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

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

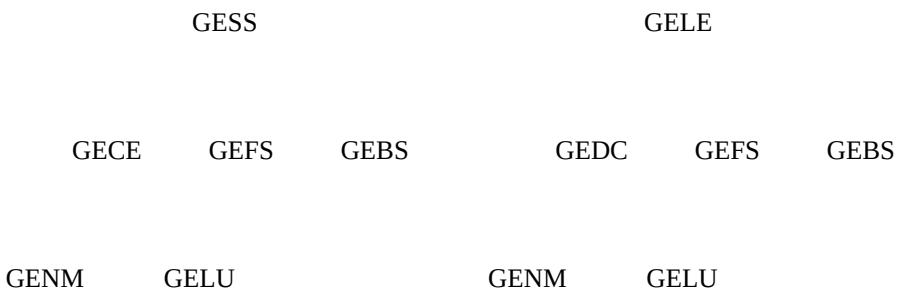
and

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

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

1	3	5	7
3	2	4	6
5	4	8	10
7	6	10	9

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
```

Fig. 1

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

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.