)
      SC2=1.0/EVEC(JJ)
C COMPUTE CONVERGENCE CRITERIA
      D1=0.0
      DO 40 I=1,N
         JXI=JX-1+I
         D1=AMAX1(D1,ABS((R(JXI)-BET*EVEC(I))*SC2))
 40   CONTINUE
      SC=SC2
      CALL SSCAL(N,SC,EVEC,1)
C TEST FOR CONVERGENCE AND IF ITERATION LIMIT EXCEEDED
      IF (D1.GT.EPS.AND.LIM.LT.LIMIT) GO TO 30
      CALL LEAVE
      RETURN
      END
```

To show that the above program works, a mainline program was written that packed into G the matrix

```
 -.75    -.5
 -.5     1.0   -1.0
        -1.0    1.0   -1.0
               -1.0    1.0   -1.
                       ...
                             -1.0    1.0   -1.0
                                    -1.0    1.0   -.5
                                             -.5   -.75
```

and invoked EIGVEC with the eigenvalue at -1.0.

```
      INTEGER N, I IWRITE, I1MACH
      REAL G(3, 200), EVEC(100)
      N=10
      DO 10 I=1,N
         G(1,I)=-1.0
         G(2,I)=1.0
         G(3,I)=-1.0
 10   CONTINUE
      G(2,1)=-.75
      G(2,N)=-.75
      G(3,1)=-.5
      G(1,2)=-.5
      G(1,N)=-.5
      G(3,N-1)=-.5
      IWRITE=I1MACH(2)
      CALL EIGVEC(N,3,2,G,3,-1.0,EVEC,2)
      DO 20 I=1,N
         WRITE(IWRITE,21)EVEC(I)
20    CONTINUE
21    FORMAT(12H EIGENVECTOR,F16.8)
      STOP
```

```
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, which has about 8 decimal digits of precision, the following was printed:

```
EIGENVECTOR    1.00000000
EIGENVECTOR    0.49999998
EIGENVECTOR    0.49999996
EIGENVECTOR    0.49999994
EIGENVECTOR    0.49999991
EIGENVECTOR    0.49999989
EIGENVECTOR    0.49999986
EIGENVECTOR    0.49999982
EIGENVECTOR    0.49999976
EIGENVECTOR    0.99999938
```

Since the true eigenvector is $(1.0, 0.5, 0.5, ..., 0.5, 1.0)^T$, the fact that the eigenvector was computed by solving an ill-conditioned linear system did not affect our answer.

BACE — LU decomposition of a banded unsymmetric matrix with condition estimation

Purpose: BACE (BAnded matrix Condition Estimation) gives a lower bound for the condition number of a general banded matrix A. It also returns the LU decomposition of the matrix and may be used in place of BADC in a linear equation package.

Usage: CALL BACE (N, ML, M, G, IG, AL, IAL, INTER, MU, COND)

N	→	the order of the matrix A
ML	→	the number of nonzero bands on and below the diagonal of A
M	→	the number of nonzero bands in A
G	→	a matrix into which the matrix A has been packed as follows: $G(ML + j-i, i) = a_{ij}$ i.e. the leftmost diagonal of A is in the first row of G (See the introduction to this chapter.) G should be dimensional (IG,KG) in the calling program, where IG≥M and KG≥N.
	←	the upper triangular factor of A (see **Note 2**)
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
AL	←	the lower triangular factor of A (see **Note 2**)
IAL	→	the row (leading) dimension of AL, as dimensioned in the calling program
INTER	←	an integer vector of length N recording the row interchanges performed during the decomposition (see **Note 2**)
MU	←	the number of nonzero bands in the upper triangular factor
COND	←	an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{\text{Bd}P}$, there may be no correct digits.

Note 2: After execution of BACE, the arrays INTER and AL are suitable for input into the forward solve subroutine BAFS, and G is suitable for input into the back solver BABS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix and U is an upper triangular matrix. On return from BACE the element u_{ij} is contained in G(j–i+1,i), so that the main diagonal occupies the first row of the G matrix, the first super diagonal occupies the second row, etc. The matrix P can be obtained from INTER (see the introduction to this chapter), and the i^{th} column of the L matrix appears permuted in the i^{th} column of the AL array. Since the diagonal elements of L are all 1, they are not stored.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M
5	IAL < ML − 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DBACE with G, AL and EPS declared double precision.

Complex version: CBACE with G and AL declared complex

Storage: N real (double precision for DBACE, complex for CBACE) locations of scratch storage in the dynamic storage stack

Time: at most N×(M×ML+6×M+ML) additions
at most N×(ML×M+3×M+ML−1) multiplications
N×(ML+1) divisions

Method: Gaussian elimination with partial pivoting
See the reference below for the method used to estimate the condition number.
BACE calls BALU with EPS=0.0

See also: BADC, BAFS, BABS, BALU, BALE, BASS

Author: Linda Kaufman

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: In the following example we obtain estimates of the condition numbers of five matrices whose nonzero elements are given by $a_{ij} = i + j$ For each matrix, the 2×ML−1 main diagonals are nonzero. The program fragment packing the matrix A into the array G uses the fact that traversing a column of G is equivalent to traversing a row of A. The extra values that get put into G by the program below are made zero inside the subroutine BACE.

In this example we compare the condition number $K = ||A||\ ||A^{-1}||$ with the estimate obtained by BACE. In the program below A^{-1} is computed one column at a time and K is computed using the 1-norm. In the 1-norm, $||A||$ is the maximum column sum. In our example $||A||$ is M×(N−ML+1)×2, which is obtained by summing the M elements in column N−ML+1.

```
      INTEGER IG, IGL, N, ML, M, I, J, MU, IWRITE, I1MACH
      INTEGER INTER(80)
      REAL G(13, 80), B(80), X(80), GL(6, 80)
      REAL START, FLOAT, AINNO, COND, CONDNO, ABS, AINNOI
      IG=13
      IGL=6
      N=80
      IWRITE=I1MACH(2)
      DO 60 ML=2,6
C
C CONSTRUCT THE MATRIX A(I,J)=I+J AND PACK IT INTO G
            M=2*ML - 1
            START=-FLOAT(M-ML)
            DO 20 I=1,N
               G(1,I)=START+FLOAT(2*I)
               DO 10 J=2,M
                  G(J,I)=G(J-1,I)+1.
```

```
  10           CONTINUE
  20        CONTINUE
C
C DETERMINE AN ESTIMATE OF THE CONDITION NUMBER
C AND COMPUTE THE LU DECOMPOSITION
C
            CALL BACE(N,ML,M,G,IG,GL,IGL,INTER,MU,COND)
C
C DETERMINE THE NORM OF THE INVERSE MATRIX BY
C SOLVING FOR ONE COLUMN OF THE INVERSE MATRIX
C AT A TIME
C
            AINNO=0.0
            DO 50 I=1,N
C
C FIND THE ITH COLUMN OF THE INVERSE MATRIX BY
C SETTING THE RIGHT HAND SIDE TO THE ITH COLUMN
C OF THE IDENTITY MATRIX
C
               DO 30 J=1,N
                  B(J)=0.0
  30           CONTINUE
               B(I)=1.0
               CALL BAFS(N,ML,GL,IGL,INTER,B,80,1)
               CALL BABS(N,G,IG,B,80,1,MU)
C FIND THE NORM OF THE ITH COLUMN
               AINNOI=0.0
               DO 40 J=1,N
                  AINNOI=AINNOI+ABS(B(J))
  40           CONTINUE
               IF(AINNOI.GT.AINNO)AINNO=AINNOI
  50         CONTINUE
             WRITE(IWRITE,51)ML
  51         FORMAT(/6H ML IS ,I4)
             WRITE(IWRITE,52)COND
  52         FORMAT(22H CONDITION ESTIMATE IS,1PE15.7)
             CONDNO=AINNO*FLOAT(M*(N-ML+1)*2)
             WRITE(IWRITE,53)CONDNO
  53         FORMAT(22H TRUE CONDITION NO. IS,1PE15.7)
  60      CONTINUE
          STOP
          END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
ML IS   2
CONDITION ESTIMATE IS  6.1040422E 03
TRUE CONDITION NO. IS  1.8941539E 04

ML IS   3
CONDITION ESTIMATE IS  5.9552785E 02
TRUE CONDITION NO. IS  2.1467948E 03

ML IS   4
CONDITION ESTIMATE IS  1.0581919E 07
TRUE CONDITION NO. IS  2.9246300E 07

ML IS   5
CONDITION ESTIMATE IS  3.2465961E 04
TRUE CONDITION NO. IS  6.7086635E 04

ML IS   6
CONDITION ESTIMATE IS  3.5264744E 07
TRUE CONDITION NO. IS  8.6640243E 07
```

In the comparison above of the condition number estimated by BACE and the true condition number, the order of magnitude of the estimate is correct, which is all one is usually interested in. Note that the inverse of a band matrix is usually a full $n \times n$ matrix, and should rarely be calculated.

BADC — decomposition of a banded unsymmetric matrix

Purpose: BADC (BAnded matrix DeComposition) finds the LU decompostion of a general banded matrix A using partial pivoting.

Usage: CALL BADC (N, ML, M, G, IG, AL, IAL, INTER, MU)

N → the order of the matrix A

ML → the number of nonzero bands on and below the diagonal of A

M → the number of nonzero bands in A

G → a matrix into which the matrix A has been packed as follows:

$$G(ML + j - i, i) = a_{ij}$$

i.e. the leftmost diagonal of A is in the first row of G
(See the introduction to this chapter.)
G should be dimensional (IG,KG) in the calling program, where IG≥M and KG≥N.

← the upper triangular factor U of A (see **Note 1**)

IG → the row (leading) dimension of G, as dimensioned in the calling program

AL ← the lower triangular factor of A (see **Note 1**)

IAL → the row (leading) dimension of AL, as dimensioned in the calling program

INTER → an integer vector of length N recording the row interchanges performed during the decomposition (see **Note 1**)

MU ← the number of nonzero bands in the upper triangular factor (MU≤M)

Note 1: After execution of BADC, the arrays INTER and AL are suitable for input into the forward solve subroutine BAFS and G is suitable for input into the back solve subroutine BABS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix and U is an upper triangular matrix. On return from BADC the element u_{ij} is contained in G(j−i+1,i), so that the main diagonal occupies the first row of the G matrix, the first super diagonal occupies the second row, etc. The matrix P can be obtained from INTER(see the introduction to this chapter), and the i^{th} column of the L matrix appears permuted in the i^{th} column of the AL array. Since the diagonal elements of L are all 1, they are not stored.

Note 2: MU≤M

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M
5	IAL < ML − 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DBADC with G, AL and EPS declared double precision.

Complex version: CBADC with G and AL declared complex

Storage: None

Time: M×N+(ML−1)×N×(M−ML) ≤ additions ≤ (ML−1)×N×(M−1)+N×M
(ML−1)×N×(M−ML) ≤ multiplications ≤ (ML−1)×N×(M−1)
N×(ML−1) divisions

Method: Gaussian elimination with partial pivoting
BADC calls BALU after setting EPS = $||A||\,\varepsilon$ where ε is the machine precision, i.e. the value returned by R1MACH(4) (or, for double precision, by D1MACH(4)).

See also: BALU, BAFS, BABS, BACE, BALE, BASS

Author: Linda Kaufman

Example: In this example we implement a linearized version of Newton's method for solving $f(x)$=0 where f and x are vectors of length N. Newton's method is normally given as

Set k to 0. Initialize $x^{(0)}$
Until $||f(x^{(k)})|| \le \varepsilon$ iterate as follows:

Solve $J(x^{(k)})y = f(x^{(k)})$
where $J_{i,l} = \dfrac{\partial f_i}{\partial x_l}$. Set $x^{(k+1)} = x^{(k)} - y$
Set k to k + 1

In some problems, especially those occurring in algorithms for solving time-varying partial differential equations, J is banded and costly to evaluate. Thus to solve f(x)=0, a linearized Newton's method is used in which $x^{(k+1)}$ is updated according to the formula $x^{(k+1)} = x^{(k)} - J(x^{(0)})^{-1} f(x^{(k)})$. In the following subroutine, implementing a linearized Newton method, FUN and JAC are assumed to be user provided functions which evaluate the function and its Jacobian. The function SNRM2 carefully computes the 2-norm of a vector.

```
      SUBROUTINE NEWTON(N,M,ML,X,EPS,FUN,JAC,LIMIT,F)
C
C THIS SUBROUTINE IMPLEMENTS A LINEARIZED FORM OF NEWTONS
C METHOD TO FIND THE ZERO OF A FUNCTION F DEFINED BY
C FUN, WHOSE BAND JACOBIAN (WITH BANDWIDTH M AND ML
C LOWER DIAGONALS) IS EVALUATED IN JAC. LIMIT GIVES
C A BOUND ON THE NUMBER OF ITERATIONS AND IN F THE
C FINAL FUNCTION VALUE IS RETURNED.
C
      INTEGER N, ML, M, LIMIT
      INTEGER JG, JAL, JINTER, ISTKGT, MU, LIM, I
      INTEGER IST(1000)
      REAL EPS, X(N), F(N)
      REAL FU, SNRM2, R(1000)
      DOUBLE PRECISION D(500)
      EXTERNAL FUN,JAC
      COMMON /CSTAK/ D
      EQUIVALENCE (D(1),R(1)),(D(1),IST(1))
C
C GET SPACE FOR G,INTER, AND AL FROM
C THE STORAGE STACK
```

```
C
       JG= ISTKGT(M*N,3)
       JAL = ISTKGT ((ML-1)*N,3)
       JINTER = ISTKGT(N,2)
       CALL JAC(N,M,ML,X,R(JG),M)
       CALL BADC(N,ML,M,R(JH),M,R(JAL),ML-1,IST(JINTER),MU)
       LIM=0
  10   CALL FUN(N,X,F)
       FU=SNRM2(N,F,1)
C
C CHECK FOR CONVERGENCE OR IF ITERATION LIMIT IS REACHED
C
      IF (FU.LE.EPS.OR.LIM.GT.LIMIT) RETURN
      LIM=LIM+1
C SOLVE THE LINEAR SYSTEM
      CALL BAFS(N,ML,R(JAL),ML-1,IST(JINTER),F,N,1)
      CALL BABS(N,R(JG),M,F,N,1,MU)
C CORRECT THE CURRENT ESTIMATE OF THE SOLUTION
      DO 20 I=1,N
         X(I)=X(I)-F(I)
  20  CONTINUE
      GO TO 10
      END
```

BALE — banded linear system solver

Purpose: BALE (BAnded Linear Equation solution) solves AX = B where A is a general banded matrix.

Usage: CALL BALE (N, ML, M, G, IG, B, IB, NB)

N	→	the number of equations
ML	→	the number of nonzero bands on and below the diagonal of A
M	→	the number of nonzero bands of A
G	→	a matrix into which the matrix A has been packed as follows: $G(ML + j-i, i) = a_{ij}$ i.e. the leftmost band of A is in the first row of G (See the introduction to this chapter.) G should be dimensional (IG,KG) in the calling program, where IG≥M and KG≥N. G is overwritten during the solution.
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
B	→	the matrix of right-hand sides, dimensioned (IB,KB) in the calling program, where IB≥N and KB≥NB.
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides

Note 1: Unless the given matrix, A, is known in advance to be well-conditioned, the user should use the routine BASS in place of BALE.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance,* should not use BALE, but should call subprograms BADC, BAFS and BABS. (See the example in BADC.) BADC is called once to get the LU decomposition (see the introduction to this chapter) and then the pair, BAFS (forward solve) and BABS (back solve), is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M
5	IB < N
6	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DBALE with G and B declared double precision.

Complex version: CBALE with G and B declared complex

Storage: N integer locations of scratch storage in the dynamic storage stack

Time: at most N×(M×(ML+1)+(ML+M−2)×(NB−1)) additions
at most N×(ML×M+(ML+M−2)×(NB−1)) multiplications
N×(NB+ML−1) divisions

Method: Gaussian elimination with partial pivoting.
Transformations to A are not saved.

See also: BABS, BACE, BADC, BAFS, BASS, BALU

Author: Linda Kaufman

Example: In this example the relative efficiencies of BALE and BASS are compared for systems of various bandwidths and dimensions and various numbers of right-hand sides. The subroutine BASS solves a linear system with a banded matrix and also returns an estimate of the condition number of the matrix. The matrix used in this example is given by the formula $a_{i,j} = |i-j|$ Since each diagonal of the matrix A corresponds to a particular row of the array G, and since all the elements on any diagonal of the matrix in our example are the same, each row of G in the program below was set to a constant. The right-hand sides were chosen randomly.

The function ILAPSZ is a timer on the Honeywell 6000 machine

with about a 1% accuracy.
It counts in 1/64 milliseconds.

```
      INTEGER IG, IWRITE, I1MACH, N, ML, II, MP1, I, K
      INTEGER IB, NB, IT, ILAPSZ
      REAL G(19, 100), B(100, 10), BB(100, 10), GG(19, 100)
      REAL COND, TIME1, TIME2, UNI
C
C THIS PROGRAM SOLVES BANDED SYSTEMS USING BALE AND
C BASS AND COMPARES THE TIME FOR EACH OF THEM. THE
C SYSTEMS HAVE VARIOUS BANDWIDTHS,DIMENSIONS, AND
C NUMBERS OF RIGHT-HAND SIDES
      DOUBLE PRECISION D(600)
      COMMON /CSTAK/ D
C MAKE SURE THE STACK MECHANISM HAS SUFFICIENT SPACE
C FOR BASS
      CALL ISTKIN(1200,3)
      IG=19
      IWRITE=I1MACH(2)
      IB=100
      DO 70 N=50,100,50
         DO 60 ML=2,10,8
            M=2*ML - 1
            MP1=M+1
            DO 50 NB=1,10,9
               WRITE(IWRITE,1)N,M,NB
  1            FORMAT(/5H N IS,I4,6H M IS ,I3,7H NB IS ,I3)
C
C CONSTRUCT THE MATRIX A(I,J)=ABS(I-J) AND PACK IT INTO G
C AND MAKE A COPY OF THE MATRIX SO THE SYSTEM CAN BE
C SOLVED WITH BOTH BALE AND BASS
C
               K=ML - 1
               DO 20 I=1,ML
                  II=MP1 - I
                  DO 10 J=1,N
                     G(I,J)=K
                     GG(I,J)=K
                     G(II,J)=K
                     GG(II,J)=K
  10              CONTINUE
                  K=K - 1
  20           CONTINUE
C
C CONSTRUCT RANDOM RIGHT-HAND SIDES
C AND MAKE A COPY
C
               DO 40 I=1,NB
                  DO 30 II=1,N
                     B(II,I)=UNI(0)
                     BB(II,I)=B(II,I)
  30              CONTINUE
  40           CONTINUE
C
C SOLVE THE SYSTEM USING BOTH BASS AND BALE
C
               IT=ILAPSZ(0)
               CALL BASS(N,ML,M,G,IG,B,IB,NB,COND)
               TIME1=(ILAPSZ(0)-IT)/64.0
               WRITE(IWRITE,41)TIME1
  41           FORMAT(34H TIME FOR BASS IN MILLISECONDS IS ,F10.1)
               IT=ILAPSZ(0)
               CALL BALE(N,ML,M,GG,IG,BB,IB,NB)
               TIME2=(ILAPSZ(0)-IT)/64.0
               WRITE(IWRITE,42)TIME2
  42           FORMAT(34H TIME FOR BALE IN MIILISECONDS IS ,F10.1)
  50        CONTINUE
  60     CONTINUE
  70  CONTINUE
```

```
      STOP
      END
```

When the above program was run on the Honeywell 6000 machine at Bell Labs with an optimizing compiler, the following was printed:

```
N IS  50 M IS   3 NB IS   1
TIME FOR BASS IN MILLISECONDS IS       50.0
TIME FOR BALE IN MIILISECONDS IS       20.0

N IS  50 M IS   3 NB IS  10
TIME FOR BASS IN MILLISECONDS IS       99.0
TIME FOR BALE IN MIILISECONDS IS       58.2

N IS  50 M IS  19 NB IS   1
TIME FOR BASS IN MILLISECONDS IS      200.8
TIME FOR BALE IN MIILISECONDS IS      147.8

N IS  50 M IS  19 NB IS  10
TIME FOR BASS IN MILLISECONDS IS      397.5
TIME FOR BALE IN MIILISECONDS IS      315.8

N IS 100 M IS   3 NB IS   1
TIME FOR BASS IN MILLISECONDS IS      102.8
TIME FOR BALE IN MIILISECONDS IS       36.4

N IS 100 M IS   3 NB IS  10
TIME FOR BASS IN MILLISECONDS IS      204.0
TIME FOR BALE IN MIILISECONDS IS      112.9

N IS 100 M IS  19 NB IS   1
TIME FOR BASS IN MILLISECONDS IS      416.6
TIME FOR BALE IN MIILISECONDS IS      302.4

N IS 100 M IS  19 NB IS  10
TIME FOR BASS IN MILLISECONDS IS      859.0
TIME FOR BALE IN MIILISECONDS IS      680.3
```

The above example indicates that the overhead for computing the condition estimate in BASS can be quite substantial for narrow banded systems with one right-hand side, but inconsequential if the bandwidth is large or if the system has many right-hand sides. The example also indicates that the execution time is linear in the number of equations, but certainly not linear in the number of right-hand sides. Users with many right-hand sides, which are known in advance and which all correspond to the same coefficient matrix, should obviously not invoke BALE for each new right-hand side, but call BALE once with NB set appropriately.

BALU — LU decomposition of a banded unsymmetric matrix

Purpose: BALU (BAnded matrix LU decomposition) finds the LU decompostion of a general banded matrix A using partial pivoting. It allows the user to specify a threshold for considering a matrix singular. BALU is called by the LU decomposition routines BACE and BADC.

Usage: CALL BALU (N, ML, M, G, IG, AL, IAL, INTER, MU, EPS)

N	$\rightarrow$	the order of the matrix A
ML	$\rightarrow$	the number of nonzero bands on and below the diagonal of A
M	$\rightarrow$	the number of nonzero bands in A
G	$\rightarrow$	a matrix into which the matrix A has been packed as follows: G (ML + j–i, i) = a_{ij} i.e. the leftmost diagonal of A is in the first row of G (See the introduction to this chapter.) G should be dimensioned (IG, KG) in the calling program, where IG≥M and KG≥N.
	$\leftarrow$	the upper triangular factor of A (see **Note 2**)
IG	$\rightarrow$	the row (leading) dimension of G, as dimensioned in the calling program
AL	$\leftarrow$	the lower triangular factor of A (see **Note 2**)
IAL	$\rightarrow$	the row (leading) dimension of AL, as dimensioned in the calling program
INTER	$\leftarrow$	an integer vector of length N recording the row interchanges performed during the decomposition (see **Note 2**)
MU	$\leftarrow$	the number of nonzero bands in the upper triangular factor
EPS	$\rightarrow$	if A = LU and there exists an index k such that $\lvert u_{kk} \rvert \leq$ EPS then A is considered singular

Note 1: After execution of BALU, (if the matrix is not found to be singular), the value of the determinant is INTER(N) × G(1,1) × G(1,2) × . . . × G(1,N). INTER(N) contains the sign of the permutation.

Note 2: After execution of BALU, the arrays INTER and AL are suitable for input into the forward solve subroutine BAFS and G is suitable for input into the back solve subroutine BABS. The LU decomposition of A satisfies the equation PA=LU where P is a permutation matrix, L is a unit lower triangular matrix and U is an upper triangular matrix. On return from BALU the element u_{ij} is contained in G(j−i+1,i), so that the main diagonal occupies the first row of the G matrix, the first super diagonal occupies the second row, etc. The matrix P can be obtained from INTER (see the introduction to this chapter), and the i^{th} column of the L matrix appears permuted in the i^{th} column of the AL array. Since the diagonal elements of L are all 1, they are not stored.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M
5	IAL < ML − 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DBALU with G, AL and EPS declared double precision.

Complex version: CBALU with G and AL declared complex

Storage: None

Time: N×(ML−1) divisions
(ML−1)×N×(M−ML)≤multiplications ≤(ML−1)×N×(M−1)
(ML−1)×N×(M−ML)≤ additions ≤(ML−1)×N×(M−1)

Method: Gaussian elimination with partial pivoting

See also: BADC, BAFS, BABS, BACE, BALE, BASS

Author: Linda Kaufman

Example: The program below computes the determinant of a band matrix stored in G in packed form. After the call to BALU the determinant is just INT(N) × the product of the elements in the first row of G. Since the subroutine BADET requires the user to provide only the space needed to hold the original matrix, it uses the stack mechanism provided in PORT to get the extra space needed by BALU. The subroutine tries to avoid underflow and overflow during the calculation. The subroutine UMKFL is used to decompose a floating-point number, F, into a mantissa, S, and an exponent E such that $F = Sb^E$ where b is the base of the machine and $1/b \leq |S| < 1$

```
      SUBROUTINE BADET(N,ML,M,A,IA,DETMAN,IDETEX)
C
C THIS SUBROUTINE COMPUTES THE DETERMINANT OF A
C BANDED MATRIX STORED IN PACKED FORM IN A
C THE DETERMINANT IS COMPUTED AS DETMAN*BETA**IDETEX,
C WHERE BETA IS THE BASE OF THE MACHINE AND
C DETMAN IS BETWEEN 1/BETA AND 1 IN ABSOLUTE VALUE
C
      INTEGER ML, M, N, IA, IDETEX
      INTEGER E, ISPAC, IALOW, ISTKGT, ISIGN, INTER, I, MU
      INTEGER IN(1000)
      REAL A(IA,1), DETMAN, BETA, ONOVBE, S
      REAL R(1000)
      DOUBLE PRECISION D(500)
      COMMON /CSTAK/D
      EQUIVALENCE(D(1),R(1)),(D(1),IN(1))
C
C ALLOCATE SPACE FROM THE STACK FOR THE PIVOT ARRAY
C AND THE EXTRA SPACE TO HOLD THE LOWER TRIANGLE
C
      ISPAC=(ML-1)*N
      IALOW=ISTKGT(ISPAC,3)
      INTER=ISTKGT(N,2)
      CALL BALU(N,ML,M,A,IA,R(IALOW),ML-1,IN(INTER),MU,0.0)
C
C THE DETERMINANT IS THE PRODUCT OF THE ELEMENTS OF
C ROW 1 OF A TIMES THE LAST ELEMENT IN THE ARRAY INTER.
C WE TRY TO COMPUTE THIS PRODUCT IN A WAY THAT WILL
C AVOID UNDERFLOW AND OVERFLOW.
C
      BETA=FLOAT(I1MACH(10))
      ONOVBE=1.0/BETA
      ISIGN=INTER+N-1
      DETMAN=IN(ISIGN)*ONOVBE
      IDETEX=1
      DO 10 I=1,N
         CALL UMKFL(A(1,I),E,S)
         DETMAN=DETMAN*S
         IDETEX=IDETEX+E
         IF (ABS(DETMAN).GE.ONOVBE) GO TO 10
            IDETEX=IDETEX-1
            DETMAN=DETMAN*BETA
10     CONTINUE
       CALL ISTKRL(2)
       RETURN
       END
```

BAML — banded matrix - vector multiplication

Purpose: BAML matrix (BAnded matrix MuLtiplication) forms the product Ax where A is a general banded matrix stored in packed form.

Usage: CALL BAML (N, ML, M, G, IG, X, B)

N	→	the order of the matrix A
ML	→	the number of nonzero bands on and below the diagonal of A
M	→	the number of nonzero bands of A
G	→	a matrix into which the matrix A has been packed as follows: $G(ML + j-i, i) = a_{ij}$ i.e. the leftmost band of A is in the first row of G (See the introduction to this chapter.) G should be dimensioned (IG,KG) in the calling program, where IG≥M and KG≥N.
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
X	→	the vector x to be multiplied
B	←	the vector Ax

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M

Double-precision version: DBAML with G, X, and B declared double precision.

Complex version: CBAML with G, X, and B declared complex

Storage: None

Time: M×N additions
M×N multiplications

See also: BABS, BACE, BADC, BALU, BALE, BASS

Author: Linda Kaufman

Example: This example checks the consistency of BAML and BASS the banded system solver. First the example uses BAML to compute for a given vector x and a given matrix A the vector $b = Ax$. Then the problem is inverted, i.e., BASS is used to find the vector x which satisfies $Ax = b$. This x is then compared with the original vector. The vector x is generated randomly and the 10×10 matrix A is given by

```
0  1  2
1  0  1  2
2  1  0  1  2
   2  1  0  1  2
            .  .  .
            2  1  0  1  2
               2  1  0  1
                  2  1  0
```

```
      INTEGER IG, M, ML, N, I, IWRITE, I1MACH
      REAL G(5,20), X(20), B(20), UNI, ERR, SASUM, ABS, COND
      IG=5
      M=5
      N=10
      ML=3
C
C CONSTRUCT THE A MATRIX AND PACK IT INTO G
C
       DO 10 I=1,N
          G(1,I)=2.0
          G(2,I)=1.0
          G(3,I)=0.0
          G(4,I)=1.0
          G(5,I)=2.0
 10    CONTINUE
C
C CONSTRUCT A RANDOM VECTOR
C
       DO 20 I=1,N
          X(I)=UNI(0)
 20    CONTINUE
C
C CONSTRUCT B=AX
C
       CALL BAML(N,ML,M,G,IG,X,B)
C
C SOLVE THE SYSTEM AX=B
C
       CALL BASS(N,ML,M,G,IG,B,N,1,COND)
C
C PRINT OUT THE TRUE SOLUTION AND THE COMPUTED SOLUTION
C
       IWRITE=I1MACH(2)
       WRITE(IWRITE,21)
 21    FORMAT(34H TRUE SOLUTION   COMPUTED SOLUTION)
       WRITE(IWRITE,22)(X(I),B(I),I=1,N)
```

```
  22      FORMAT(1H ,2E17.8)
C
C COMPUTE THE RELATIVE ERROR
C
          ERR=0.0
          DO 30 I=1,N
             ERR=ERR+ABS(B(I)-X(I))
  30      CONTINUE
          ERR=ERR/SASUM(N,X,1)
          WRITE(IWRITE,31)ERR
  31      FORMAT(19H RELATIVE ERROR IS ,1PE15.7)
          WRITE(IWRITE,32)COND
  32      FORMAT(20H CONDITION NUMBER IS,1PE15.7)
          STOP
          END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, which has a machine precision of $1.\times10^{-8}$, the following was printed:

```
TRUE SOLUTION   COMPUTED SOLUTION
   0.22925607E 00   0.22925605E 00
   0.76687502E 00   0.76687499E 00
   0.68317685E 00   0.68317685E 00
   0.50919111E 00   0.50919110E 00
   0.87455959E 00   0.87455962E 00
   0.64464101E 00   0.64464102E 00
   0.84746840E 00   0.84746839E 00
   0.35396343E 00   0.35396345E 00
   0.39889160E 00   0.39889155E 00
   0.45709422E 00   0.45709425E 00
RELATIVE ERROR IS   3.8447574E-08
CONDITION NUMBER IS  4.3333333E 01
```

The condition number of the matrix and the precision of the Honeywell suggest that even in the absence of roundoff error in BAML, a relative error of 4.3×10^{-7} would not be surprising. The value computed above is quite reasonable.

BANM — norm of a banded unsymmetric matrix

Purpose: BANM (BAnded matrix NorM) computes the infinity norm of a general banded matrix A. The infinity norm is defined as $\max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$

Type: Real function

Usage: <answer> = BANM (N, ML, M, G, IG)

N	→	the number of rows in A
ML	→	the number of nonzero bands on and below the diagonal of A
M	→	the number of nonzero bands in A
G	→	a matrix into which the matrix A has been packed as follows: G (ML + j−i, i) = a_{ij} i.e. the leftmost diagonal of A is in the first row of G (See the introduction to this chapter.) G should be dimensioned (IG,KG) in the calling program, where IG≥M and KG≥N.
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
<answer>	←	$\max_{1 \le i \le n} \sum_{j=1}^{n} \lvert a_{ij} \rvert$

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M

Double-precision version: DBANM with G and DBANM declared double precision

Complex version: CBANM with G declared complex

Storage: None

Time: N × M additions
N comparisons

See also: BADC, BALU, BALE, BASS, BACE

Author: Linda Kaufman

Example: In this example we verify the correctness of BANM by obtaining norms of five matrices of various bandwidths whose nonzero elements are given by $a_{ij} = i + j$ For each matrix, the 2×ML−1 main diagonals are nonzero. The program fragment packing the matrix A into the array G uses the fact that traversing a column of G is equivalent to traversing a row of A. The extra values that get put into G by the program are not referenced by BANM.

In our example the norm computed by BANM is compared with the true norm, known to be M×(N−ML+1)×2, which is obtained by summing the M elements in column N−ML+1.

```
      INTEGER IG,  ML,  M,  N,  I,  J, IWRITE, I1MACH
      REAL G(13,  80),  START,  BANM,  TRNORM
      IG=13
      N=80
      DO 30 ML=2,6
C
C CONSTRUCT THE MATRIX A(I,J)=I+J AND PACK IT INTO G
C
         M=2*ML-1
         START=-FLOAT(M-ML)
         DO 20 I=1,N
            G(1,I)=START+FLOAT(2*I)
            DO 10 J=2,M
               G(J,I)=G(J-1,I)+1.0
 10         CONTINUE
 20      CONTINUE
C
C PRINT OUT THE NORM CALCULATED FROM BANM AND THE TRUE NORM
C
         TRNORM=M*(N-ML+1)*2
         IWRITE=I1MACH(2)
         WRITE(IWRITE,21)ML
 21      FORMAT(/6H ML IS ,I4)
         WRITE(IWRITE,22)TRNORM,BANM(N,ML,M,G,IG)
 22      FORMAT(15H THE TRUE NORM=,E15.5,15H COMPUTED NORM=,E15.5)
 30   CONTINUE
      STOP
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
ML IS   2
THE TRUE NORM=    0.47400E 03 COMPUTED NORM=    0.47400E 03
```

```
ML IS   3
THE TRUE NORM=    0.78000E 03 COMPUTED NORM=    0.78000E 03

ML IS   4
THE TRUE NORM=    0.10780E 04 COMPUTED NORM=    0.10780E 04

ML IS   5
THE TRUE NORM=    0.13680E 04 COMPUTED NORM=    0.13680E 04

ML IS   6
THE TRUE NORM=    0.16500E 04 COMPUTED NORM=    0.16500E 04
```

BASS — banded linear system solution with condition estimation

Purpose: BASS (BAnded System Solution) solves AX=B where A is a general banded matrix. An estimate of the condition number of A is provided.

Usage: CALL BASS (N, ML, M, G, IG, B, IB, NB, COND)

N	→	the number of equations
ML	→	the number of nonzero bands on and below the diagonal of A
M	→	the number of nonzero bands of A
G	→	a matrix into which the matrix A has been packed as follows: G(ML+j−i, i) = a_{ij} i.e. the leftmost band of A is in the first row of G (See the introduction to this chapter.) G should be dimensioned (IG,KG) in the calling program, where IG≥M and KG≥N. G is overwritten during the solution
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
B	→	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB.
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides
COND	←	an estimate of the condition number of A (See **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as **d** − $\log_{10}$(COND) correct decimal digits. Thus if COND is greater than $10^{\mathrm{Bd}\,P}$, there may be no correct digits.

If the given matrix, A, is known in advance to be well-conditioned, then the user may wish to use the routine BALE, which is a little faster than BASS. Ordinarily, however, the user is strongly urged to choose BASS, and to follow it by a test of the condition estimate.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance,* should not use BASS, but should call subprograms BACE, BAFS and BABS. (See the example of BADC.) BACE is called once to get the LU decomposition (see the introduction to this chapter) and then the pair, BAFS (forward solve) and BABS (back solve), is called for each new right-hand side.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	M < ML
4	IG < M
5	IB < N
6	NB < 1
10 + k*	singular matrix whose rank is at least k

Double-precision version: DBASS with G, B, and COND declared double precision.

Complex version: CBASS with G and B declared complex

Storage: N integer locations and N×ML real (double precision for DBASS, complex for CBASS) locations of scratch storage in the dynamic storage stack

Time: at most N×((ML+M–2)×(NB+1)+M×(ML+5)+3) additions
at most N×((ML+M–2)×(NB+1)+M×(ML+2)+2) multiplications
N×(NB+ML+1) divisions

Method: Gaussian elimination with partial pivoting
See the reference below for the method to estimate the condition number.
BASS calls BACE, BAFS, and BABS

See also: BABS, BACE, BADC, BALU, BALE, BAFS

Author: Linda Kaufman

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: In this example several banded linear systems are solved with various number of diagonals and two right-hand sides each. Estimates of the condition numbers of the coefficient matrices are found and the relative errors in the solution are calculated. In the example the nonzero elements of the matrix are given by $a_{ij}=i+j$ For each matrix the 2×ML−1 main diagonals are nonzero. The program fragment packing the matrix A into the array G uses the fact that traversing a column of G is equivalent to traversing a row of A. The right-hand sides are determined so that the elements of the first solution are all ones and the i^{th} element of the second solution is i. The subroutine BAML, which multiplies a vector by a banded matrix packed appropriately into G, is invoked to compute the right-hand sides.

```
      INTEGER N, IG, ML, M, I, J, IWRITE, I1MACH
      REAL G(13,80), B(80,2), X(80)
      REAL START, FLOAT, ERR, ERR2, ABS, COND
      IG=13
      N=80
      DO 60 ML=2,6
C
C CONSTRUCT THE MATRIX A(I,J)=I+J AND PACK IT INTO G
C
            M=2*ML-1
            START=-FLOAT(M-ML)
            DO 20 I=1,N
               G(1,I)=START+FLOAT(2*I)
               IF(M.EQ.1) GO TO 20
               DO 10 J=2,M
                  G(J,I)=G(J-1,I)+1.
 10            CONTINUE
 20         CONTINUE
C CONSTRUCT FIRST RIGHT-HAND SIDE SO SOLUTION IS ALL 1S
            DO 30 I=1,N
 30            X(I)=1
            CALL BAML(N,ML,M,G,IG,X,B)
C CONSTRUCT THE SECOND COLUMN SO X(I)=I
            DO 40 I=1,N
 40            X(I)=I
            CALL BAML(N,ML,M,G,IG,X,B(1,2))
C SOLVE THE SYSTEM
            CALL BASS(N,ML,M,G,IG,B,80,2,COND)
C COMPUTE THE ERRORS IN THE SOLUTION
            ERR=0.0
            ERR2=0.0
            DO 50 I=1,N
               ERR=ERR+ABS(B(I,1)-1.0)
               ERR2=ERR2+ABS(B(I,2)-FLOAT(I))
 50         CONTINUE
            ERR=ERR/FLOAT(N)
            ERR2=ERR2/FLOAT(N*(N+1))*2.0
            IWRITE=I1MACH(2)
            WRITE(IWRITE,51)ML,COND
 51         FORMAT(/9H WHEN ML=,I4,21H THE CONDITION NO. IS,1PE15.7)
            WRITE(IWRITE,52)ERR
 52         FORMAT(38H REL. ERROR IN THE FIRST SOLUTION IS  ,1PE15.7)
            WRITE(IWRITE,53)ERR2
 53         FORMAT(38H REL. ERROR IN THE SECOND SOLUTION IS ,1PE15.7)
 60      CONTINUE
 70   CONTINUE
      STOP
      END
```

When the above program was executed on on the Honeywell 6000 computer at Bell Labs, the following was printed.

```
WHEN ML=   2 THE CONDITION NO. IS  6.1040422E 03
REL. ERROR IN THE FIRST SOLUTION IS    5.0114468E-07
REL. ERROR IN THE SECOND SOLUTION IS   6.0025235E-07

WHEN ML=   3 THE CONDITION NO. IS  5.9552785E 02
REL. ERROR IN THE FIRST SOLUTION IS    1.2554228E-07
REL. ERROR IN THE SECOND SOLUTION IS   1.1807790E-07

WHEN ML=   4 THE CONDITION NO. IS  1.0581919E 07
REL. ERROR IN THE FIRST SOLUTION IS    5.9645883E-04
REL. ERROR IN THE SECOND SOLUTION IS   2.1994722E-03

WHEN ML=   5 THE CONDITION NO. IS  3.2465961E 04
REL. ERROR IN THE FIRST SOLUTION IS    2.4201348E-06
REL. ERROR IN THE SECOND SOLUTION IS   7.6222429E-07

WHEN ML=   6 THE CONDITION NO. IS  3.5264744E 07
REL. ERROR IN THE FIRST SOLUTION IS    4.2312816E-04
REL. ERROR IN THE SECOND SOLUTION IS   2.4684287E-03
```

The above program certainly indicates that there is a correlation between the relative errors in the solution and the condition number of the coefficient matrix. Moreover it indicates that the relative error depends also on the choice of the right-hand side. Although the relative errors for some of the systems might appear quite large, they are not unreasonably large in light of the following analysis:

Let Δb represent a perturbation in the right-hand side of a linear system. If $Ax = b$ then $A(x+\Delta x) = b+\Delta b$ where $\frac{||\Delta x||}{||x||} \leq K(A)\left[\frac{||\Delta b||}{||b||}\right]$ where $K(A)$ is the condition number of A, $K(A) = ||A||\ ||A^{-1}||$ and $||\cdot||$, is some norm, e.g., $||x||_1 = \sum_{i=1}^{n}|x_i|$ if x is a vector.

The methods used in our linear equation package are guaranteed to provide an accurate answer to a slightly perturbed problem. If we assume that our method produces the correct answer to a problem where $||\Delta b|| \leq \varepsilon||b||$, where ε is the machine precision, then on the Honeywell 6000 where ε is about 10^{-8}, a relative error for the above problem (when ML=6) of 3.5×10^{-1} is not surprising.

Appendix 4

BANDED< SYMMETRIC< POSITIVE-DEFINITE MATRICES

BPBS - Back Solve
BPCE - Condition Estimation
BPDC - DeComposition
BPFS - Forward Solve
BPLE - Linear Equation solution
BPLD - LDL^T decomposition
BPML - MuLtiplication
BPNM - NorM
BPSS - System Solution

Purpose: BPBS (Banded symmetric Positive definite matrix Back Solve) solves DRX = B where D is diagonal and R is banded unit upper triangular (1's on the diagonal and 0's below the diagonal). It can be used for the back solution phase of a banded linear system solution. (It is used in this way by the routines BPSS and BPLE.)

Usage: CALL BPBS (N, ML, G, IG, B, IB, NB)

N → the number of equations

ML → the number of nonzero diagonals of R (including the unit diagonal)

G → a matrix (which may contain results obtained by the routines BPLD, BPCE, or BPDC) into which D and R are packed as follows:

$$G(1, i) = d_i$$
$$G(j-i+1, i) = r_{ij} \text{ for } j > i$$

(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥ML and KG≥N.

IG → the row (leading) dimension of G, as dimensioned in the calling program

B → the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB.

← the solution X

IB → the row (leading) dimension of B, as dimensioned in the calling program

NB → the number of right-hand sides

Note: BPFS and BPBS can be used directly on the output matrix produced by BPDC, BPLD, or BPCE to solve a banded symmetric positive definite linear system.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	IG < ML
4	IB < N
5	NB < 1
10 + k*	singular D with kth diagonal element 0.0

Double-precision version: DBPBS, with G and B declared double precision

Complex Hermitian version: CBPBS with G and B declared complex

Storage: None

Time: NB×(ML−1)×N additions
NB×(ML−1)×N multiplications
NB×N divisions

See also: BPCE, BPDC, BPFS, BPLD, BPLE, BPSS

Author: Linda Kaufman

Example: The program fragment below solves a linear system AX = B, where A is a symmetric positive definite band matrix. It is assumed that the A matrix has been packed into G according to the scheme G(j–i+1, i)=a_{ij}. The subroutine BPCE factors A into LDL^T, where L is unit lower triangular and D is diagonal. The factors are returned in G so that BPFS can forward solve (solve LY=B) and BPBS can back solve (solve $DL^TX=Y$).

The subroutine BPCE also provides an estimate of the condition number of A. In the code below if the condition number is larger than the reciprocal of the machine precision (given by R1MACH(4)), the matrix is considered too ill-conditioned and the system is not solved.

```
      IWRITE=I1MACH(2)
      CALL BPCE(N,ML,G,IG,COND)
      IF (COND .GT. 1.0/R1MACH(4)) GO TO 10
      CALL BPFS(N,ML,G,IG,B,IB,NB)
      CALL BPBS(N,ML,G,IG,B,IB,NB)
      GO TO 20
10    WRITE(IWRITE,11)
11    FORMAT(26H MATRIX TOO ILL-CONDITIONED)
20    CONTINUE
```

BPCE — LDL^T decomposition with condition estimation

Purpose: BPCE (Banded symmetric Positive definite matrix Condition Estimation) gives a lower bound for the condition number of a banded symmetric positive definite matrix A. It also returns the LDL^T decomposition of A and may be used in a linear equation package.

Usage: CALL BPCE (N, MU, G, IG, COND)

N	$\rightarrow$	the order of the matrix A
MU	$\rightarrow$	the number of nonzero bands on and above the diagonal of A
G	$\rightarrow$	a matrix into which the upper triangular portion of the matrix A has been packed as follows: $G(j-i+1, i) = a_{ij}$ for $j \geq i$ (See the introduction to this chapter.) G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.
	$\leftarrow$	LDL^T decomposition suitable for input into BPFS and BPBS (see **Note 2**)
IG	$\rightarrow$	the row (leading) dimension of G, as dimensioned in the calling program
COND	$\leftarrow$	an estimate of the condition number of A (see **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{Bd P}$, there may be no correct digits.

Note 2: The LDL^T decomposition of A satisfies the equation $A = LDL^T$ where L is lower unit triangular (1's on the diagonal, 0's above the diagonal) and D is diagonal. On return from BPCE, the diagonal of D occupies the first row of G and $G(i-j+1,i) = l_{ij}$ for $i>j$.

Note 3: For complex Hermitian matrices (A = A*, where A* represents the conjugate transpose of A) , the complex version of this subroutine computes the LDL* decomposition and returns the conjugate of L rather than L in G.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU
10 + k*	singular matrix whose rank is at least k
10 + N + k*	the k^{th} principal minor is not positive definite

Double-precision version: DBPCE with G and COND declared double precision

Complex Hermitian version: CBPCE with G declared complex (see **Note 3** above).

Storage: N real (double precision for DBPCE, complex for CBPCE) locations of scratch storage in the dynamic storage stack

Time: $N \times ((MU-1) \times (10+MU/2)+8)$ additions
$N \times ((MU-1) \times (6+MU/2)+4)$ multiplications
$N \times (MU+1)$ divisions

Method: Gaussian elimination without pivoting
See the reference below for the method used to estimate the condition number.
BPCE calls BPLD with EPS=0.0

See also: BPBS, BPDC, BPFS, BPLD, BPLE, BPSS

Authors: Doris Ryan and Linda Kaufman

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: In the following example we obtain estimates of the condition numbers of the matrix

$$\begin{pmatrix} 1+x & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1+x \end{pmatrix}$$

with x=1., .01, and .0001. The matrix is singular when x is 0, and becomes more ill-conditioned as x approaches 0.

In this example we compare the condition number $K = ||A|| \ ||A^{-1}||$ with the estimate obtained from BPCE. In the code below A^{-1} is computed one column at a time and K is computed using the 1-norm. In the 1-norm, $||A||$ is the maximum column sum in absolute value, which is obviously 4.

```
      INTEGER N, MU, IG, K, I, IWRITE, I1MACH, J
      REAL G(2,100), B(200)
      REAL X, COND, AINVNO, AINORM, ABS
      N=100
      X=1.0
      MU=2
      IG=2
      DO 50 K=1,3
C CONSTRUCT MATRIX
         DO 10 I=1,N
            G(1,I)=2.0
            G(2,I)=-1.0
 10      CONTINUE
         G(1,1)=1.0+X
         G(1,N)=1.0+X
C GET ESTIMATE OF CONDITION NUMBER FROM BPCE
         CALL BPCE(N,MU,G,IG,COND)
         IWRITE=I1MACH(2)
         WRITE(IWRITE,11)X
 11      FORMAT(/10H WHEN X IS,E14.6)
         WRITE(IWRITE,12)COND
 12      FORMAT(25H CONDITION ESTIMATE IS   ,E15.8)
C SINCE CONDITION NUMBER IS NORM(A)*NORM(INVERSE(A)),
C FIND THE NORM OF EACH COLUMN OF INVERSE(A). GENERATE
C THE COLUMNS ONE AT A TIME AND REUSE SPACE
         AINVNO=0.0
         DO 40 I=1,N
C GENERATE ITH COLUMN OF IDENTITY MATRIX AS RIGHT HAND SIDE
            DO 20 J=1,N
               B(J)=0.0
 20         CONTINUE
            B(I)=1.0
```

```
C SOLVE AX=B TO GET ITH COLUMN OF A(INVERSE)
               CALL BPFS(N,MU,G,IG,B,N,1)
               CALL BPBS(N,MU,G,IG,B,N,1)
C FIND NORM OF COLUMN
               AINORM=0.0
               DO 30 J=1,N
                  AINORM=AINORM+ABS(B(J))
 30            CONTINUE
               IF(AINVNO.LT.AINORM)AINVNO=AINORM
 40         CONTINUE
            COND=4.0*AINVNO
            WRITE(IWRITE,41)COND
 41         FORMAT(25H TRUE CONDITION NUMBER IS,E15.8)
            X=X/100.0
 50      CONTINUE
         STOP
         END
```

When the above code was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
WHEN X IS  0.100000E 01
CONDITION ESTIMATE IS    0.40807862E 04
TRUE CONDITION NUMBER IS 0.50999824E 04

WHEN X IS  0.100000E-01
CONDITION ESTIMATE IS    0.23329148E 05
TRUE CONDITION NUMBER IS 0.24899523E 05

WHEN X IS  0.100000E-03
CONDITION ESTIMATE IS    0.19933923E 07
TRUE CONDITION NUMBER IS 0.19950487E 07
```

The comparison above of the condition number estimated by BPCE with the true condition number indicates that the order of magnitude (which is all one usually is interested in) of the estimated condition number is correct. Note that the inverse of a band matrix is usually a full $n \times n$ matrix and should rarely be calculated.

BPDC — LDL^T decomposition of a band symmetric positive definite matrix A

Purpose: BPDC (Banded symmetric Positive definite matrix DeComposition) decomposes a banded symmetric positive definite matrix A into LDL^T where L is lower unit triangular (1's on the diagonal and 0's above the diagonal) and D is diagonal. It is called by BPLE as the first step of the solution of a banded symmetric positive definite linear system.

Usage: CALL BPDC (N, MU, G, IG)

N $\longrightarrow$ the number of equations

MU $\longrightarrow$ the number of nonzero bands on and above the diagonal of A

G $\longrightarrow$ a matrix into which the upper triangular portion of the matrix A has been packed as follows:

$$G(j-i+1, i) = a_{ij} \text{ for } j \geq i$$

(See the introduction to the chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.

$\longleftarrow$ LDL^T decomposition suitable for input into BPFS and BPBS (see **Note 1**)

IG $\longrightarrow$ the row (leading) dimension of G, as dimensioned in the calling program

Note 1: The LDL^T decomposition of A satisfies the equation $A = LDL^T$ where L is lower unit triangular (1's on the diagonal, 0's above the diagonal) and D is diagonal. On return from BPDC, the diagonal of D occupies the first row of G and $G(i-j+1,i) = l_{ij}$ for i>j.

Note 2: For complex Hermitian matrices (A = A*, where A* represents the conjugate transpose of A), the complex version of this subroutine computes the LDL* decomposition and returns the conjugate of L, rather than L, in G.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU
10 + k*	singular matrix whose rank is at least k
10 + N + k*	the k^{th} principal minor is not positive definite

Double-precision version: DBPDC, with G declared double precision.

Complex Hermitian version: CBPDC with G declared complex (see **Note 2**).

Storage: None

Time: N×(MU−1)× MU/2 + N×(2MU−1) additions
N×(MU−1)× MU/2 multiplications
N×(MU−1) divisions

Method: BPDC calls BPLD after setting EPS = ||A||ε, where ε is machine precision, i.e. the value returned by R1MACH(4) (or, for double precision, by D1MACH(4)).

See also: BPBS, BPCE, BPFS, BPLD, BPLE, BPSS

Author: Linda Kaufman

Example: This example is designed to indicate the relative efficiency of BPDC, BPLD, and BPCE as a function of the width of the band. All three subroutines compute the same factorization, but the criterion for singularity is treated differently in each of the three. In all the subroutines the matrix is considered singular if for some i, $d_i \leq$ EPS. In BPLD the user provides EPS; in BPDC the subroutine computes EPS (see **Method**); in BPCE, 0.0 is used as EPS. (However, BPCE also provides an estimate of the condition number of the matrix.)

The matrix in the example is derived from the traditional 5-point approximation to the Laplace operator on the unit square. The N×N matrix A has the form

$$\begin{array}{ccccccc} C & -I & & & & & \\ -I & C & -I & & & & \\ & -I & C & -I & & & \\ & & \cdots\cdots & & & & \\ & & & -I & C & -I & \\ & & & & -I & C & \end{array}$$

where I is the identity matrix and C is the matrix

$$\begin{array}{ccccccc} 4 & -1 & & & & & \\ -1 & 4 & -1 & & & & \\ & -1 & 4 & -1 & & & \\ & & & \cdots & & & \\ & & & -1 & 4 & -1 & \\ & & & & -1 & 4 & \end{array}$$

In this problem we vary the sizes of the blocks C and I which changes the bandwidth. For a given blocksize, the number of blocks is varied which changes N.

The subroutine ILAPSZ is a timer on the Honeywell 6000 with about 1% accuracy. It counts in 1/64 milliseconds.

```
      INTEGER IG, MLM1, IWRITE, I1MACH, K, N, MU
      INTEGER NBLOK, KBLOK, KK, I, J, IT, ILAPSZ, IT2
      REAL G(17, 100), G2(17, 100), G3(17, 100)
      REAL COND
      IG=17
      MLM1=4
      IWRITE=I1MACH(2)
      DO 70  K=1,3
         DO 60 N=48,96,48
            MU=MLM1+1
            I=0
            NBLOK=N/MLM1
C
C SET UP THREE MATRICES FOR ELLIPTIC PDE IN 2 DIMENSION
C
            DO 30 KBLOK=1,NBLOK
               DO 20 KK=1,MLM1
                  I=I+1
                  G(1,I)=4.0
                  G(2,I)=-1.0
                  G(MU,I)=-1.0
                  DO 10 J=3,MLM1
                     G(J,I)=0.0
 10               CONTINUE
 20            CONTINUE
               G(2,I)=0.0
```

```
  30         CONTINUE
             DO 50 I=1,N
                DO 40 J=1,MU
                   G2(J,I)=G(J,I)
                   G3(J,I)=G(J,I)
  40            CONTINUE
  50         CONTINUE
             WRITE(IWRITE,51)N,MU
  51         FORMAT(/6H N IS ,I4,30H ,NUMBER OF UPPER DIAGONALS IS,I3)
C TIME DECOMPOSITION BY BPLD
             IT=ILAPSZ(0)
             CALL BPLD(N,MU,G,IG,0.0)
             IT=ILAPSZ(0)-IT
             WRITE(IWRITE,52)IT
  52         FORMAT(14H TIME FOR BPLD,I7)
C TIME DECOMPOSITION BY BPDC
             IT2=ILAPSZ(0)
             CALL BPDC(N,MU,G2,IG)
             IT2=ILAPSZ(0)-IT2
             WRITE(IWRITE,53)IT2
  53         FORMAT(14H TIME FOR BPDC,I7)
C TIME DECOMPOSITION BY BPCE
             IT3=ILAPSZ(0)
             CALL BPCE(N,MU,G3,IG,COND)
             IT3=ILAPSZ(0)-IT3
             WRITE(IWRITE,54)IT3
  54         FORMAT(14H TIME FOR BPCE,I7)
  60     CONTINUE
         MLM1=MLM1*2
  70  CONTINUE
      STOP
      END
```

When the above code was run on the Honeywell 6000 at Bell Laboratories with an optimizing compiler the following was printed

```
 N IS   48 ,NUMBER OF UPPER DIAGONALS IS  5
 TIME FOR BPLD    959
 TIME FOR BPDC   1413
 TIME FOR BPCE   3670

 N IS   96 ,NUMBER OF UPPER DIAGONALS IS  5
 TIME FOR BPLD   1868
 TIME FOR BPDC   2673
 TIME FOR BPCE   7965

 N IS   48 ,NUMBER OF UPPER DIAGONALS IS  9
 TIME FOR BPLD   2395
 TIME FOR BPDC   3181
 TIME FOR BPCE   6377

 N IS   96 ,NUMBER OF UPPER DIAGONALS IS  9
 TIME FOR BPLD   4854
```

```
TIME FOR BPDC   6450
TIME FOR BPCE  13170

N IS   48 ,NUMBER OF UPPER DIAGONALS IS 17
TIME FOR BPLD   7143
TIME FOR BPDC   8227
TIME FOR BPCE  12722

N IS   96 ,NUMBER OF UPPER DIAGONALS IS 17
TIME FOR BPLD  15001
TIME FOR BPDC  17160
TIME FOR BPCE  49809
```

As the example indicates, the cost of computing the condition estimate can be substantially greater than the cost of factoring the matrix when the width of the band is small. The ratio of BPCE to BPDC does not always decrease as N increases, as it does in the case of general linear systems, but depends rather on the scaling that is sometimes done by BPCE to prevent overflow during the calculation of the condition estimate. If no scaling is done, the ratio of the times for BPCE to BPDC remains constant, but for some examples most of the time is spent scaling and the time ratio increases as N increases.

BPFS — band symmetric lower (unit) triangular linear system solution

Purpose: BPFS (Banded symmetric Positive definite matrix Forward Solution) solves LX = B where L is a banded unit lower triangular matrix, (i.e. 1's on the diagonal, 0's above the diagonal). BPFS can be used for the forward solution phase of a band symmetric positive definite linear system. (It is used in this way by the routines BPSS and BPLE.)

Usage: CALL BPFS (N, ML, G, IG, B, IB, NB)

N → the number of equations

ML → the number of nonzero diagonals on and below the diagonal of L

G → a matrix (which may contain the results obtained by the routines BPCE, BPDC, or BPLD) into which L is packed as follows:

$$G(1+i-j, i) = l_{ij} \text{ for } i>j$$

(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥ML and KG≥N.

IG → the row (leading) dimension of G, as dimensioned in the calling program

B → the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB.

← the solution X

IB → the row (leading) dimension of B, as dimensioned in the calling program

NB → the number of right-hand sides

Note 1: BPFS and BPBS can be used directly on the output matrix produced by BPDC, BPLD, or BPCE to solve a general linear system.

Note 2: Users who have to solve a sequence of problems with the same coefficient matrix, but different right-hand sides, *not all known in advance*, should not call BPSS or BPLE repeatedly, but should use BPDC to find the LDL^T decompostion of the coefficient matrix, which can then be used repeatedly (and efficiently) for the sequence of forward solutions (using BPFS) and back solutions (using BPBS) for each set of right-hand sides.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	ML < 1
3	IG < ML
4	IB < N
5	NB < 1

Double-precision version: DBPFS, with G and B declared double precision

Complex Hermitian version: CBPFS with G and B declared complex
G should contain the conjugate of L

Storage: None

Time: N×(ML−1)×NB additions
N×(ML−1)×NB multiplications

See also: BPBS, BPCE, BPDC, BPLD, BPLE, BPSS

Author: Linda Kaufman

Example: In the following example we solve three problems that might arise from a discretization of a 1-dimensional differential equation. The coefficient matrix in each problem has the form

$$\begin{array}{ccccccccc}
1+x & -1 & & & & & & & \\
-1 & 2 & -1 & & & & & & \\
 & -1 & 2 & -1 & & & & & \\
 & & -1 & 2 & -1 & & & & \\
 & & & . & & & & & \\
 & & & & . & & & & \\
 & & & & & -1 & 2 & -1 & \\
 & & & & & & -1 & 1+x &
\end{array}$$

with x=1., .01, and .0001 defining the three problems. To make it easy to detect errors in the solution, the right-hand side has been chosen to make the solution a vector of all 1's.

The program below is an encoding of an iterative refinement algorithm which may be used to obtain a highly accurate solution to a system of linear equations with an ill-conditioned matrix. When the condition number is not excessively high, the iterative refinement algorithm usually returns a solution that is accurate to the working precision of the machine. The iterative refinement algorithm is essentially:

(1) Solve Ax = b
(2) Set tol = $\varepsilon \sum |x_i|$
where ε is the precision of the machine
(3) Compute in double precision the residual, Ax–b,
and put it in the real vector r.
(4) Solve A δx = r
(5) Compute norm = $\sum |\delta x_i|$
(6) Set x to x + δ x
(7) If norm $\leq$ tol stop, else return to step 3

In our code, step (1) is accomplished using the the linear equation solver for band positive definite matrices, BPLE. The subroutine BPLE leaves in the G matrix a decomposition which may be used by BPFS and BPBS to solve problems with the same coefficient matrix but different right-hand sides as in step(4). Since it is possible that the matrix will be so ill-conditioned that the iterative refinement algorithm will diverge, steps (3) through (7) in our code are performed only a finite number of times. This number is chosen to be an upper bound on number of of bits in the mantissa of the floating-point number supported by the machine.

This algorithm is not included in PORT because for double-precision matrices part of the computation would have to be done in extended precision.

The following program has been specifically tailored to the three problems.

```
      INTEGER N, ML, IG, NM1, K, I, IWRITE, I1MACH, IT, IEND, ITER
      REAL G(2,100), B(200), R(200)
      REAL X, ERR, AMAX1, RNORM, BNORM, R1MACH, ABS
      DOUBLE PRECISION DBLE
C CONSTRUCT MATRIX AND RIGHT HAND SIDE SO TRUE SOLUTION IS
C COMPOSED ENTIRELY OF 1S
      N=100
      X=1
      ML=2
      IG=2
      NM1=N-1
      DO 90 K=1,3
         DO 10 I=1,N
            G(1,I)=2.0
            G(2,I)=-1.0
            B(I)=0.0
 10      CONTINUE
         G(1,1)=1.0+X
         G(1,N)=1.0+X
         B(1)=X
         B(N)=X
C SOLVE THE SYSTEM
         CALL BPLE(N,ML,G,IG,B,N,1)
         IWRITE=I1MACH(2)
         WRITE(IWRITE,11)X
 11      FORMAT(/5H X IS,F16.8)
C COMPUTE THE ERROR
         ERR=0.0
         DO 20 I=1,N
            ERR=AMAX1(ERR,ABS(B(I)-1.0))
 20      CONTINUE
         WRITE(IWRITE,21)ERR
 21      FORMAT(22H FOR BPLE THE ERROR IS,F16.8)
         IEND=I1MACH(11)*IFIX(R1MACH(5)/ALOG10(2.0)+1.0)
C FIND THE NORM OF THE SOLUTION
         BNORM=0.0
         DO 30 I=1,N
             BNORM=AMAX1(BNORM,ABS(B(I)))
 30      CONTINUE
C REFINE THE SOLUTION
         DO 60 ITER=1,IEND
            IT=ITER
C COMPUTE THE RESIDUAL R=B-AX, IN DOUBLE PRECISION
            DO 40 I=2,NM1
               R(I)=DBLE(B(I-1))+DBLE(B(I+1))-2.D0*DBLE(B(I))
 40         CONTINUE
            R(1)=X+B(2)-DBLE(1.0+X)*DBLE(B(1))
            R(N)=X+B(N-1)-DBLE(1.+X)*DBLE(B(N))
C SOLVE A(DELTAX)=R
            CALL BPFS(N,ML,G,IG,R,N,1)
            CALL BPBS(N,ML,G,IG,R,N,1)
C DETERMINE NORM OF CORRECTION AND ADD IN CORRECTION
            RNORM=0.0
```

```
            DO 50 I=1,N
               B(I)=B(I)+R(I)
               RNORM=RNORM+ABS(R(I))
 50         CONTINUE
            IF(RNORM.LT.R1MACH(4)*BNORM) GO TO 70
 60      CONTINUE
         WRITE(IWRITE,61)
 61      FORMAT(18H REFINEMENT FAILED)
C COMPUTE NEW ERROR
 70      ERR=0.0
         DO 80 I=1,N
            ERR=AMAX1(ERR,ABS(B(I)-1.0))
 80      CONTINUE
         WRITE(IWRITE,81)IT,ERR
 81      FORMAT(24H ERROR AFTER REFINEMENT ,I4,3H IS,E14.7)
         X=X/100.0
 90   CONTINUE
      STOP
      END
```

When the above program was executed on the Honeywell 6000 at Bell Laboratories, the following was printed

```
X IS      1.00000000
FOR BPLE THE ERROR IS      0.00000431
ERROR AFTER REFINEMENT    2 IS 0.

X IS      0.01000000
FOR BPLE THE ERROR IS      0.00002055
ERROR AFTER REFINEMENT    3 IS 0.

X IS      0.00010000
FOR BPLE THE ERROR IS      0.00491761
ERROR AFTER REFINEMENT    4 IS 0.
```

This problem was chosen because as x approaches 0, the matrix becomes more ill-conditioned, the error in the solution grows, and more steps of iterative refinement are required. The reader should be aware that in this example we were able to represent the matrix and the right-hand side precisely, but due to roundoff error this is not always the case. Often the iterative refinement algorithm produces an exact, but useless, solution to a slightly incorrect problem.

BPLD — LDL^T decomposition of a band symmetric positive definite matrix

Purpose: BPLD (Band Positive definite LDL^T decomposition) decomposes a banded symmetric positive definite matrix A into LDL^T where L is lower triangular and D is diagonal. It allows the user to provide a threshold for considering a matrix singular BPLD is called by the decomposition routines BPCE and BPDC.

Usage: CALL BPLD (N, MU, G, IG, EPS)

N $\longrightarrow$ the number of equations

MU $\longrightarrow$ the number of nonzero bands on and above the diagonal of A

G $\longrightarrow$ a matrix into which the upper triangular portion of the matrix A has been packed as follows:

$$G(j-i+1, i) = a_{ij} \text{ for } j \geq i$$

(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.

$\longleftarrow$ the LDL^T decomposition suitable for input into BPFS and BPBS (see **Note 2**)

IG $\longrightarrow$ the row (leading) dimension of G, as dimensioned in the calling program

EPS $\longrightarrow$ if there exists an index *k* such that $| d_{kk} | \leq$ EPS then A is considered singular

Note 1: After the execution of BPLD, (if the matrix has not been found singular), the determinant of A is the product of the elements of the first row of G.

Note 2: The LDL^T decomposition of A satisfies the equation $A = LDL^T$ where L is lower unit triangular (1's on the diagonal, 0's above the diagonal) and D is diagonal. On return from BPLD, the diagonal of D occupies the first row of G and $G(i-j+1,i) = l_{ij}$ for i>j.

Note 3: For complex Hermitian matrices (A = A*, where A* represents the conjugate transpose of A), the complex version of this subroutine computes the LDL* decomposition and returns the conjugate of L rather than L in G.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU
10 + k*	singular matrix whose rank is at least k
10 + N + k*	the k^{th} principal minor is not positive definite

Double-precision version: DBPLD with G and EPS declared double precision

Complex Hermitian version: CBPLD with G declared complex (see **Note 3** above)

Storage: None

Timing: N×(MU−1)×MU/2 additions
N×(MU−1)×MU/2 multiplications
(MU−1)×N divisions

Method: Gaussian elimination without pivoting

See also: BPBS, BPCE, BPDC, BPFS, BPLE, BPSS

Author: Linda Kaufman

Example: As noted above, after execution of BPLD, the determinant of the matrix stored in G is the product of the elements stored in the first row of G. The subroutine computes this product taking care to avoid underflow and overflow. The subroutine UMKFL is used to decompose a floating-point number, F, into a mantissa, M, and an exponent E such that $F = Mb^E$ where b is the base of the machine and $1/b \leq |M| \leq 1$

```
      SUBROUTINE BPDET(N,MU,G,IG,DETMAN,IDETEX)
C
C THIS SUBROUTINE COMPUTES THE DETERMINANT OF A
C BAND SYMMETRIC POSITIVE DEFINITE MATRIX STORED IN G.
C IT IS GIVEN BY DETMAN*BETA**IDETEX
C WHERE BETA IS THE BASE OF THE MACHINE
C AND DETMAN IS BETWEEN 1/BETA AND 1
C
      REAL G(IG,N),DETMAN
      REAL ONOVBE,M
      INTEGER E
      INTEGER IDETEX
      CALL BPLD(N,MU,G,IG,0.0)
C
C THE DETERMINANT IS THE PRODUCT OF THE ELEMENTS OF ROW 1 OF G
C WE TRY TO COMPUTE THIS PRODUCT IN A WAY THAT WILL
C AVOID UNDERFLOW AND OVERFLOW
C
      ONOVBE=1.0/FLOAT(I1MACH(10))
      DETMAN=ONOVBE
      BETA=FLOAT(I1MACH(10))
      IDETEX=1
      DO 10 I=1,N
          CALL UMKFL(G(1,I),E,M)
          DETMAN=DETMAN*M
          IDETEX=IDETEX+E
          IF(DETMAN.GE.ONOVBE) GO TO 10
             IDETEX=IDETEX-1
             DETMAN=DETMAN*BETA
 10   CONTINUE
      RETURN
      END
```

BPLE — band symmetric positive definite linear system solution

Purpose: BPLE (Banded symmetric Positive definite Linear Equation solution) solves the system AX = B where A is a banded symmetric positive definite matrix

Usage: CALL BPLE (N, MU, G, IG, B, IB, NB)

N → the number of equations

MU → the number of nonzero bands on and below the diagonal of A

G → a matrix into which the upper triangular portion of the matrix A has been packed as follows:

$$G(j-i+1,i) = a_{ij} \text{ for } j \geq i$$

(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.

← L and D from the factorization of A into LDL^T (see **Note 3**)

IG → the row (leading) dimension of G, as dimensioned in the calling program

B → the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB.

← the solution X

IB → the row (leading) dimension of B, as dimensioned in the calling program

NB → the number of right-hand sides

Note 1: Unless the given matrix A is known in advance to be well-conditioned, the user should use BPSS instead of BPLE.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should call subprograms BPLE, BPFS and BPBS. (See the example in BPFS.) BPLE is called once to get the LDL^T decomposition (see the introduction to this chapter) and to solve for the first solution and then the pair, BPFS (forward solve) and BPBS (back solve), is called for each additional right-hand side.

Note 3: The LDL^T decomposition of A satisfies the equation $A = LDL^T$ where L is lower unit triangular (1's on the diagonal, 0's above the diagonal) and D is diagonal. On return from BPLE, the diagonal of D occupies the first row of G and $G(i-j+1,i) = l_{ij}$ for i>j.

Note 4: For complex Hermitian matrices (A = A*, where A* represents the conjugate transpose of A), the complex version of this subroutine computes the LDL* decomposition and returns the conjugate of L rather than L in G.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU
4	IB < N
5	NB < 1
10 + k*	singular matrix whose rank is at least k
10 + N + k*	k^{th} principal minor is not positive definite

Double-precision version: DBPLE with G and B declared double precision

Complex Hermitian version: CBPLE with G and B declared complex (see **Note 4**).

Storage: None

Time: N×((MU−1)×(MU/2+2×(NB+1))+1) additions
N×(MU−1)×(MU/2+2×NB) multiplications
N×(MU−1+NB) divisions

Method: BPLE calls BPDC to form the LDL^T decomposition, and then calls BPFS for the forward solution, and BPBS for the back solution.

See also: BPBS, BPCE, BPDC, BPFS, BPLD, BPSS

Author: Linda Kaufman

Example: The matrix in this example is derived from the usual five-point approximation to the Laplace operator on the unit square with an 11×11 mesh. The 100×100 matrix A has the form

$$\begin{array}{cccccccc} C & -I & & & & & & \\ -I & C & -I & & & & & \\ & -I & C & -I & & & & \\ & & \cdots\cdots & & & & & \\ & & & -I & C & -I & & \\ & & & & -I & C & & \end{array}$$

where I is the identity matrix of order 10 and C is the matrix

$$\begin{array}{cccccccc} 4 & -1 & & & & & & \\ -1 & 4 & -1 & & & & & \\ & -1 & 4 & -1 & & & & \\ & & & \cdots & & & & \\ & & & -1 & 4 & -1 & & \\ & & & & -1 & 4 & & \end{array}$$

To make it easy to detect errors in the example, the right-hand side has been chosen to make the solution a vector of all 1's. To construct the right-hand side the subroutine BPML is used which produces b=Ax where A is a band symmetric positive definite matrix packed into the matrix G.

```
            INTEGER IG, N, MU, MLM1, I, KBLOK, KK, J
            INTEGER IWRITE, I1MACH
            REAL G(11,100), B(100), X(100)
            REAL ERR, AMAX1
            IG=11
            N=100
            MU=11
C
C SET UP MATRIX FOR ELLIPTIC PDE IN 2 DIMENSIONS
C
            MLM1=MU-1
            I=0
            DO 30 KBLOK=1,MLM1
               DO 20 KK=1,MLM1
                  I=I+1
                  G(1,I)=4.0
                  G(2,I)=-1.0
                  DO 10 J=3,MLM1
                     G(J,I)=0.0
   10             CONTINUE
```

```
                  G(MU,I)=-1.0
   20          CONTINUE
               G(2,I)=0.0
   30       CONTINUE
C
C SET UP RIGHT HAND SIDE SO SOLUTION IS ALL 1'S
C
            DO 40 I=1,N
               X(I)=1.0
  40        CONTINUE
            CALL BPML(N,MU,G,IG,X,B)
C
C SOLVE THE SYSTEM
C
            CALL BPLE(N,MU,G,IG,B,100,1)
C
C COMPUTE THE ERROR
C
            ERR=0.0
            DO 50 I=1,N
               ERR=AMAX1(ERR,ABS(B(I)-1.0))
  50        CONTINUE
            IWRITE=I1MACH(2)
            WRITE(IWRITE,51)ERR
  51        FORMAT(31H ERROR IN SOLUTION FROM BPLE IS,F15.8)
            STOP
            END
```

When the above code was run on the Honeywell 6000 machine at Bell Laboratories, the following was printed:

```
 ERROR IN SOLUTION FROM BPLE IS     0.00000012
```

BPML — banded positive definite matrix - vector multiplication

Purpose: BPML (Banded Positive definite matrix MuLtiplication) forms the product *Ax* where *A* is a symmetric banded positive matrix stored in packed form.

Usage: CALL BPML (N, MU, G, IG, X, B)

N → the length of *x*

MU → the number of nonzero bands on and above the diagonal of A

G → a matrix into which the upper triangular portion of the matrix A has been packed as follows:

$$G(1 + j-i,i) = a_{ij} \text{ for } j \geq i.$$

(See the introduction to this chapter.) G should be dimensioned (IG, KG) in the calling program, where IG≥MU and KG≥N.

IG → the row (leading) dimension of G, as dimensioned in the calling program

X → the vector *x* to be multiplied

B ← the vector A*x*

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU

Double-precision version: DBPML with G, X, and B declared double precision.

Complex Hermitian version: CBPML with G, X, and B declared complex

Storage: None

Time: (2MU−1)×N additions
(2MU−1)×N multiplications

See also: BPBS, BPCE, BPDC, BPLU, BPLE, BPSS

Author: Linda Kaufman

Example: This example checks the consistency of BPML and BPSS the banded system solver. First the example uses BPML to compute for a given vector x and a given matrix A the vector $b = Ax$. Then the problem is inverted, i.e., BPSS is used to find the vector x which satisfies $Ax = b$. This x is then compared with the original vector. The vector x is generated randomly and the 10×10 matrix A is given by

$$\begin{bmatrix} 4 & -1 & -1 & & & & & & \\ -1 & 4 & -1 & -1 & & & & & \\ -1 & -1 & 4 & -1 & -1 & & & & \\ & -1 & -1 & 4 & -1 & -1 & & & \\ & & . & . & . & & & & \\ & & & -1 & -1 & 4 & -1 & -1 & \\ & & & & -1 & -1 & 4 & -1 & \\ & & & & & -1 & -1 & 4 & \end{bmatrix}$$

```
      INTEGER IG, N, MU, I, IWRITE, I1MACH
      REAL G(3,20), X(20), B(20)
      REAL UNI, ERR, COND, SASUM, ABS
      IG=3
      N=10
      MU=3
C
C CONSTRUCT MATRIX A AND PACK IT INTO G
C
       DO 10 I=1,N
          G(1,I)=4.0
          G(2,I)=-1.0
          G(3,I)=-1.0
 10    CONTINUE
C
C CONSTRUCT A RANDOM VECTOR
C
       DO 20 I=1,N
          X(I)=UNI(0)
 20    CONTINUE
C
C CONSTRUCT B=AX
```

```
C
         CALL BPML(N,MU,G,IG,X,B)
C
C SOLVE THE SYSTEM AX=B
C
         CALL BPSS(N,MU,G,IG,B,N,1,COND)
C
C PRINT OUT THE TRUE SOLUTION AND THE COMPUTED SOLUTION
C
         IWRITE=I1MACH(2)
         WRITE(IWRITE,21)
  21     FORMAT(34H TRUE SOLUTION   COMPUTED SOLUTION)
         WRITE(IWRITE,22)(X(I),B(I),I=1,N)
  22     FORMAT(1H ,2E16.8)
         ERR=0.0
         DO 30 I=1,N
             ERR=ERR+ABS(B(I)-X(I))
  30     CONTINUE
         ERR=ERR/SASUM(N,X,1)
         WRITE(IWRITE,31)ERR
  31     FORMAT(19H RELATIVE ERROR IS ,1PE15.7)
         WRITE(IWRITE,32)COND
  32     FORMAT(20H CONDITION NUMBER IS,1PE15.7)
         STOP
         END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, which has a machine precision of $1.\times10^{-8}$, the following was printed:

```
TRUE SOLUTION   COMPUTED SOLUTION
  0.22925607E 00  0.22925608E 00
  0.76687502E 00  0.76687504E 00
  0.68317685E 00  0.68317687E 00
  0.50919111E 00  0.50919112E 00
  0.87455959E 00  0.87455962E 00
  0.64464101E 00  0.64464103E 00
  0.84746840E 00  0.84746842E 00
  0.35396343E 00  0.35396345E 00
  0.39889160E 00  0.39889160E 00
  0.45709422E 00  0.45709422E 00
RELATIVE ERROR IS   3.1985797E-08
CONDITION NUMBER IS  2.1901962E 01
```

The condition number of the matrix and the precision of the Honeywell suggest that even in the absence of roundoff error in BPML, a relative error of 2.2×10^{-7} would not be surprising. The value computed above is quite reasonable.

BPNM — norm of a banded symmetric positive definite matrix

Purpose: BPNM (Banded Positive definite matrix NorM) computes the norm of a banded symmetric positive definite matrix A stored in packed form. The infinity norm is defined as $\max_{1 \le i \le n} \sum_{j=1}^{n} | a_{ij} |$

Type: Real function

Usage: <answer> = BPNM (N, MU, G, IG)

N → the number of rows in A

MU → the number of nonzero bands in A on and above the diagonal

G → a matrix into which the upper triangular portion of the matrix A has been packed as follows:

$$G(1 + j - i, i) = a_{ij} \text{ for } j \ge i$$

i.e. the main diagonal of A is in the first row of G
(See the introduction to this chapter.)
G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.

IG → the row (leading) dimension of G, as dimensioned in the calling program

<answer> ← $\max_{1 \le i \le n} \sum_{j=1}^{n} | a_{ij} |$

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU

Double-precision version: DBPNM with G and DBPNM declared double precision

Complex version: CBPNM with G declared complex

Storage: None

Time: $N \times (2 \times MU - 1)$ additions

See also: BPDC, BPLD, BPLE, BPSS, BPCE

Author: Linda Kaufman

Example: Because of roundoff error it is often very difficult to decide whether a matrix is singular. One criteria often used for symmetric positive definite matrices is to compute the LDL^T decomposition of the matrix and declare the matrix singular if any element of the diagonal matrix D is less than $\varepsilon ||A||$ where ε is the machine precision and is computed by R1MACH(4).

The following program fragment might be used to indicate whether the symmetric positive definite banded matrix packed into G is singular or nearly singular. It uses the fact that the subroutine BPLD, which computes the LDL^T decomposition of a banded symmetric matrix, issues a recoverable error when it detects an element of D less than EPS, an input parameter to the subroutine.

```
      IWRITE=I1MACH(2)
      CALL ENTSRC(IROLD,1)
      EPS=BPNM(N,MU,G,IG)*R1MACH(4)
      CALL BPLD(N,MU,G,IG,EPS)
      IF (NERROR(IERR).EQ.0) GO TO 10
         CALL ERROFF
         WRITE(IWRITE,1)
1        FORMAT(16H SINGULAR MATRIX)
10    CONTINUE
```

BPSS — band positive definite linear system solution with condition estimation

Purpose: BPSS (Band Positive definite System Solution) solves the system AX = B where A is a band symmetric positive definite matrix. An estimate of the condition number of A is provided.

Usage: CALL BPSS (N, MU, G, IG, B, IB, NB, COND)

N	→	the number of equations
MU	→	the number of nonzero bands on and above the diagonal of A
G	→	a matrix into which the upper triangular portion of the matrix A has been placed as follows: $G(j-i+1, i) = a_{ij}$ for $j \geq i$ (See the introduction to this chapter.) G should be dimensioned (IG,KG) in the calling program, where IG≥MU and KG≥N.
	←	L and D from the factorization of A into LDL^T (see **Note 3**)
IG	→	the row (leading) dimension of G, as dimensioned in the calling program
B	→	the matrix of right-hand sides, dimensioned (IB, KB) in the calling program, where IB≥N and KB≥NB.
	←	the solution X
IB	→	the row (leading) dimension of B, as dimensioned in the calling program
NB	→	the number of right-hand sides
COND	←	an estimate of the condition number of A (See **Note 1**)

Note 1: The condition number measures the sensitivity of the solution of a linear system to errors in the matrix and in the right-hand side. If the elements of the matrix and the right-hand side(s) of your linear system have **d** decimal digits of precision, the solution might have as few as $\mathbf{d} - \log_{10}(\text{COND})$ correct decimal digits. Thus if COND is greater than $10^{\text{Bd}P}$, there may be no correct digits.

If the given matrix, A, is known in advance to be well-conditioned, then the user may wish to

use the routine BPLE, which is a little faster than BPSS. Ordinarily, however, the user is strongly urged to choose BPSS, and to follow it by a test of the condition estimate.

Note 2: Users who wish to solve a sequence of problems with the same coefficient matrix, but different right-hand sides *not all known in advance*, should not use BPSS, but should call subprograms BPCE, BPFS and BPBS. (See the example of BPFS.) BPCE is called once to get the LDL^T decomposition (see the introduction to this chapter) and then the pair, BPFS (forward solve) and BPBS (back solve), is called for each new right-hand side.

Note 3: The LDL^T decomposition of A satisfies the equation $A = LDL^T$ where L is lower unit triangular (1's on the diagonal, 0's above the diagonal) and D is diagonal. On return from BPSS, the diagonal of D occupies the first row of G and $G(i-j+1,i) = l_{ij}$ for i>j.

Note 4: For complex Hermitian matrices (A = A*, where A* represents the conjugate transpose of A), the complex version of this subroutine computes the LDL* decomposition and returns the conjugate of L rather than L in G.

Error situations: *(The user can elect to 'recover' from those errors marked with an asterisk — see *Error Handling*, Framework Chapter)

Number	Error
1	N < 1
2	MU < 1
3	IG < MU
4	IB < N
5	NB < 1
10 + k*	singular matrix whose rank is at least k
10 + N + k*	k^{th} principal minor is not positive definite

Double-precision version: DBPSS, with G, B, and COND declared double precision.

Complex Hermitian version: CBPSS with G and B declared complex (see **Note 4** above)

Storage: N real (double precision for DBPSS, complex for CBPSS) locations of scratch storage in the dynamic storage stack

Time: N×((MU−1)×(MU/2+2×NB+8)+8) additions
N×((MU−1)×(MU/2+2×NB+6)+4) multiplications
N×(MU+1+NB) divisions

Method: BPSS calls BPCE to form the LDL^T decomposition, and then calls BPFS for the forward solution, and BPBS for the back solution. See the reference below for the method used to estimate the condition number.

See also: BPBS, BPCE, BPDC, BPFS, BPLD, BPLE

Author: Linda Kaufman

Reference: Cline, A. K., Moler, C. B., Stewart, G. W., and Wilkinson, J. H., An estimate for the condition number, *SIAM J. Numer. Anal. 16* (1979), 368-375.

Example: In the following example we solve three problems that might arise from a discretization of a 1-dimensional differential equation. The coefficient matrix in each problem has the form

$$\begin{bmatrix} 1+x & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & -1 & 2 & -1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1+x \end{bmatrix}$$

with x=1.0, .01, and .0001 defining the three problems. The matrix is singular when x is 0, and, as our output suggests, the matrix becomes more ill-conditioned as x approaches 0. To make it easy to detect errors in the solution, the right-hand side has been chosen to make the solution a vector of all 1's. The reader should notice in the output the correlation between the condition number and the error. As with most problems with nearly constant diagonal, it was very easy to write the code to set up the problem for BPSS.

```
      INTEGER N, K, I, IWRITE, I1MACH, MU
      REAL G(2,100), B(200)
      REAL X, COND, ERR, AMAX1
C CONSTRUCT MATRIX AND RIGHT-HAND SIDE SO TRUE SOLUTION IS
```

```
C COMPOSED ENTIRELY OF ONES
      N=100
      X=1
      DO 30  K=1,3
         DO 10 I=1,N
            G(1,I)=2.0
            G(2,I)=-1.0
            B(I)=0.0
 10      CONTINUE
         G(1,1)=1.0+X
         G(1,N)=1.0+X
         B(1)=X
         B(N)=X
C SOLVE THE SYSTEM
         MU=2
         CALL BPSS(N,MU,G,2,B,N,1,COND)
         IWRITE=I1MACH(2)
         WRITE(IWRITE,11)X
 11      FORMAT(/5H X IS,F15.7)
         WRITE(IWRITE,12)COND
 12      FORMAT(20H CONDITION NUMBER IS,1PE15.7)
C COMPUTE THE ERROR
         ERR=0.0
         DO 20 I=1,N
            ERR=AMAX1(ERR,ABS(B(I)-1.0))
 20      CONTINUE
         WRITE(IWRITE,21)ERR
 21      FORMAT(22H FOR BPSS THE ERROR IS,F16.8)
         X=X/100.
 30   CONTINUE
      STOP
      END
```

When the above program was executed on the Honeywell 6000 machine at Bell Laboratories, the following was printed

```
X IS      1.0000000
CONDITION NUMBER IS  4.0807862E 03
FOR BPSS THE ERROR IS      0.00000431

X IS      0.0100000
CONDITION NUMBER IS  2.3329148E 04
FOR BPSS THE ERROR IS      0.00002055

X IS      0.0001000
CONDITION NUMBER IS  1.9933923E 06
FOR BPSS THE ERROR IS      0.00491761
```

BASIC LINEAR ALGEBRA MODULES

ISAMAX	-	index if the largest element of a vector
SASUM	-	1-norm of a vector
SAXPY	-	add multiple of one vector to another
SDOT	-	dot product of two vectors
SSCAL	-	scale a vector
SSWAP	-	interchange two vectors

ISAMAX — index of the maximum magnitude element of a vector

Purpose: ISAMAX looks through a vector to find the (first) component with maximum magnitude. The integer position of that component in the vector is returned.

Type: Integer function

Usage: <answer> = ISAMAX(N, X, INCX)

N	$\longrightarrow$	the number of elements to be compared
X	$\longrightarrow$	the vector of elements
INCX	$\longrightarrow$	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N−1)∗INCX)
<answer>	$\longleftarrow$	M, the position of the (first) component of maximum magnitude: If INCX = 1, $\lvert X(M)\rvert$ is largest. In general, $\lvert X(1 + (M - 1) * INCX)\rvert$ is largest.

Note 1: Since Fortran stores arrays in column-wise order we can use ISAMAX to deal with the rows of a 2-dimensional array as in the example below.

Note 2: If N=0, ISAMAX=0 is returned.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	$N < 0$
2	$INCX \leq 0$

Double-precision version: IDAMAX with X declared double precision

Complex version: ICAMAX with X declared complex

See also: SAMAX, ISMAX

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software 5*, 3 (1979), 308-323.

Example: In this example the columns of an $m times n$ matrix A, dimensioned (IA,N) are permuted so that the (1,1) element of A is the largest in modulus of the elements of the first row in A. The subroutine SSWAP interchanges two vectors.

```
J=ISAMAX(N,A,IA)
CALL SSWAP(M,A,1,A(1,J),1)
```

SASUM — 1-norm of a vector

Purpose: SASUM computes the sum of the absolute values of a vector: $sum from i=1 to n ⎪ x sub i ^ ⎪$

Type: Real function

Usage: <answer> = SASUM(N, X, INCX)

N	→	the number of elements to be summed
X	→	the vector of elements
INCX	→	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N–1)INCX)
<answer>	←	⎪ X(1) ⎪ + ⎪ X(1+INCX) ⎪ + ... + ⎪ X(1+(N–1)INCX) ⎪

Note 1: Since Fortran stores arrays in column-wise order we can use SASUM to deal with the rows of a 2-dimensional array as in the example below.

Note 2: If N=0, SASUM=0.0 is returned.

Note 3: For complex vectors, SCASUM computes $sum from i=1 to n (⎪Re(x sub i)⎪+⎪Im(x sub i)⎪)$.

Note 4: No attempt is made to prevent or give warning of underflow or overflow.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N<0
2	INCX≤0

Double-precision version: DASUM with X declared double precision

Complex version: SCASUM with X declared complex (see Note 3).

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran Usage, Report SAND77-0898, Sandia Laboratories, Albuquerque, New Mexico 87115, October 1977.

Examples: The following program fragment computes the 1-norm of an $m times n$ matrix A. The 1-norm is defined by $max from 1<=j<=n sum from i=1 to m | a sub i,j^^|$

```
      ANORM1=0.0
      DO 10 J=1,N
         ANORMJ = SASUM(M,A(1,J),1)
         IF (ANORMJ .GT. ANORM1) ANORM1 = ANORMJ
   10 CONTINUE
```

The next program fragment computes the infinity norm of an $m times n$ matrix A, dimensioned (IA,N). The infinity norm is defined as $max from 1<=i<=m sum from j=1 to n | a sub i,j^^|$

```
      ANORM=0.0
      DO 10 I=1,M
         ANORMI = SASUM(N,A(I,1),IA)
         IF (ANORMI .GT. ANORM) ANORM=ANORMI
    10 CONTINUE
```

SAXPY — add multiple of one vector to another

Purpose: SAXPY scales a vector x by a scalar a and adds the result to a vector y.

Usage: CALL SAXPY (N, A, X, INCX, Y, INCY)

N	→	the number of affected elements in X and Y
A	→	the scalar variable
X	→	the vector which is to be scaled
INCX	→	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N−1)∗INCX)
Y	→	the vector which is to be added
	←	AX + Y
INCY	→	the elements are space at intervals of INCY in Y: Y(1), Y(1+INCY), ..., Y(1+(N−1)∗INCY)

Note: If N=0, no action is performed.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N<0
2	INCX≤0
3	INCY≤0

Double-precision version: DAXPY with X and Y declared double precision

Complex version: CAXPY with X and Y declared complex

See also: SSCAL

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software 5*, 3 (1979), 308-323.

Example: The following program fragment forms the product Ax where A is an $m times n$ matrix and puts the result in an array Y:

```
   DO 10 I=1,M
      Y(I)=0.0
10 CONTINUE
   DO 20 I=1,N
      CALL SAXPY(M,X(I),A(1,I),1,Y,1)
20 CONTINUE
```

Matrix by vector multiplication is usually done using inner products as in the example in SDOT, but on a paged machine using the above program fragment can be preferable because FORTRAN stores two-dimensional arrays column-wise and this program refers to the array A one column at a time.

SDOT — dot product of two vectors

Purpose: SDOT determines the inner product of two vectors x and y, $sum from i=1 to n x sub i y sub i$

Type: Real function

Usage: <answer> = SDOT(N, X, INCX, Y, INCY)

N	→	the number of elements to be summed
X	→	the first vector
INCX	→	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N−1)∗INCX)
Y	→	the second vector
INCY	→	the elements are spaced at intervals of INCY in Y: Y(1), Y(1+INCY), ..., Y(1+(N−1)∗INCY)
<answer>	←	X(1)∗Y(1)+X(1+INCX)∗Y(1+INCY)+...+ X(1+(N−1)∗INCX)∗Y(1+(N−1)∗INCY)

Note 1: Since Fortran stores arrays in column-wise order we can use SDOT to deal with the rows of a 2-dimensional array as in the example below.

Note 2: If N=0, SDOT=0.0 is returned.

Note 3: No attempt is made to prevent underflow or overflow in the subroutine.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N<0
2	INCX≤0
3	INCY≤0

Double-precision version: DDOT with X and Y declared double precision

Complex versions: CDOTU with X and Y declared complex. CDOTC with X and Y declared complex. CDOTC $= sum from i=1 to n x bar sub i y sub i$, i.e. the conjugate of the elements of X are used.

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software 5*, 3 (1979), 308-323.

Example: The following program fragment forms the product $A bold x$ where A is an $m times n$ matrix dimensioned (IA,N), and puts the result in an array Y:

```
      DO 10 I=1,M
         Y(I)=SDOT(N,A(I,1),IA,X,1)
   10 CONTINUE
```

Because of page faults, the execution of this program fragment on certain machines might require an excessive amount of time. The program fragment given in the example in SAXPY, which accesses the elements of A one column at a time, would be preferable in this case.

SSCAL — scale a vector

Purpose: SSCAL multiplies a vector x by a scalar A

Usage: CALL SSCAL(N, A, X, INCX)

N	→	the number of affected elements in X
A	→	the scaling factor
X	→	the vector to be scaled
	←	the scaled vector
INCX	→	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N−1)∗INCX)

Note 1: If N=0, no action is performed.

Note 2: Since Fortran stores arrays in column-wise order, we can use SSCAL to deal with the rows of a 2-dimensional array as in the example below.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N<0
2	INCX≤0

Double-precision version: DSCAL with X and A declared double precision

Complex versions: CSCAL with X and A declared complex
CSSCAL with X declared complex and A declared real

See also: SAXPY

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software 5*, 3 (1979), 308-323.

Example: In this example the rows of an $m times n$ matrix A, dimensioned (IA,N) are scaled so that the sum of the modulus of the elements in each row is 1.0. The function SASUM returns the sum of the absolute values of the elements of a vector.

```
      DO 10 J=1,M
         SC=1.0/SASUM(N,A(J,1),IA)
         CALL SSCAL(N,SC,A(J,1),IA)
10 CONTINUE
```

SSWAP — interchange two vectors

Purpose: SSWAP interchanges two vectors

Usage: CALL SSWAP(N, X, INCX, Y, INCY)

N	→	the number of affected elements in X and Y
X	→	the first vector
	←	the vector Y
INCX	→	the elements are spaced at intervals of INCX in X: X(1), X(1+INCX), ..., X(1+(N−1)INCX)
Y	→	the second vector
	←	the vector X
INCY	→	the elements are spaced at intervals of INCY in Y: Y(1), Y(1+INCY), ..., Y(1+(N−1)*INCY)

Note 1: If N=0, no action is performed.

Note 2: Since Fortran stores arrays in column-wise order, we can use SSWAP to deal with the rows of a 2-dimensional array as in the example below.

Error situations: (All errors in this subprogram are fatal — see *Error Handling*, Framework Chapter)

Number	Error
1	N<0
2	INCX≤0
3	INCY≤0

Double-precision version: DSWAP with X and Y declared double precision

Complex version: CSWAP with X and Y declared complex

See also: MOVExx (Utility Chapter)

Author: Linda Kaufman

Reference: Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T., Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software 5*, 3 (1979), 308-323.

Example: In this example the rows of an $m times n$ matrix A, dimensioned (IA,N) are permuted so that the (1,1) element of A is the largest in modulus of the elements in the first column of A. The subroutine ISAMAX computes the index of the element of a vector having maximum modulus.

```
J=ISAMAX(M,A,1)
CALL SSWAP(N,A,IA,A(J,1),IA)
```