A COMPARISON OF ALGORITHMS FOR SOLVING SYMMETRIC INDEFINITE SYSTEMS OF LINEAR EQUATIONS

by

Victor Barwell

and

Alan George

Research Report CS-75-05

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

February, 1975

Research supported in part by Canadian National Research Council Grant A8111 and a Canadian National Research Council Science Scholarship.

1. Introduction

Consider the problem of solving the dense $n \times n$ linear algebraic system

(1.1) $Ax = b$.

When A has no special characteristics other than being non-singular, the usual procedure is to apply Gaussian elimination with partial pivoting, yielding the factorization

(1.2) $PA = LU$,

where L is unit lower triangular, U is upper triangular, and P is a permutation matrix reflecting the interchanges performed [5]. The solution x is then obtained by solving the two triangular systems $Ly = Pb$ and $Ux = y$. The number of multiplications and divisions required to solve (1.1) by this procedure is $n^3/3 + O(n^2)$, and the storage required is $n^2 + O(n)$, assuming A is overwritten by L and U.

In many important applications A is symmetric and positive definite. In this case it is well known that Cholesky's method may be applied, yielding the factorization

(1.3) $A = LL^T$,

where L is lower triangular. This method is stable and exploits symmetry: the number of multiplications and divisions required is only $n^3/6 + O(n^2)$, and since only the lower triangle of A is needed, which can be overwritten by L, only $n(n+1)/2$ storage locations are required. The closely related methods of congruence transformations (Westlake [13]) or symmetric Gaussian elimination share the above advantages. The essential difference is that the factorization of A obtained is of the form

(1.4) $A = \tilde{L}D\tilde{L}^T$,

where $\tilde{L}$ is unit lower triangular and D is a positive diagonal matrix.

Unfortunately, when A is symmetric but not definite, these schemes which exploit symmetry can be numerically unstable or fail. (See [4, pp.643-648] for an excellent discussion.) In recent years a number of schemes which share some or all of the advantages of Cholesky's method have been proposed for solving symmetric indefinite systems [1,2,9]. The aim of this paper is to compare some Fortran implementations of these and other methods for solving symmetric indefinite systems.

In Section 2, we review the basic idea of the methods to be compared, and discuss any important features of their implementation. Section 3 describes our experiments and results of the experiments, which were performed on two different computers, an IBM 360/75 and a Honeywell 6050. Finally, Section 4 contains our conclusions. A listing of the codes used is in the appendix.

2. A Brief Description of the Methods to be Compared

1) Parlett and Reid's Tri-diagonal Method [9]

The basic idea is to reduce A to a symmetric tri-diagonal matrix T using stabilized elementary congruence transformations, and then to solve the tri-diagonal system using Gaussian elimination with partial pivoting. The loss of symmetry during the latter stage is of little consequence. The first step of the algorithm (ignoring interchanges), can be expressed as

$$(2.1) \qquad A = \begin{pmatrix} a & b & u^T \\ b & c & v^T \\ u & v & B \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & O \\ & 1 & \\ 0 & \frac{u}{b} & I_{n-2} \end{pmatrix} \begin{pmatrix} a & b & 0 \\ b & c & v^T - \frac{c}{b}u \\ 0 & v - \frac{c}{b}u & B - \frac{1}{b}(uv^T + vu^T) + \frac{c}{b^2}uu^T \end{pmatrix} \begin{pmatrix} 1 & & 0 \\ & 1 & u^T/b \\ O & & I_{n-2} \end{pmatrix}$$

$$= \quad L_1 A_1 L_1^T \ .$$

The basic step is then recursively applied to A_1 yielding A_2 and so on, finally obtaining a tri-diagonal matrix $A_{n-2} = T$. At the k-th step, A_k is symmetrically row and column permuted so that the largest component in magnitude in column k is in position (k+1,k). The method thus displays essentially the same stability as Gaussian elimination with partial pivoting. With regard to implementation, it is convenient to store the components L_{ik}, $i > k$, of column k of L in positions (i,k-1), since column one of L is zero below the diagonal. This leaves space for the diagonal and subdiagonal of T, and the factorization can be carried out "in place". Our code stores A in "symmetric storage mode" [8]; the lower triangle of A including the diagonal is stored row by row in a one dimensional array of length $\geq n(n+1)/2$. The code we used is our own implementation of the algorithm.

Thus, the steps of the procedure are as follows:

(2.2a) Apply the tri-diagonalization algorithm to obtain the factorization $PAP^T = LTL^T$, where P is a permutation matrix reflecting the interchanges made at each step of the algorithm.

(2.2b) Solve $Ly = Pb$

(2.2c) Solve the tri-diagonal system $Tz = y$.

(2.2d) Solve $L^Tw = z$.

(2.2e) Compute $x = P^Ty$.

The storage requirement for this method is $n^2/2 + O(n)$, but the number of multiplications and divisions is $n^3/3 + O(n^2)$. Thus, the main attraction this method has over ordinary Gaussian elimination is its reduced storage requirements.

2) Aasen's Method [1]

The algorithm is in principle identical to that of Parlett and Reid, except for an ingenious reordering of the tri-diagonalization calculation which allows further exploitation of symmetry and structure. The number of multiplications and divisions is now only $n^3/6 + O(n^2)$, and the storage requirement remains $n^2/2 + O(n)$.

The key idea is as follows. Ignoring the permutations, define the upper Hessenberg matrix H by

(2.3) $H = TL^T$,

whence

(2.4) $A = LH$.

Now Aasen's observation was that if the first k-1 columns of T and the first k columns of L are known, then the k-th column H_{*k} of H can be computed. That is, use (2.3) to calculate components H_{rk} of H, $1 \le r < k$ from (2.3), which requires $\approx 3k$ multiplications. Then use (2.4) to compute H_{kk}, $H_{k+1,k}$, $T_{i,i}$ and $T_{i+1,i}$, and then compute $H_{r,k}$, $k+1 < r \le n$ using (2.3); these steps require $\approx k(n-k)$ multiplications. With H_{*k} available, we now can compute column k+1 of L and repeat the process with k replaced by k+1. A rough operation count follows easily:

$$\sum_{k=1}^{n-2} (3k+kn-k^2) = \frac{n^3}{6} + O(n^2).$$

Just as in the implementation of Parlett and Reid's algorithm, it is natural to store the columns of L "shifted left" one column position in the storage array. Again, A was stored in symmetric storage mode. Our code is essentially a direct translation of Aasen's Algol 60 code provided in [1], with some minor language-connected modifications.

3) Bunch's Block Diagonal Pivoting Method [2]

This algorithm is similar in spirit to standard symmetric Gaussian elimination, except that both 1 × 1 and 2 × 2 (block) pivots are admitted, the choice depending upon numerical stability considerations. Ignoring permutations, the first step of the algorithm is depicted by the following matrix equation.

$$(2.5)\qquad A = \begin{pmatrix} P & V^T \\ V & B \end{pmatrix}$$

$$= \begin{pmatrix} I_r & \\ VP^{-1} & I_{n-r} \end{pmatrix} \begin{pmatrix} P & 0 \\ 0 & B-VP^{-1}V^T \end{pmatrix} \begin{pmatrix} I_r & P^{-1}V^T \\ & I_{n-r} \end{pmatrix}$$

where r is 1 or 2, and I_q denotes an identity matrix of order q. The basic step is then recursively applied. The algorithm requires $n^3/6 + O(n^2)$ multiplications and divisions, and can be carried out in place, so the storage requirement is $n^2/2 + O(n)$.

Before the k-th step in the factorization, the largest component in magnitude, say a_{rs}, in the $(n-t+1) \times (n-t+1)$ part of the matrix remaining to be factored is permuted via symmetric row and column interchanges into the (t+1,t) position if $r \neq s$, or into position (t,t) if r = s. Thus, before each step, r and s must be determined, which means the algorithm requires (roughly) between $n^3/6$ and $n^3/12$ comparisons. In our implementation, in order to reduce loop and indexing overhead, we determine r and s for step k during the (k-1)st transformation.

In our initial experimentations we used the implementation of Bunch's algorithm LEQT1S provided in the IMSL library [8]. However, it performed so poorly that we used a slightly modified version of George's implementation of Bunch's algorithm appearing in [7, Appendix C]. We have included times for both LEQT1S and our implementation, to show how the careless implementation of an algorithm can severely damage its performance.

4) Householder tri-diagonalization

This scheme is similar in spirit to the first two methods described, except that elementary Hermitian matrices are used to transform A to tri-diagonal form. Thus, we obtain the factorization

$$A = QTQ^T,$$

where T is symmetric tri-diagonal and Q is a product of n-2 elementary Hermitian matrices; if Q is kept in factored form, which is natural for this application,the storage required for the method is $n^2/2 + O(n)$. However, the operation count is $2n^3/3 + O(n^2)$, about four times that required for methods (2) or (3). As a partial compensation, no interchanges are required during the reduction to tri-diagonal form, which reduces the overhead. Also, it is conceivable that on some computers which have very fast floating point arithmetic instructionscompared to the execution time of other instructions, the overhead decrease may more than compensate for the increased arithmetic. Also, on a paged computer,the advantage of not having to permute rows and columns, thus avoiding page faults, could be enormous.

Our code for the Householder transformations is essentially a direct translation of the Algol 60 code Tred3 in [14]. The main modification is to use scaling to prevent overflows, as is done for the Fortran code Tred2 [11].

5) Gaussian elimination with partial pivoting

This algorithm ignores symmetry, and at least on the surface would not appear to be competitive. Nevertheless, it might be recommended for small problems. One reason for including it was to provide a basis for comparison. The method of Parlett and Ried, in theory, should execute about as fast as this algorithm. However, a certain amount of overhead is introduced because only the lower triangle of A is available.

The Gaussian elimination code was kindly provided by M.A. Malcolm and C.B. Moler; this code is to appear in [6].

6) <u>Cholesky decomposition</u>

Again, this code is not strictly relevant to our investigation, but we include execution times for positive definite systems of the same size as the indefinite ones we solved to provide a benchmark. Ideally, we would like a code for solving indefinite symmetric systems to execute as fast and require no more store than our implementation of Cholesky's algorithm.

3. Description of Numerical Experiments and the Results

For each method (code) the table below gives the order of operations (multiplications and divisions), the number of comparisons, and the storage required.

Method	Operations	Comparisons	Storage
Cholesky	$n^3/6$	none	$n^2/2$
Aasen	$n^3/6$	$n^2/2$	$n^2/2$
Bunch	$n^3/6$	$\geq n^3/12, \leq n^3/6$	$n^2/2$
LEQT1S	$n^3/6$	$\geq n^3/12, \leq n^3/6$	$n^2/2$
Parlett & Reid	$n^3/3$	$n^2/2$	$n^2/2$
Gaussian Elimination	$n^3/3$	$n^2/2$	n^2
Householder Transformations	$2n^3/3$	none	$n^2/2$

Description of the Timing Tests

We tested our codes in the following manner. For a given dimension n, we generated a random matrix (a matrix consisting of random numbers between 0 and 1). Assuming the solution was a vector with components all ones, we generated an appropriate right hand side using double precision arithmetic, since all our codes are single precision. The resulting system was solved and the time required to solve it recorded. We repeated this procedure with each code, for the same matrix. We tested matrices from dimension 25 to 200 in steps of 25. To optimize our codes the program was compiled under Fortran H (opt = 2) on an IBM 360/75 and Fortran Y (opt = OPTZ) on a Honeywell 6050. Tables 1 through 5 contain the results for these experiments. Our timing runs were subject to the usual vagaries of modern large scale computing systems, and are accurate only to about .1 seconds. This leads to some minor anomolies in the table entries, but the trends upon which we make our remarks are clear.

TABLE 1 Timings for the IBM 360/75*

Dimension n	Cholesky	Aasen	Parlett & Reid	Bunch	Householder transformation	LEQT1S	Gaussian elimination
25	0.05	0.06	0.05	0.06	0.11	0.16	0.05
50	0.23	0.28	0.32	0.34	0.68	1.08	0.33
75	0.62	0.73	1.01	1.02	2.06	3.43	1.05
100	1.30	1.47	2.30	2.28	4.72	8.39	2.43
125	2.36	2.62	4.38	4.29	8.99	15.25	4.70
150	3.89	4.24	7.54	7.28	15.81	25.87	8.02
175	5.92	6.44	11.79	11.30	24.93	41.09	12.62
200	8.59	9.21	17.46	16.67	37.10	64.74	18.71

TABLE 2 Times for the Honeywell 6050

Dimension n	Cholesky	Aasen	Parlett & Reid	Bunch	Householder transformation	LEQT1S	Gaussian elimination
25	0.13	0.18	0.19	0.24	0.33	0.36	0.16
50	0.66	0.87	1.23	1.44	2.18	2.29	1.02
75	1.90	2.38	3.88	4.40	6.90	7.32	3.20
100	4.15	4.95	8.86	9.96	15.84	15.90	7.34
125	7.69	8.96	16.97	18.91	30.32	30.68	14.05
150	12.82	14.64	28.91	32.04	51.69	57.08	23.96
175	19.86	22.29	45.48	50.21	81.31	89.34	37.70
200	29.04	32.18	67.34	74.16	120.50	131.05	55.89

* All times given in the tables are in seconds.

TABLE 3 Ratio of Honeywell to IBM entries in Tables 1 & 2

Dimension	Cholesky	Aasen	Parlett & Reid	Bunch	Householder transformations	LEQT1S	Gaussian elimination
25	2.60	3.00	3.80	4.00	3.00	2.25	3.20
50	2.87	3.11	3.84	4.24	3.21	2.12	3.09
75	3.06	3.26	3.73	4.31	3.35	2.13	3.05
100	3.19	3.37	3.82	4.37	3.36	1.90	3.02
125	3.26	3.42	3.87	4.41	3.37	2.01	2.99
150	3.30	3.45	3.83	4.40	3.27	2.21	2.99
175	3.35	3.46	3.86	4.44	3.26	2.17	2.99
200	3.38	3.49	3.86	4.45	3.25	2.02	2.99

TABLE 4 Ratio to Cholesky on the IBM 360/75

Dimension	Cholesky	Aasen	Parlett & Reid	Bunch	Householder transformations	LEQT1S	Gaussian elimination
25	1.00	1.20	1.00	1.20	2.20	3.20	1.00
50	1.00	1.22	1.39	1.48	2.96	4.70	1.43
75	1.00	1.18	1.68	1.65	3.32	5.53	1.69
100	1.00	1.13	1.78	1.75	3.63	6.45	1.87
125	1.00	1.11	1.86	1.82	3.81	6.46	1.99
150	1.00	1.09	1.94	1.87	4.06	6.65	2.06
175	1.00	1.09	1.99	1.91	4.21	6.94	2.13
200	1.00	1.07	2.03	1.94	4.32	7.54	2.18

TABLE 5 Ratio to Cholesky on the Honeywell 6050

Dimension	Cholesky	Aasen	Parlett & Reid	Bunch	Householder transformations	LEQT1S	Gaussian elimination
25	1.00	1.38	1.46	1.85	2.54	2.77	1.23
50	1.00	1.32	1.86	2.18	3.30	3.47	1.55
75	1.00	1.25	2.04	2.32	3.63	3.85	1.68
100	1.00	1.19	2.13	2.40	3.82	3.83	1.77
125	1.00	1.17	2.21	2.46	3.94	3.99	1.83
150	1.00	1.14	2.26	2.50	4.03	4.45	1.87
175	1.00	1.12	2.29	2.53	4.09	4.50	1.90
200	1.00	1.11	2.32	2.55	4.15	4.51	1.92

Tables 1 and 2 illustrate several points. Probably the most noteworthy is the relative execution times of the Aasen and Bunch codes. Although their arithmetic operation counts are essentially the same, the $O(n^3)$ comparisons required in Bunch's algorithm compared to the $O(n^2)$ comparisons in Aasen's algorithm led to the latter's superior performance. In fairness, we should point out that Bunch has recently developed a partial pivoting modification of his algorithm [3], and we would expect a careful implementation of it to perform about as well but probably no better than our implementation of Aasen's algorithm.

Tables 1 and 2 show that the code LEQT1S fared very poorly, and as mentioned in section 2, this was the reason we included George's implementation of Bunch's algorithm. We should point out that IMSL is aware of the shortcomings of LEQT1S, and is taking steps to improve it. Examination of the code provides immediately the reasons for its poor performance; a substantial number of unnecessary subscript related calculations is being done in the inner loops of the code. An example taken from the code appears below.

```
      DO 190 K=I0,I
         DO 190 J=I1,N
            JK=J*(J-1)/2+K
            B(K)=B(K)-A(JK)*B(J)
  190 CONTINUE
```

This loop could be profitably replaced by the essentially equivalent code below.

```
      DO 190 K=I0,I
         JK=K+I1*(I1-1)/2
         DO 190 J=I1,N
            B(K)=B(K)-A(JK)*B(J)
            JK=JK+J
  190 CONTINUE
```

The other basic difference in the two implementations of Bunch's algorithm was mentioned in section 2. At each stage in the decomposition, we must find the largest component in magnitude in the part of the matrix remaining to be factored. In LEQT1S this search is done in a separate loop, which means that the component in the part of the matrix remaining to be factored is examined twice. In our implementation, we find the maximal element and its position during the transformation which actually produces the component. Although this obviously does not reduce the number of comparisons, it does reduce the amount of loop overhead.

It is clear from Table 3 that the execution times depend strongly on the Fortran compilers. Compared with the Honeywell, LEQT1S ran twice as fast on the IBM 360/75 while BUNCH ran almost 4-1/2 times as fast. This factor of 2-1/2 could easily lead one to draw incorrect conclusions.

Tables 4 and 5 illustrate some interesting points. Clearly, the ratio of the execution times for the AASEN code compared to the Cholesky code is converging to one. This is expected since the inner loop routine in both cases is the same (i.e. SUBIP). However, for the other subroutines, the generated code for the inner loops is (understandably) different, and the relative execution times predicted in theory by counting operations do not always hold. For example, one would predict that Parlett and Reid would execute in the same time as Gaussian elimination, and both these to execute about half as fast as the Cholesky code (for large n). On the IBM 360/75, the code PAREID is about half as fast as CHLSKY (for large n), but GAUSS is slightly slower. On the Honeywell this situation is reversed. Also BUNCH is about half as fast as CHLSKY on the 360/75, but on the 6050 it is only about 2/5 as fast.

In the interests of portability, our codes have been run through the Fortran verifier [10] written at Bell Telephone Laboratories. As far as we can discover, our codes comply with the ANSI standard Fortran [12].† Of course the two routines RANDS and TIMER are not portable. The routine RANDS simply generates random numbers for our matrices and the routine TIMER prints out the hollerith string passed to it and the time elapsed since the previous call to TIMER. We found it a relatively simple matter to move our code from the IBM 360/75 to the Honeywell 6050.

† The IMSL code LEQT1S does not conform to the ANSI standard, but since IMSL provides codes for various machines, it was not changed.

4. Conclusions

We feel Tables 1 and 2 provide fairly clear-cut conclusions. For symmetric indefinite linear systems of equations, the algorithm proposed by Aasen appears to be substantially superior to its potential competitors. Again, we note that the recently developed partial pivoting version of Bunch's algorithm should perform comparably [3]. For very small systems (of dimension less than 20), it is probably simplier and cheaper to store the entire matrix and use a good code for ordinary Gaussian elimination with partial pivoting.

5. Acknowledgement

The authors are grateful to Professor W.M. Gentleman for assistance in the use of the Fortran verifier, the Honeywell computer system, and in the interpretation of some of the results in our tables.

References

[1] J.O. Aasen, "On the reduction of a symmetric matrix to tridiagonal form", BIT 11 (1973), pp.233-242.

[2] James R. Bunch, "Analysis of the diagonal pivoting method", SIAM J. Numer. Anal. 8 (1971), pp.656-680.

[3] James R. Bunch, "Partial pivoting strategies for symmetric matrices", SIAM J. Numer. Anal., 11 (1974), pp.521-528.

[4] James R. Bunch and B.N. Parlett, "Direct methods for solving symmetric indefinite systems of linear equations", SIAM J. Numer. Anal. 8 (1971), pp.639-655.

[5] G.E. Forsythe and Cleve B. Moler, Computer Solution of Linear Algebraic Systems, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1967.

[6] G.E. Forsythe, M.A. Malcolm, C.B. Moler, Computer Methods for Solving Mathematical Problems, to be published by Prentice Hall.

[7] Alan George, "Computer implementation of the finite element method", Report Stan-CS-71-208, Stanford University Computer Science Department Technical Report, Stanford, California 94305.

[8] The IMSL Library, Vol.1, International Mathematical and Statistical Libraries Inc., 1973, Houston, Texas.

[9] B.N. Parlett and J.K. Reid, "On the solution of a system of linear equations whose matrix is symmetric but not definite", BIT 10 (1970), pp.386-397.

[10] Barbara G. Ryder, "The Fortran Verifier, Motivation and Implementation", Software Practices and Experience, (to appear).

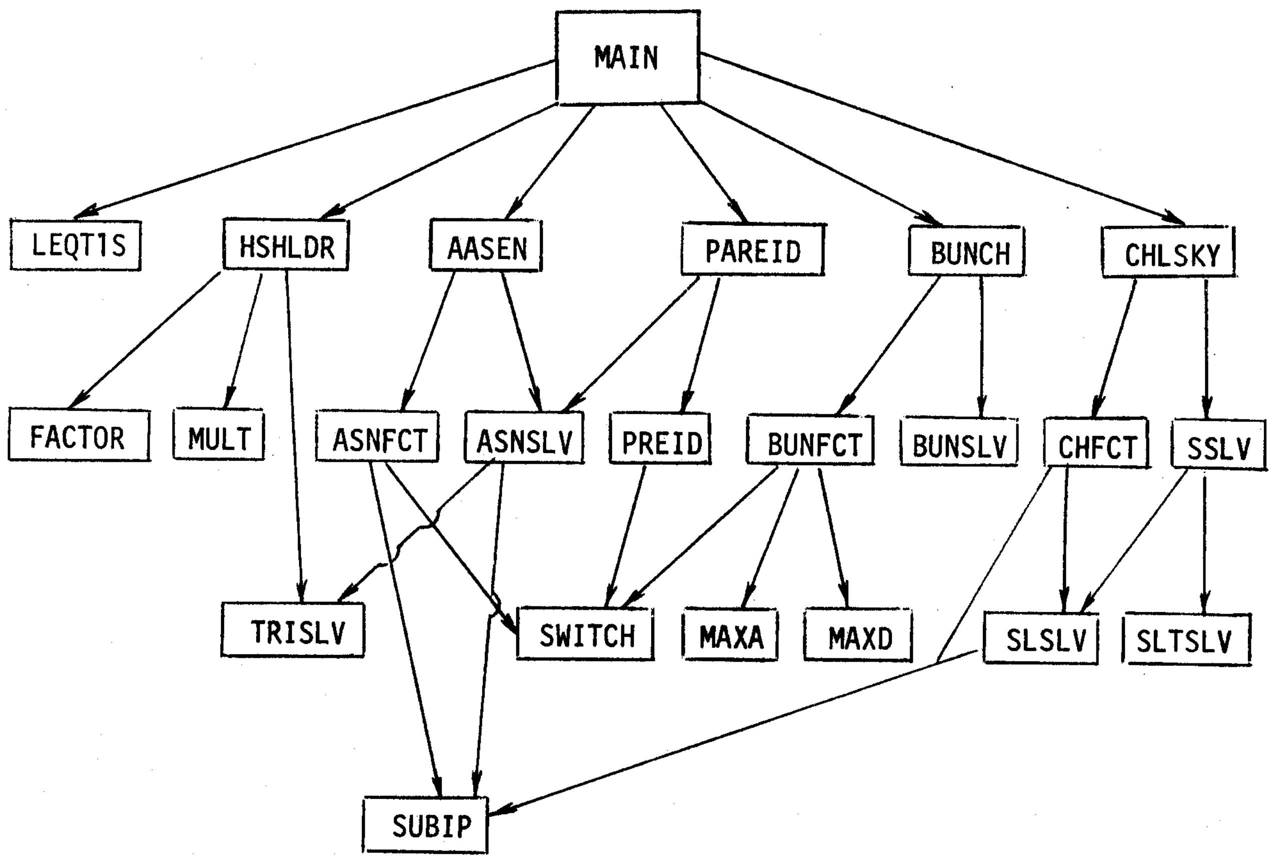
[11] B.T. Smith, J.M. Boyle, B.S. Garbow, Y. Ikebe, V.C. Kelma, and C.B. Moler, Lecture Notes in Computer Science, Matrix Eigensystem Routines - Eispack Guide, Springer Verlag, New York, 1974, pp.351-352.

[12] "USA Standard Fortran", USAS X3.9 - 1966, United States of America Standards Institute, New York, 1966.

[13] J. Westlake, Handbook of Numerical Matrix Inversion and Solution of Linear Equations, John Wiley, New York, 1968.

[14] J.H. Wilkinson and C. Reinseh, Handbook for Automatic Computation, Vol.II, Springer Verlag, New York, 1971.

APPENDIX



Control relationship among the principal subroutines

Page numbers for listings of the subroutines:

```
C***********************************************************************
C       MAIN
C       THIS PROGRAM IS DESIGNED TO TEST VARIOUS SUBROUTINES FOR
C       SOLVING SYMMETRIC (BUT PROBABLY INDEFINITE) FULL SYSTEMS
C       OF EQUATIONS
C
C       N-DIMENSION OF THE SYSTEM
C
C       A-ARRAY USED TO STORE THE MATRIX
C
C       AZ-SUBROUTINE TO MULTIPLY A TIMES A VECTOR
C
C       RHS-RIGHT HAND SIDE FOR THE SYSTEM
C
C       X - SOLUTION VECTOR
C
C       C, L, E, IP - WORKING STORAGE
C***********************************************************************
        REAL A(20100), X(200), RHS(200), C(200), D(200), E(200)
        INTEGER IP(200), N, L, I
C-----------------------------------------------------------------------
C
      CALL ALLOW(100000,0,0)
      CALL STIMER
C
      DO 5 N  = 25, 25, 25
         WRITE(6, 99) N
99       FORMAT( 26HIDIMENSION BEING TESTED IS  , I4)
         DO 1 ITEST  =  1, 1
            DO 7 L  =  1, 6
               NSTART  =  1234567
               CALL RANDS(NSTART, A, N*(N + 1)/2)
C
               DO 2 I  =  1, N
                  X(I)  =  1.0
    2          CONTINUE
C
               CALL AZ(N, A, X, RHS, L - 1)
               CALL TIMER( 20HINITIALIZATION DONE  )
C
               GO TO (10, 20, 30, 40, 50, 60), L
C
   10          CALL CHLSKY(N, A, RHS, 1)
               CALL TIMER( 20HCHOLESKY   DONE       )
               GO TO 70
C
   20          CALL AASEN(N, A, RHS, C, D, E, IP, 1)
               CALL TIMER( 20HAASEN  DONE           )
               GO TO 70
C
   30          CALL PARLID(N, A, RHS, C, D, E, IP, 1)
               CALL TIMEP( 20HPARLETT-REID DONE     )
               GO TO 70
C
   40          CALL BUNCH(N, A, RHS, C, D, E, IP, 1)
               CALL TIMER( 20HBUNCH DONE            )
               GO TO 70
C
   50          CALL HSLDR(N, A, RHS, C, D, E, 1)
               CALL TIMER( 20HHSLDR DONE            )
               GO TO 70
C
   60          CALL LECTIS(A, N, RHS, IP, C, IER)
               CALL TIMER( 20HLECTIS DONE           )
C
   70          CONTINUE
C
               ERROR  =  0.0
               DO 3 I  =  1, N
                  ERROR  =  AMAX1(ERROR, ABS(RHS(I) - 1.0))
    3          CONTINUE
C
               WRITE(6, 101) ERROR
  101          FORMAT(1H0, 12HMAX ERROR IS , E15.8/,
     1         48H------------------------------------------------ )
    7       CONTINUE
    1    CONTINUE
    5 CONTINUE
      STOP
      END
```

```
C***********************************************************************
C
        SUBROUTINE AZ(N, A, Z1, Z2, IFLAG)
C
C***********************************************************************
C
C       THIS ROUTINE CALCULATES THE PRODUCT OF A SYMMETRIC
C       MATRIX AND A VECTOR Z1 AND STORES THE RESULT IN Z2.
C       THE LOWER TRIANGULAR PART OF A IS STORED ROW BY ROW IN THE
C       ONE DIMENSION ARRAY A.
C       IF IFLAG = 0, THE MATRIX IS MADE DIAGONALLY DOMINANT, AND
C       THE RIGHT HAND SIDE IS MODIFIED SO THE SOLUTION TO THE
C       SYSTEM IS STILL ALL ONES.
C***********************************************************************
        REAL Z1(1), Z2(1), A(1)
        DOUBLE PRECISION TEMP, TEMP1
        INTEGER I, I1, J, K, INDEX, IFLAG
C-----------------------------------------------------------------------
C
        DO 1 I  =  1, N
           TEMP  =  0.0D0
           I1  =  I - 1
           INDEX  =  I*I1/2
           IF ( I1  .EQ.  0 )   GO TO 3
C
           DO 2 J  =  1, I1
              TEMP1  =  Z1(J)
              K = INDEX + J
              TEMP  =  TEMP + A(K)*TEMP1
    2      CONTINUE
C
    3      DO 4 J  =  I, N
              TEMP1  =  Z1(J)
              K = INDEX + I
              TEMP  =  TEMP + A(K)*TEMP1
              INDEX  =  INDEX + J
    4      CONTINUE
C
           Z2(I)  =  TEMP
    1   CONTINUE
C
        IF ( IFLAG .NE. 0 ) RETURN
C
        DO 5 I  =  1, N
           K  =  I*(I + 1)/2
           A(K)  =  A(K) + FLOAT(N)
           Z2(I)  =  Z2(I) + FLOAT(N)
    5   CONTINUE
C
         RETURN
         END
```

```
C*************************************************************************
C
      SUBROUTINE CHLSKY(N, A, RHS, IFACT)
C
C*************************************************************************
C  THIS IS AN IMPLEMENTATION OF CHOLESKY-S ALGORITHM (SEE  COMPUTER
C  SOLUTION OF LINEAR ALGEBRAIC SYSTEMS  BY FORSYTHE AND MOLER,
C  PAGE 114 ) WHERE THE MATRIX A IS STORED IN SYMMETRIC MODE. THE
C  SOLUTION OF  AX = B  IS DONE IN TWO STEPS:
C
C
C                                  T
C      STEP  (1)  FACTOR A = LL    WHERE L IS LOWER TRIANGULAR.
C
C
C                          T
C      STEP(2)  SOLVE LL X = B
C
C
C      STEP (1) IS PERFORMED BY CHFCT UNLESS IFACT = 1, IN WHICH CASE
C  THE STEP IS SKIPPED.  STEP(2) IS PERFORMED BY THE ROUTINE SSLV.
C*************************************************************************
      REAL A(1), RHS(1)
      INTEGER N, IFACT
C-------------------------------------------------------------------------
C
      IF ( IFACT .EQ. 1 ) CALL CHFCT(N, A)
C
      CALL SSLV(N, A, RHS)
C
      RETURN
C
      END
C*************************************************************************
C
      SUBROUTINE CHFCT(N, A)
C
C*************************************************************************
C     SYMMETRIC FACTORIZATION OF A SYMMETRIC POSITIVE DEFINITE
C     MATRIX IN TO L * L(T).  THE LOWER TRIANGULAR MATRIX L REPLACES
C     THE ORIGINAL MATRIX A.  THE LOWER TRIANGLE OF A IS STORED
C     ROW BY ROW IN THE ARRAY A.
C
C
C        THE STANDARD BORDERING TECHNIQUE IS USED. IF B IS THE
C     MATRIX:
C                                  E          V
C                    B    =
C                                  V(T)       S
C
C     WHERE E IS ALREADY FACTORED AS  L * L(T),
C     THEN WE HAVE
C                                  L          0      L(T)      W
C                    B    =
C                                  W(T)       R       0        R
C
C
C     WHERE          W    =     L(-1) V,
C                    R    =     =    SQRT ( S -  W(T) * W ).
C
C
C     NOTE: HERE A(T) IS THE TRANSPOSE OF THE MATRIX A;
C           AND A(-1) IS THE INVERSE OF THE MATRIX A.
C*************************************************************************
      REAL A(1)
      INTEGER N, NM1, J, K, JMK
C-------------------------------------------------------------------------
C
      A(1)  =  SQRT(A(1))
C
      IF ( N .EQ. 1 ) RETURN
C
C     ------------------------------------------
C        APPLY BORDERING TECHNIQUE  N-1 TIMES:
C     USE  SLSLV  TO SOLVE FOR  L(-1) V;
C     AND  SUBIP  TO COMPUTE  S - W(T) * W.
C     ------------------------------------------
      NM1  =  N - 1
      J  =  1
      DO 1 K  =  1, NM1
         J  =  J + 1
         CALL SLSLV(K, A, A(J))
         JMK = J
         J  =  J + K
         A(J)  =  SQRT( SUBIP(K, A(JMK), A(JMK), A(J)))
    1 CONTINUE
      RETURN
      END
C*************************************************************************
C
      SUBROUTINE SLSLV(N, A, X)
C
C*************************************************************************
C     ROUTINE SLSLV SOLVES THE SYSTEM   L Y  =  X,
C     WHERE L IS LOWER TRIANGULAR.
C     THE LOWER TRIANGLE OF L IS STORED ROW BY ROW IN A.
C*************************************************************************
      REAL A(1), X(1)
      INTEGER N, I, K, KPI
C-------------------------------------------------------------------------
C
      K  =  1
      DO 1 I  =  1, N
         KPI = K + I
         X(I)  =  SUBIP(I - 1, X, A(K), X(I))/A(KPI - 1)
         K  =  KPI
    1 CONTINUE
C
      RETURN
      END
C*************************************************************************
C
      SUBROUTINE SLTSLV(N, A, X)
C
C*************************************************************************
C     SLTSLV SOLVES THE LINEAR SYSTEM  L(T) Y  =  X,
C     WHERE L IS LOWER TRIANGULAR.
C     THE LOWER TRIANGLE OF L IS STORED ROW BY ROW IN A.
C*************************************************************************
      REAL A(1), X(1), T
      INTEGER KPJ, I, J, K, L
C-------------------------------------------------------------------------
C
      K  =  N*(N + 1)/2
      T  =  X(N)/A(K)
      X(N)  =  T
C
      IF ( N .EQ. 1 ) RETURN
C
      DO 1 L  =  2, N
         I  =  N - L + 1
         K  =  K - I - 1
         DO 2 J  =  1, I
            KPJ = K + J
            X(J)  =  X(J) - T*A(KPJ)
    2    CONTINUE
         T  =  X(I)/A(K)
         X(I)  =  T
    1 CONTINUE
C
      RETURN
      END
C*************************************************************************
C
      SUBROUTINE SSLV(N, A, X)
C
C*************************************************************************
C     SUBROUTINE SSLV SOLVES THE SYSTEM    L  L(T)  Y  =  X,
C     WHERE L IS LOWER TRIANGULAR AND STORED ROW BY ROW IN THE ARRAY A.
C*************************************************************************
C
      REAL A(1), X(1)
      INTEGER N
C
      CALL SLSLV(N, A, X)
C
      CALL SLTSLV(N, A, X)
C
      RETURN
      END
```

```
C*********************************************************************
C
      SUBROUTINE AASEN(N, A, RHS, C, D, E, IP, IFACT)
C
C*********************************************************************
C     THIS ROUTINE USES AASEN-S ALGORITHM (SEE THE ROUTINE ASNFCT) TO
C  SOLVE THE SYMMETRIC SYSTEM AX = B .  THIS IS DONE AS FOLLOWS:
C
C                                  T
C     STEP (1)  FACTOR A = RSR     WHERE S IS TRIDIAGONAL AND R IS LOWER
C                                  TRIANGULAR
C
C
C                         T
C     STEP (2)  SOLVE  RSR  = B
C
C     STEP (1) IS DONE BY ASNFCT AND STEP (2) BY ASNSLV
C*********************************************************************
      REAL A(1), RHS(1), C(1), D(1), E(1)
      INTEGER IP(1), IFACT
C---------------------------------------------------------------------
C
      IF (IFACT .EQ. 1 ) CALL ASNFCT(N, A, IP, C)
C
      CALL ASNSLV(N, A, RHS, C, D, E, IP)
C
      RETURN
      END


C*********************************************************************
C
      SUBROUTINE ASNSLV(N, A, B, C, D, E, IP)
C
C*********************************************************************
C
C     THIS ROUTINE ACCEPTS THE OUTPUT OF THE ROUTINES PREID OR
C     ASNFCT, WHICH FACTOR THE SYMMETRIC MATRIX A INTO R S R(T),
C     WHERE R IS LOWER TRIANGULAR AND S IS TRIDIAGONAL.  IT USES
C     THESE TO SOLVE RSR(T) = B.
C
C     S(I,I) IS IN POSITION A(K), WHERE K = I*(I+1)/2.
C     S(I,I-1) IS IN POSITION A(K-1).
C     R(I,J) IS IN POSITION A(K-J+I-1)
C
C     THE UNIT LOWER TRIANGULAR MATRIX R HAS ZEROS BELOW THE FIRST
C     DIAGONAL POSITION, SO ITS COLUMNS ARE SHIFTED LEFT ONE
C     POSITION COMPARED TO WHERE THEY WOULD RESIDE IN THE NORMAL
C     SYMMETRIC STORAGE MODE.  THIS LEAVES SPACE FOR THE SUBDIAGONAL
C     OF S.
C     IN ORDER TO UTILIZE THE SUBROUTINE TRISLV, WE UNPACK
C     S AND STORE ITS DIAGONALS IN THE ARRAYS C AND D, THUS USING
C     ONE MORE TEMPORARY VECTOR THAN NECESSARY.
C
C*********************************************************************
      REAL A(1), B(1), C(1), D(1), E(1), TEMP
      INTEGER IP(1), I, J, K, L, JPLUS1, IPLUS1, IM1, NM1, N, NNN
C     ----------------------------------------------------------
C     PERMUTE VECTOR B CORRESPONDING TO THE INTERCHANGES OF STEP 1.
C     ----------------------------------------------------------
C
      IF ( N .EQ. 1 ) GO TO 200
C
      DO 20  I  =  2, N
         K  =  IP(I)
         TEMP  =  B(I)
         B(I)  =  B(K)
         B(K)  =  TEMP
   20 CONTINUE
C
C     ----------------------------------------
C     SOLVE FOR X IN RX  =  B ; SOLUTION IS IN B.
C     FIRST 2 ELEMENTS ARE UNCHANGED
C     ----------------------------------------
C
      IF ( N .LT. 3 )   GO TO 200
C
      K  =  2
      DO 21 I  =  3, N
         K  =  K + I - 1
         B(I)  =  SUBIP( I - 2, A(K), B(2), B(I) )
   21 CONTINUE
C
C
C     ----------------------------------------
C     SOLVE THE TRIDIAGONAL SYSTEM OF EQUATIONS
C     ----------------------------------------
C
  200 CALL UNPACK(N, A, C, D)
C
      CALL TRISLV(N, D, C, E, B)
C
C     -------------------------------------------------------------
C     SOLVE R(T)X  =  Y FOR X ; FIRST AND LAST ELEMENTS UNCHANGED
C     -------------------------------------------------------------
C
      NM1 = N - 1
C
      IF ( N .LE. 2 ) GO TO 400
C
      NNN = N*(N + 1)/2
      K  =  NNN - 1
      I  =  N
      DO 30 L  =  2, NM1
         K  =  K - I
         TEMP  =  B(I)
         I  =  I - 1
C
         DO 31 J  =  2, I
            KPJ = K + J
            B(J)  =  B(J) - TEMP*A(KPJ)
   31    CONTINUE
C
   30 CONTINUE
C
C     -------------------------------------------------------------
C     PERMUTE VECTOR B CORRESPONDING TO THE INTERCHANGES OF STEP 1.
C     -------------------------------------------------------------
C
  400 IF ( N .EQ. 1 ) RETURN
C
      DO 40 I  =  1, NM1
         K  =  N - I + 1
         J  =  IP(K)
         TEMP  =  B(K)
         B(K)  =  B(J)
         B(J)  =  TEMP
   40 CONTINUE
C
  500 RETURN
      END
```

```
C*********************************************************************
C
C       SUBROUTINE ASNFCT(N, A, IP, H)
C
C*********************************************************************
C THIS POUTINE FACTORS A INTO THE FORM R.S.R-TRANSPOSE
C WHERE R IS LOWER TRIANGULAR AND S IS TRIDIAGONAL.  THE
C LOWER TRIANGULAR PART OF A IS STORED ROW BY ROW IN A
C LINEAR ARRAY.  S IS STORED IN THE MAIN AND SUBDIAGONAL
C OF A AND R IS STORED BELOW S IN A.  THE ROW AND COLUMNS
C ARE  INTERCHANGED IF NECESSARY, AND THE CHANGES STORED
C IN IP.
C
C N-DIMENSION OF THE SYSTEM
C A-LINEAR ARRAY OF DIMENSION N*(N + 1)/2 WHICH CONTAINS
C       THE SYMMETRIC ARRAY A
C IP-PERMUTATION VECTOR OF DIMENSION N
C H-WORKING STORAGE OF DIMENSION N
C
C REFERENCE: "ON THE REDUCTION OF A SYMMETRIC MATRIX TO
C TRIDIAGONAL FORM" JAN AASEN, BIT 11(1973) 233-242
C*********************************************************************
        REAL A(1), H(1)
        REAL S, EK1, EK, CMAX
        INTEGER IA, IM2, IM1, I, IPLUS1, IT, IC, K, N, IQ, M
        INTEGER IP(1), J, L, IPLUS2
C--------------------------------------------------------------------
C
        IP(2)  =  2
C
        IF ( N .LE. 2 ) RETURN
C
        IA  =  0
        DO 2 I  =  1, N
C
C          ---------------------------------------------------
C          COMPUTE THE DO LOOP LIMITS NEEDED IN FORTRAN AND THE
C          POINTERS NEEDED IN THE ALGORITHM
C          ---------------------------------------------------
C
           IM2  =  I - 2
           IM1  =  I - 1
           IPLUS1  =  I + 1
           IPLUS2  =  I + 2
           IT  =  IA + 2
           IA  =  IA + I
C
C          ---------------------------------------------------------
C          COMPUTE THE I-1 ELEMENTS H(2), ..., H(I) IN THE I-TH COLUMN
C          OF H AND THE DIAGONAL ELEMENT A(I, I)
C          FIRST THREE COLUMNS OF H MUST BE HANDLED SEPARATELY
C          ---------------------------------------------------------
C
           IF ( I  .GE.  4 )   GO TO 23
C
           GO TO (24, 21, 22), I
C
   21      H(2)  =  A(3)
           GO TO 24
C
   22      H(2)  =  A(3)*A(4) + A(5)
           H(3)  =  A(6) - H(2)*A(4)
           A(6)  =  H(3) - A(5)*A(4)
           GO TO 24
C
C
C          ---------------------------------
C          INITIALIZE THE VARIABLES FOR THE LOOP
C          ---------------------------------
C
   23      IC  =  5
           H(2)  =  A(3)*A(IT - 1) + A(5)*A(IT)
           S  =  A(IA) - H(2)*A(IT - 1)
C
           IF ( IM2  .EQ.  2 )   GO TO 31
C
           DO 3 K  =  3, IM2
              ICMK = IC + 1
              IC  =  IC + K + 1
              IT  =  IT + 1
              H(K)  =  A(ICMK - 1)*A(IT - 2) + A(ICMK)*A(IT - 1)
     1                 + A(IC)*A(IT)
              S  =  S - H(K)*A(IT - 1)
    3      CONTINUE
C
   31      H(IM1)  =  A(IC)*A(IT - 1) + A(IC + 1)*A(IT) + A(IT + 1)
           H(I)  =  S - H(IM1)*A(IT)
           A(IA)  =  H(I) - A(IA - 2)*A(IA - 1)
C
   24      CONTINUE
C
C          ---------------------------------------------------------
C          COMPUTE H(I + 1), ..., H(N).  DETERMINE THE MAXIMUM CMAX
C          OF THESE QUANTITIES AND THE INDEX IQ FOR WHICH |H(IQ)|
C          IS A MAXIMUM.
C          ---------------------------------------------------------
C
           IF ( I .EQ. N )   GO TO 2
C
           IP(IPLUS1)  =  IPLUS1
           CMAX  =  0.0
           IC  =  IA
           J  =  IA + 1
C
           DO 5 K  =  IPLUS1, N
C
              IC  =  IC + K - 1
              A(IC)  =  SUBIP(I - 1, H(2), A(J), A(IC) )
              J  =  J + K
C
              IF ( ABS(A(IC))  .LE.  ABS(CMAX) )   GO TO 5
C
              CMAX  =  A(IC)
              IQ  =  K
C
    5      CONTINUE
C
C          ---------------------------------------------
C          IF MX  =  0 OR Q <  =  IPLUS1 NO PIVOTING IS NEEDED
C          ---------------------------------------------
C
           IF ( CMAX  .EQ.  0.0 )   GO TO 2
C
           IF ( IQ  .NE.  IPLUS1 )   CALL SWITCH(N, A, IQ, IPLUS1)
C
           IP(IPLUS1)  =  IQ
C
C          ---------------------------------------------
C          COMPUTE THE ELEMENTS OF THE TRANSFORMATION MATRIX R
C          ---------------------------------------------
C
           IF ( N  .LT.  IPLUS2 )   GO TO 2
C
           L  =  IA + IPLUS1 + I
C
           DO 6 K  =  IPLUS2, N
              A(L)  =  A(L)/CMAX
              L  =  L + K
    6      CONTINUE
C
    2   CONTINUE
C
        RETURN
        END
```

```
C***********************************************************************
C
      SUBROUTINE PAREID(N, A, RHS, C, D, E, IP, IFACT)
C
C***********************************************************************
C     THIS ROUTINE USES PARLETT AND REID-S ALGORITHM TO SOLVE THE
C  SYMMETRIC SYSTEM AX = B .  THIS WORKS THE SAME AS THE AASEN ROUTINE
C  EXCEPT THAT IT USES THE ROUTINE PAREID TO FACTOR A.
C***********************************************************************
      REAL A(1), RHS(1), C(1), D(1), E(1)
      INTEGER IP(1), IFACT
C-----------------------------------------------------------------------
C
      IF ( IFACT .EQ. 1 ) CALL PREID(N, A, IP)
C
      CALL ASNSLV(N, A, RHS, C, D, E, IP)
C
      RETURN
      END
C***********************************************************************
C
      SUBROUTINE PREID(N, A, IP)
C
C***********************************************************************
C THIS ROUTINE FACTORS A INTO THE FORM R.S.R-TRANSPOSE
C WHERE R IS LOWER TRIANGULAR AND S IS TRIDIAGONAL.  THE
C LOWER TRIANGULAR PART OF A IS STORED ROW BY ROW IN A
C LINEAR ARRAY.  S IS STORED IN THE MAIN AND SUBDIAGONAL
C OF A AND R IS STORED BELOW S IN A.  THE ROW AND COLUMNS
C ARE  INTERCHANGED IF NECESSARY, AND THE CHANGES STORED
C IN IP.
C
C N-DIMENSION OF THE SYSTEM
C A-LINEAR ARRAY OF DIMENSION N*(N + 1)/2 WHICH CONTAINS
C      THE SYMMETRIC ARRAY A
C IP-PERMUTATION VECTOR OF DIMENSION N
C
C REFERENCE: "ON THE SOLUTION OF A SYSTEM OF EQUATIONS WHOSE MATRIX
C IS SYMMETRIC BUT NOT DEFINITE", PARLETT AND REID, BIT 10 (1970).
C***********************************************************************
      REAL A(1), SAVE, T1, T2
      INTEGER IP(1), I, J, K, K1, L, IPLUS1, IPLUS2, DIAG, IPIVOT, J1
C-----------------------------------------------------------------------
C
      IP(N) = N
C
      IF ( N .LE. 2 ) RETURN
C
      NM2 = N - 2
C
      DO 1 I = 1, NM2
          IPLUS1 = I + 1
          IPLUS2 = I + 2
          DIAG = IPLUS1*IPLUS2/2
C
C         -----------------------------------------------------
C         FIND LARGEST COMPONENT IN MAGNITUDE BELOW THE DIAGONAL
C         IN COLUMN I. ITS POSITION IS IPIVOT, AND ITS MAGNITUDE
C         IS PIVOT.
C         -----------------------------------------------------
C
          IPIVOT = IPLUS1
          PIVOT = ABS(A(DIAG - 1))
          K = DIAG + I
C
          DO 2 J = IPLUS2, N
C
              IF ( ABS(A(K)) .LT. PIVOT ) GO TO 3
C
              IPIVOT = J
              PIVOT = ABS(A(K))
C
    3         K = K + J
    2     CONTINUE
C
          IP(IPLUS1) = IPIVOT
C
C         -----------------------------------------------------------
C         IF PIVOT = 0.0 NO TRANFORMATION REQUIRED. (REDUCIBLE MATRIX)
C         -----------------------------------------------------------
C
          IF ( PIVOT .EQ. 0.0 ) GO TO 1
C
C         -----------------------------------------------------------
C         INTERCHANGE ROWS AND COLUMNS TO GET PIVOT IN POSITION(I+1,I).
C         -----------------------------------------------------------
C
          IF ( IPIVOT .NE. IPLUS1 ) CALL SWITCH(N, A, IPIVOT, IPLUS1)
C
C         ------------------------------------------------
C         USE PIVOT TO ZERO POSITIONS (I+J,I), J = 2,3,...N.
C         ------------------------------------------------
C
          SAVE = A(DIAG)
          PIVOT = A(DIAG - 1)
          K = DIAG + I
C
          DO 4 J = IPLUS2, N
              T1 = A(K)/PIVOT
              A(K) = T1
              T2 = A(K + 1)
              A(K + 1) = T2 - SAVE*T1
C
              K1 = DIAG + I
              L = K + 2
C
              DO 5 J1 = IPLUS2, J
                  A(L) = A(L) - T2*A(K1) - T1*A(K1 + 1)
                  K1 = K1 + J1
                  L = L + 1
    5         CONTINUE
C
              K = K + J
    4     CONTINUE
    1 CONTINUE
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE BUNCH(N, A, RHS, C, D, E, IP, IFACT)
C
C***********************************************************************
C     THIS ROUTINE USES BUNCH-S DIAGONAL PIVOTING ALGORITHM TO SOLVE
C  THE SYMMETRIC LINEAR SYSTEM AX = B. (SEE BUNFCT FOR DETAILS OF
C  BUNCH-S ALGORITHM) THE SOLUTION IS DONE AS FOLLOWS.
C
C                            T
C     STEP(1)  FACTOR A = LDL  WHERE D IS A DIAGONAL MATRIX OF 1 BY 1
C                         AND 2 BY 2 BLOCKS AND L IS LOWER TRIANGULAR
C
C                           T
C     STEP (2) SOLVE LDL  = B
C
C     STEP (1) IS PERFORMED BY BUNFCT WHICH IS CALLED IF IFACT IS ONE.
C  STEP (2) IS DONE BY THE ROUTINE  BUNSLV.
C***********************************************************************
      REAL A(1), RHS(1), C(1), D(1), E(1)
      INTEGER IP(1), IFACT
C-----------------------------------------------------------------------
C
      IF ( IFACT .EQ. 1 ) CALL BUNFCT(N, A, IP, C, D)
C
      CALL  BUNSLV(N, A, IP, C, D, RHS)
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE MAXD (N, A, K, J, M1)
C
C***********************************************************************
C     FIND M1 = MAX |A(I,I)| FOR K-1 < I < N+1. J IS THE LEAST
C     INTEGER SUCH THAT M1 = |A(J,J)|.
C     THE MATRIX IS STORED IN SYMMETRIC STORAGE MODE.
C***********************************************************************
      INTEGER N, K, J, I, L, KP1
      REAL A(1), M1
C-----------------------------------------------------------------------
C
      L = K * (K+1) / 2
      M1 = ABS(A(L))
      J = K
C
      IF (K .GE. N) RETURN
C
      KP1 = K + 1
C
      DO 1 I = KP1, N
        L = L + I
        IF (ABS(A(L)) .LE. M1) GO TO 1
        M1 = ABS(A(L))
        J = I
    1 CONTINUE
C
      RETURN
      END
C***********************************************************************
C
      SUBROUTINE MAXA(N, A, R, S, M0)
C
C***********************************************************************
C      FIND M0 = MAX |A(I,J)| FOR 0 < I,J < N+1. THE INTEGERS
C      R AND S ARE THE LEAST INTEGERS FOR WHICH M0 = |A(R,S)|.
C      THIS ROUTINE IS ONLY USED ONCE. AFTER THE FIRST STEP
C      OF THE REDUCTION, M0 IS DETERMINED AS THE REDUCTION
C      PROCEEDS IN BUNFCT.
C      THE MATRIX IS STORED IN SYMMETRIC STORAGE MODE.
C***********************************************************************
      REAL TEMP, M0, A(1)
      INTEGER K, IT, I, N, R, S
C-----------------------------------------------------------------------
C
      M0 = A(1)
      R = 1
      S = 1
      IF (N .EQ. 1) RETURN
      IT = 1
      DO 20 I = 2,N
         K = IT
         DO 10 J = 1, I
            K = K + 1
            IF (ABS(A(K)) .LE. M0) GO TO 10
            M0 = ABS(A(K))
            R = I
            S = J
   10    CONTINUE
         IT = IT + I
   20 CONTINUE
C
      RETURN
      END
```

```
C************************************************************************
C
      SUBROUTINE BUNFCT(N, A, PIVOT, PERM, DET)
C
C************************************************************************
C     THIS ROUTINE USES THE DIAGONAL PIVOTING ALGORITHM OF
C     J. R. BUNCH TO COMPUTE THE L-D-L(TRANSPOSE) DECOMPOSITION
C     OF A, WHERE L IS LOWER TRIANGULAR AND D IS A DIAGONAL
C     MATRIX OF 1X1 AND 2X2 BLOCKS. CONSECUTIVE ROWS OF THE
C     LOWER TRIANGLE OF THE MATRIX TO BE DECOMPOSED ARE
C     ASSUMED TO BE IN A, WITH THF I-TH ROW IN POSITIONS
C     A(K), K = L1, ... L2, WHERE L1 = I*(I-1)/2 AND
C     L2 = I*(I+1)/2. A IS ASSUMED INDEFINITE.
C       A IS REPLACED BY L AND D, WHERE L(I,I-1) = 0 IF
C     D(I,I-1) ¬= 0.
C        FOR DETAILS ON THE ALGORITHM, CONSULT BUNCH-S
C     ARTICLE  ANALYSIS CF THE DIAGCNAL PIVOTING METHOD ,
C     SIAM J. NUMER. ANAL. 8 (1971), PP. 656-680.
C************************************************************************
      REAL A(1), DET(1), M0, M1, ALPHA, TEMP, SAVE, TSTRT
      INTEGER N, R, S, I, J, K, IPLUS1, PERM(1), PIVOT(1)
      INTEGER DIAG1, DIAG2, IPLUS2, JT
      LOGICAL TWO
C------------------------------------------------------------------------
C
      ALPHA = (1.0 + SORT(17.0)) / 8.0
C
      I = 1
      CALL MAXA(N, A, R, S, M0)
C
C     ------------------
C     BEGIN MAIN LOOP ...
C     ------------------
C
    1 CALL MAXD(N, A, I, K, M1)
      TWO = M0 * ALPHA .GT. M1
      M0 = 0.0
      IPLUS1 = I + 1
      DIAG1 = I*IPLUS1/2
      DIAG2 = DIAG1 + IPLUS1
      IF (TWO) GO TO 200
C
C     -------------------
C     USE A 1 X 1 PIVOT....
C     -------------------
C
      IF (K .NE. I) CALL SWITCH(N, A, K, I)
      PERM(I) = K
      IF (A(DIAG1) .EQ. 0.0) GO TC 999
      IF (IPLUS1 .GT. N) GO TC 110
C
      TEMP = A(DIAG1)
      JT = DIAG1 + 1
      DO 103 J = IPLUS1, N
         KT = DIAG1 + I
         L = JT
         SAVE = A(L)
         A(L) = SAVL/TEMP
         DO 102 K = IPLUS1, J
            L = L + 1
            A(L) = A(L) - A(KT) * SAVE
            KT = KT + K
            IF (M0 .LE. ABS(A(L))) GO TO 102
            R = J
            S = K
            M0 = ABS(A(L))
  102    CONTINUE
         JT = JT + J
  103 CONTINUE
C
  110 PIVOT(I) = 1
      I = IPLUS1
      IF (I .LE. N) GO TO 1
C
      GO TO 500
C
C     -----------------
C     2 X 2 PIVOT USED ....
C     -----------------
C
  200 PERM(I) = S
      IF (S .NE. I) CALL SWITCH(N, A, S, I)
      PERM(IPLUS1) = R
      IF (R .NE. IPLUS1) CALL SWITCH(N, A, R, IPLUS1)
C
      DET(I) = A(DIAG1) * A(DIAG2) - A(DIAG2 - 1) * A(DIAG2 - 1)
      IF (DET(I).EQ. 0.0) GO TC 999
C
      IPLUS2 = I + 2
      IF (IPLUS2 .GT. N) GO TC 210
C
      JT = DIAG2 + I
      DO 202 J = IPLUS2, N
         J1 = J - 1
         L = JT
         SAVE = A(L)
         TEMP = A(L+1)
         IF (IPLUS2 .GT. J1) GO TO 203
         KT = DIAG2 + 1
         L = L + 1
         DO 201 K = IPLUS2, J1
            L = L + 1
            A(L) = A(L) - A(KT) * SAVE - A(KT + 1) * TEMP
            KT = KT + K
            IF (M0 .LE. ABS(A(L))) GO TO 201
            R = J
            S = K
            M0 = ABS(A(L))
  201    CONTINUE
C
  203    A(JT) = (A(DIAG2) * SAVE - A(DIAG2 - 1) * TEMP)/ DET(I)
         A(JT + 1) = (A(DIAG1) * TEMP - A(DIAG2 - 1) * SAVE) / DET(I)
         K = KT + J
         A(K) = A(K) - A(JT) * SAVE - A(JT + 1) * TEMP
         JT = K
  202 CONTINUE
C
  210 PIVOT(I) = 2
      PIVCT(IPLUS1) = 0
      I = IPLUS2
      IF (I .LE. N) GO TC 1
C
C     ----------------------
C     E N D   M A I N   L O O P
C     ----------------------
C
  500 RETURN
C
C     ------------
C      ERROR RETURN
C     ------------
C
  999 WRITE(6, 123)
  123 FORMAT(16H SINGULAR MATRIX )
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE  BUNSLV(N, A, PIVOT, PERM, DET, B)
C
C***********************************************************************
C     THIS ROUTINE SOLVES A.X = B, WHERE A CONTAINS THE L-D-L(T)
C     DECOMPOSITION PRODUCED BY BUNFCT.
C***********************************************************************
      REAL A(1), DET(1), M0, M1, B(1), TEMP, SAVE
      INTEGER I, J, K, K00, K11, K10, IT, IM1, IT1, I1, JT, IC, N, R, S
      INTEGER PERM(1), PIVOT(1)
C-----------------------------------------------------------------------
C
C     ----------------------------------------------
C     APPLY PERMUTATIONS PERFORMED BY BUNFCT.
C     ----------------------------------------------
C
      DO 301 I = 1, N
         SAVE = B(I)
         IC = PERM(I)
         B(I) = B(IC)
         B(IC) = SAVE
  301 CONTINUE
C
C     ----------------------------------------------------
C     SOLVE L.X = B AND PLACE THE RESULT BACK IN B.
C     ----------------------------------------------------
C
      I = 0
  400 IM1 = I
      I = I + 1
      IF (I .GT. N) GO TO 499
      IF (PIVOT(I) .EQ. 0) IM1 = IM1 - 1
      IF (IM1 .LT. 1) GO TO 400
C
      IT = I * (I-1) / 2
      DO 401 J = 1, IM1
         K = IT + J
         B(I) = B(I) - A(K) * B(J)
  401 CONTINUE
      GO TO 400
C
C     -----------------------------------------------
C     SOLVE D.X = B AND PLACE THE RESULT IN B.
C     -----------------------------------------------
C
  499 I = 1
      IT = 0
  500 IF (PIVOT(I) .EQ. 2) GO TO 502
      K = IT + I
      B(I) = B(I) / A(K)
      I = I + 1
      GO TO 509
C
  502 TEMP = B(I)
      I1 = I + 1
      IT1 = I * I1 / 2
      SAVE = B(I1)
      K11 = IT1 + I1
      K10 = IT1 + I
      K00 = IT + I
      B(I) = (TEMP * A(K11) - SAVE * A(K10)) / DET(I)
      B(I1) = (SAVE * A(K00) - TEMP * A(K10)) / DET(I)
      I = I + 2
  509 IT = I * (I-1) / 2
      IF (I .LE. N) GO TO 500
C
C     -------------------------------------------------
C     SOLVE L(T).X = B AND PLACE THE RESULT IN B.
C     -------------------------------------------------
C
      I = N
  600 I1 = I
      I = I - 1
      IF (I .LE. 0) GO TO 1000
      IF (PIVOT(I) .EQ. 2) I1 = I1 + 1
      IF (I1 .GT. N) GO TO 600
C
      JT = I * I1 / 2
      DO 601 J = I1, N
         K = JT + I
         B(I) = B(I) - A(K) * B(J)
         JT = JT + J
  601 CONTINUE
      GO TO 600
C
C     -----------------------------------------------------
C     APPLY INVERSE OF PERMUTATIONS PERFORMED BY BUNFCT.
C     -----------------------------------------------------
C
 1000 I = N
  700 IC = PERM(I)
      SAVE = B(I)
      B(I) = B(IC)
      B(IC) = SAVE
      I = I - 1
      IF (I .GT. 0) GO TO 700
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE HSLER( N, A, B, D, E, W, FACT )
C
C***********************************************************************
C
C     THIS POUTINE SOLVES THE LINEAR SYMMETRIC SYSTEM
C
C            A X = B
C
C     USING HOUSEHOLDER TRANSFORMATIONS TO REDUCE A MATRIX A TO
C     TRIDIAGONAL FORM.  THAT IS, WE PUT
C
C                  T
C            A = P T P
C
C
C     WHERE P IS ORTHOGONAL AND T  IS TRIDIAGONAL.
C     THE FOLLOWING STEPS ARE PERFORMED:
C
C
C                     T
C        (1) FACTOR A = P T P
C
C                    T
C        (2) COMPUTE P B
C
C                        T
C        (3) SOLVE  T Y = P B
C
C        (4) COMPUTE  X = PY
C
C        THE MATRIX A IS STORED IN SYMMETRIC MODE ( THAT IS, THE
C     LOWER TRIANGULAR PART OF A IS STORED ROW BY ROW IN A LINEAR
C     ARRAY ).  THE MATRIX P IS STORED IN FACTORED FORM IN THE
C     ORIGINAL MATRIX A.
C
C     N - THE DIMENSION OF THE SYSTEM
C
C     A(1::N*(N+1)/2) - THE LINEAR ARRAY FOR THE MATRIX
C
C     D(1::N) - THE ARRAY FOR THE DIAGONAL ENTRIES OF THE TRIDIAGONAL
C               MATRIX
C
C     E(2::N) - THE ARRAY FOR THE SUB - DIAGONAL ENTRIES OF THE
C               TRIDIAGONAL MATRIX
C
C     B(1::N) - ON ENTRY B CONTAINS THE RIGHT HAND SIDE OF THE SYSTEM
C     ON COMPLETION IT CONTAINS THE SOLUTION
C
C     W(1::N) - WORKING STORAGE OF DIMENSION N
C***********************************************************************
      REAL A(1), B(1), D(1), E(1), W(1)
      INTEGER N, FACT
C-----------------------------------------------------------------------
C     ---------------
C     STEPS (1) TO (4)
C     ---------------
C
      IF ( FACT .NE. 0 ) CALL FACTOR(N, A, D, E)
C
      CALL MULT( N, A, B, 1 )
C
      CALL TRISLV(N, D, E, W, B)
C
      CALL MULT( N, A, B, 0 )
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE MULT( N, A, B, TRANS )
C
C***********************************************************************
C
C        THIS ROUTINE MULTIPLIES THE VECTOR B BY EITHER AN
C     ORTHOGONAL MATRIX OR ITS TRANSPOSE, WHERE THE MATRIX IS
C     STORED IN FACTORED HOUSEHOLDER FORM. (AS PRODUCED BY THE
C     ROUTINE FACTOR)
C
C     N - DIMENSION OF THE VECTOR B
C
C     TRANS = 0  -  MULTIPLIES BY THE MATRIX
C             1  -  MULTIPLIES BY THE TRANSPOSE OF THE MATRIX
C***********************************************************************
      REAL  A(1), B(1), H, S
      INTEGER  NPLUS2, N, IA, I, L, K, I1, TRANS
C-----------------------------------------------------------------------
C
      NPLUS2 = N + 2
      IA = 0
      IF (TRANS .EQ. 1 ) IA = N*(N + 1)/2
C
      DO 30 I1 = 2, N
C
         IF (TRANS .EQ. 0 ) GO TO 1
C
            I = NPLUS2 - I1
            IA = IA - I
            L = I - 1
C
            GO TO 2
C
    1    I = I1
         L = I - 1
         IA = IA + L
C
    2    CONTINUE
         J = IA + I
         H = A(J)
C
         IF ( H .EQ. 0.0) GO TO 30
C
         S = 0.0
         DO 10 K = 1, L
            J = IA + K
            S = S + A(J)*B(K)
   10    CONTINUE
         S = S/H
         DO 20 K = 1, L
            J = IA + K
            B(K) = B(K) - S*A(J)
   20    CONTINUE
   30 CONTINUE
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE FACTOR( N, A, D, E )
C
C***********************************************************************
C
C        THIS ROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE
C     TRED3, HANDBOOK FOR AUTOMATIC COMPUTATION, VOLUME 2, LINEAR
C     ALGEBRA, 218 - 219 (1971), BY WILKINSON AND REINSCH.  THE
C     ROUTINE PERFORMS HOUSEHOLDER TRANSFORMATIONS ON A TO REDUCE
C     IT TO TRIDIAGONAL FORM.  THE MATRIX A IS STORED IN SYMMETRIC
C     MODE.
C
C     N - DIMENSION OF THE SYSTEM
C
C     A - THE LINEAR ARRAY USED TO STORE THE MATRIX
C
C     D(1:N) - DIAGONAL COMPONENTS OF THE TRIDIAGONAL MATRIX
C
C     E(2:N) - SUB-DIAGONAL COMPONENTS OF THE TRIDIAGONAL MATRIX
C
C        ON COMPLETION A CONTAINS THE FACTORED FORM OF THE
C     ORTHOGONAL MATRIX USED IN THE TRANSFORMATION.
C***********************************************************************
      REAL A(1), D(1), E(1), H, SCALE, F, G, T
      INTEGER N, NPLUS1, I, IA, I1, L, K, J, JK
C-----------------------------------------------------------------------
C
      NPLUS1 = N + 1
      IA = NPLUS1*N/2
C
C     --------------------------
C     DETERMINE THE TRANSFORMATION
C     --------------------------
C
C     DO FOR I = N TO 1 STEP -1
C
      DO 300 I1 = 1, N
         I = NPLUS1 - I1
         L = I - 1
         IA = IA -1
         H = 0.0
         SCALE = 0.0
C
         IF ( L .LT. 2 ) GO TO 130
C
C        -------------------------------------
C        DETERMINE THE SCALING FACTOR AND SCALE
C        -------------------------------------
C
         DO 120 K = 1, L
            J = IA + K
            SCALE = SCALE + ABS( A(J) )
  120    CONTINUE
C
         IF ( SCALE .NE. 0.0 ) GO TO 140
C
  130    H = 0.0
         J = IA + L
         IF ( L .GT. 0 ) E(I) = A(J)
C
         GO TO 290
C
  140    DO 150 K = 1, L
            J = IA + K
            F = A(J)/SCALE
            A(J) = F
            H = H + F*F
            D(K) = F
  150    CONTINUE
C
         G = -SIGN( SQRT(H), F )
         E(I) = SCALE*G
         H = H -F*G
         T = F - G
         J = IA + L
         A(J) = T
         D(L) = T
         F = 0.0
C
C        -------------------
C        FORM ELEMENTS OF A.U
C        -------------------
C
         JPK = 1
         DO 210 J = 1, L
            G = 0.0
            JM1 = J - 1
            JPK = JPK + JM1
            JK = JPK
C
            IF ( JM1 .EQ. 0 )  GO TO 201
C
            DO 200 K = 1, JM1
               G = G + A(JK)*D(K)
               JK = JK + 1
  200       CONTINUE
C
  201       DO 202 K = J, L
               G = G + A(JK)*D(K)
               JK = JK + K
  202       CONTINUE
C
C           ---------------------
C           FORM THE ELEMENT OF P
C           ---------------------
C
            E(J) = G/H
            F = F + E(J)*D(J)
  210    CONTINUE
C
C        ------
C        FORM K
C        ------
C
         HH = F/( H + H)
         JK = 0
C
C        ------------------
C        FORM THE REDUCED A
C        ------------------
C
         DO 230 J = 1, L
            F = D(J)
            G = E(J) - HH*F
            E(J) = G
            DO 220 K = 1, J
               JK = JK + 1
               A(JK) = A(JK) - F*E(K) - G*D(K)
  220       CONTINUE
  230    CONTINUE
C
         DO 280 K = 1, L
            JK = IA + K
            A(JK) = A(JK)*SCALE
  280    CONTINUE
C
  290    J = IA + I
         D(I) = A(J)
         A(J) = H*SCALE*SCALE
  300 CONTINUE
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE TRISLV(N, U, V, W, B)
C
C***********************************************************************
C  THIS SUBROUTINE PERFORMS GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C  ON A SYMMETRIC TRIDIAGONAL SYSTEM WHOSE DIAGONAL ELEMENTS ARE STORED
C  IN U AND WHOSE SUBDIAGONAL ELEMENTS ARE STORED IN V(2), ..., V(N)
C
C  ON ENTRY,  B CONTAINS THE RIGHT - HAND - SIDE
C  ON EXIT,  B CONTAINS THE SOLUTION
C***********************************************************************
      REAL U(1), V(1), B(1), W(1)
      INTEGER I, J, N, NM1
C-----------------------------------------------------------------------
C
C      ---------------------------------
C      INITIALIZE THE GAUSSIAN ELIMINATION
C      ---------------------------------
C
      P = U(1)
      Q = V(2)
      W(N - 1) = 0.0
      NM1 = N - 1
C
      IF ( N .EQ. 1 ) GO TO 215
C
      DO 213 I = 1, NM1
C
         IF(ABS(V(I + 1)).LE.ABS(P))GO TO 211
C
C         ------------------
C         INTERCHANGE OF ROWS
C         ------------------
C
         U(I) = V(I + 1)
         V(I) = U(I + 1)
C
         IF (I .NE. NM1) W(I) = V(I + 2)
C
C         ---------------------------------
C         INTERCHANGE THE RIGHT HAND SIDES
C         ---------------------------------
C
         TEMP = B(I)
         B(I) = B(I + 1)
         B(I + 1) = TEMP
C
         X = P
         Y = Q
         Z = 0.0
C
         GO TO 212
C
  211    CONTINUE
C
C         ---------------------
C         NO INTERCHANGE OF ROWS
C         ---------------------
C
         U(I) = P
         V(I) = Q
         W(I) = 0.0
         X = V(I + 1)
         Y = U(I + 1)
C
         IF (I .NE. NM1) Z = V(I + 2)
C
C
  212    XMULT = X/U(I)
         P = Y - XMULT*V(I)
         Q = Z - XMULT*W(I)
         B(I + 1) = B(I + 1) - XMULT*B(I)
  213 CONTINUE
C
C
C      ------------------------------------------
C      PERFORM BACK SUBSTITUTION TO GET THE SOLUTION
C      ------------------------------------------
C
  215 B(N) = B(N)/P
C
      IF ( N .EQ. 1 ) RETURN
C
      B(N - 1) = (B(N - 1) - V(N - 1)*B(N))/U(N - 1)
C
      IF ( N .EQ. 2 ) RETURN
C
      DO 214 I = 3, N
         J = N - I + 1
         B(J) = (B(J) - V(J)*B(J + 1) - W(J)*B(J + 2))/U(J)
  214 CONTINUE
C
      RETURN
      END
C***********************************************************************
C
      FUNCTION SUBIP( N, A, B, R)
C
C***********************************************************************
C     A AND B ARE REAL ARRAYS OF LENGTH N. SUBIP RETURNS THE
C     THE VALUE R - A DOT B.  THE INNER PRODUCT IS COMPUTED
C     IN SINGLE PRECISION. THE PROCEDURE WORKS CORRECTLY FOR N = 0.
C-----------------------------------------------------------------------
      REAL R, T, A(1), B(1)
      INTEGER N, I
C-----------------------------------------------------------------------
C
      T  =  R
      IF ( N .LE. 0 ) GO TO 1
C
      DO 2 I  =  1, N
         T  =  T - A(I)*B(I)
    2 CONTINUE
C
    1 SUBIP  =  T
C
      RETURN
      END
C***********************************************************************
C
      SUBROUTINE UNPACK(N, A, C, D)
C
C***********************************************************************
C     THE ARRAY A CONTAINS A MATRIX STORED IN SYMMETRIC STORAGE
C     MODE.  THIS ROUTINE PUTS THE DIAGONAL ELEMENTS OF THE MATRIX
C     IN D(1), D(2), ... D(N),  AND PUTS THE SUBDIAGONAL ELEMENTS
C     IF THE MATRIX IN C(2), C(3), ... C(N).
C***********************************************************************
      REAL A(1), C(1), D(1)
      INTEGER N, I, K
C-----------------------------------------------------------------------
C
      D(1) = A(1)
C
      IF ( N .EQ. 1 ) RETURN
C
      K = 1
      DO 1 I = 2, N
         K = K + I
         D(I) = A(K)
         C(I) = A(K - 1)
    1 CONTINUE
C
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE SWITCH(N, A, K, I)
C
C***********************************************************************
C     INTERCHANGE ROWS AND COLUMNS I AND K, WHERE WE ASSUME
C     THAT I < K.
C     THE LOWER TRIANGLE OF THE SYMMETRIC MATRIX IS STORED ROW
C     BY ROW IN THE ARRAY A.
C***********************************************************************
      REAL T, A(1)
      INTEGER N, I, J, K, L, M, IPLUS1, KPLUS1, KMI, KDIAG, J1, J2
C-----------------------------------------------------------------------
C
       IPLUS1 = I + 1
       KPLUS1 = K + 1
       KMI = K - I
       KDIAG = K*KPLUS1/2
       J1 = I*(I - 1)/2 + 1
       J2 = J1 + I - 2
C
       IF ( K .EQ. IPLUS1 ) GO TO 10
C
C     ---------------------------------------------------------------
C     INTERCHANGE ELEMENTS I+1, I+2, ..., N-K-1  OF ROW (COLUMN) I
C     AND ROW (COLUMN) K.
C     ---------------------------------------------------------------
C
       M = J1 + I + I - 1
       L = KMI - 1
       L1 = KDIAG - KMI + 1
       DO 1 J = 1, L
           T = A(L1)
           A(L1) = A(M)
           A(M) = T
           L1 = L1 + 1
           M = M + I + J
   1   CONTINUE
C
  10   IF ( I .EQ. 1 ) GO TO 20
C
C      ------------------------------------------------------------
C      INTERCHANGE THE FIRST I-1 ELEMENTS OF ROW (COLUMN) I AND
C      ROW (COLUMN) K.
C      ------------------------------------------------------------
C
        L = KDIAG - K + 1
        DO 2 J = J1, J2
            T = A(J)
            A(J) = A(L)
            A(L) = T
            L = L + 1
   2    CONTINUE
C
  20    IF ( K .EQ. N ) GO TO 30
C
C       -----------------------------------------------------------
C       INTERCHANGE THE LAST N-K ELEMENTS OF ROW (COLUMN) I AND
C       ROW (COLUMN) K.
C       -----------------------------------------------------------
C
        L = KDIAG + 1
        M = L + KMI
        DO 3 J = KPLUS1, N
             T = A(L)
             A(L) = A(M)
             A(M) = T
             L = L + J
             M = L + KMI
   3    CONTINUE
C
  30    T = A(KDIAG)
        A(KDIAG) = A(J2 + 1)
        A(J2 + 1) = T
C
        RETURN
        END
```

```
C***********************************************************************
C
      SUBROUTINE RANDS(NSTART, A, NUM)
C
C***********************************************************************
C  THIS ROUTINE GENERATES RANDOM NUMBERS FOR THE IBM 360/75
C  IT GENERATES NUM NUMBERS BETWEEN -1 AND +1 USING THE
C  SEED  NSTART WHICH IS ANY ODD INTEGER.
C***********************************************************************
      REAL A(1)
      INTEGER IMAX, I, NSTART,NUM
C-----------------------------------------------------------------------
      IMAX  =  2147483647
      DO 1 I  =  1, NUM
           A(I)  =  SNGL(DFLOAT(NSTART)/DFLOAT(IMAX))
           NSTART  =  NSTART*65539
           IF ( NSTART  .LE.  0 ) NSTART  =  NSTART + IMAX
    1 CONTINUE
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE DECOMP(N, NDIM, A, IP)
C
C***********************************************************************
      INTEGER  N, NDIM, IP(NDIM)
      DIMENSION A(NDIM, NDIM)
      INTEGER  KP1, I, J, K, M
C***********************************************************************
C
C  MATRIX TRIANGULARIZATION BY GAUSSIAN ELIMINATION.
C  INPUT..
C     N  =  ORDER OF MATRIX.
C     NDIM  =  DECLARED DIMENSICN CF ARRAY  A .
C     A  =  MATRIX TO BE TRIANGULARIZED.
C  OUTPUT..
C     A(I, J), I.LE.J  =  UPPER TRIANGULAR FACTOR, U.
C     A(I, J), I.GT.J  =  MULTIPLIERS  =  LOWER TRIANGULAR
C                          FACTOR, I-L .
C     IP(K), K.LT.N  =  INDEX OF K-TH PIVOT ROW.
C     IP(N)  =  (-1)**(NUMBER OF INTERCHANGES) OR 0  .
C  USE 'SOLVE' TO OBTAIN SOLUTION CF LINEAR SYSTEM.
C  DETERM(A)  =  IP(N)*A(1, 1)*A(2, 2)*...*A(N, N)
C  IF IP(N)  =  0, A IS SINGULAR, SOLVE WILL DIVIDE BY ZERO.
C  INTERCHANGES FINISHED IN  U , ONLY PARTLY IN  L .
C***********************************************************************
C
      IP(N)  =  1
      DO 60 K  =  1, N
         IF(K.EQ.N) GO TC 50
         KP1  =  K+1
         M  =  K
         DO 10 I  =  KP1, N
            IF(ABS(A(I, K)).GT.ABS(A(M, K))) M  =  I
   10    CONTINUE
         IP(K)  =  M
         IF(M.NE.K) IP(N)  =  -IP(N)
         T  =  A(M, K)
         A(M, K)  =  A(K, K)
         A(K, K)  =  T
         IF(T.EQ.0.) GO TO 50
         DO 20 I  =  KP1, N
            A(I, K)  =  -A(I, K)/T
   20    CONTINUE
         DO 40 J  =  KP1, N
            T  =  A(M, J)
            A(M, J)  =  A(K, J)
            A(K, J)  =  T
            IF(T.EQ.0.) GC TO 40
            DO 30 I  =  KP1, N
               A(I, J)  =  A(I, J) + A(I, K)*T
   30       CONTINUE
   40    CONTINUE
   50    IF(A(K, K).EQ.0.) IP(N)  =  0
   60 CONTINUE
      RETURN
      END
```

```
C***********************************************************************
C
      SUBROUTINE SOLVE(N, NDIM, A, B, IP)
C
C***********************************************************************
      INTEGER  N, NDIM, IP(NDIM)
      INTEGER  KB, KM1, NM1, KP1, I, K, M
      DIMENSION A(NDIM, NDIM), B(NDIM)
C***********************************************************************
C
C  SOLUTION OF LINEAR SYSTEM, A*X  =  B .
C  INPUT..
C    N  =  ORDER OF MATRIX.
C    NDIM  =  DECLARED DIMENSION OF ARRAY A.
C    A  =  TRIANGULARIZED MATRIX CBTAINED FROM 'DECOMP' .
C    B  =  RIGHT HAND SIDE VECTOR.
C    IP  =  PIVOT VECTOR OBTAINED FROM 'DECOMP' .
C  DO NOT USE IF DECOMP HAS SET IP(N)  =  0 .
C  OUTPUT..
C    B  =  SOLUTION VECTOR, X .
C***********************************************************************
C
      IF(N.EQ.1) GO TO 50
      NM1  =  N-1
      DO 20 K  =  1, NM1
         KP1  =  K+1
         M  =  IP(K)
         T  =  B(M)
         B(M)  =  B(K)
         B(K)  =  T
         DO 10 I  =  KP1, N
            B(I)  =  B(I) + A(I, K)*T
   10    CONTINUE
   20 CONTINUE
      DO 40 KB  =  1, NM1
         KM1  =  N-KB
         K  =  KM1+1
         B(K)  =  B(K)/A(K, K)
         T  =  -B(K)
         DO 30 I  =  1, KM1
            B(I)  =  B(I) + A(I, K)*T
   30    CONTINUE
   40 CONTINUE
   50 B(1)  =  B(1)/A(1, 1)
      RETURN
      END
```