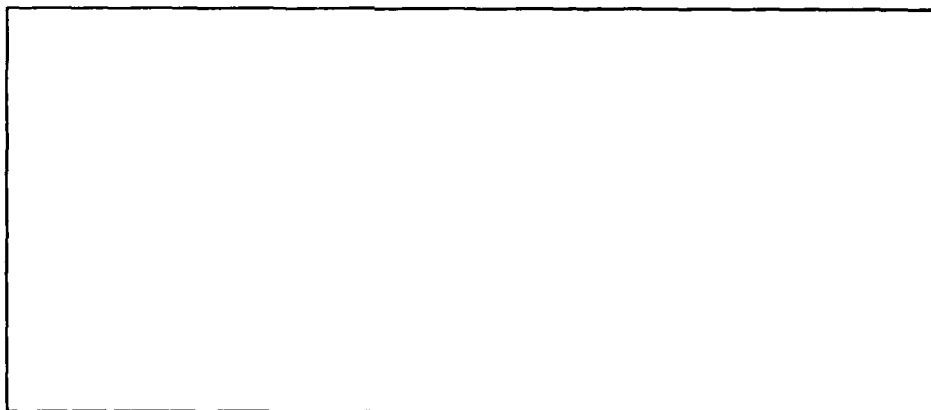


AD A 0 4 7 7 2 5

12



AD No. —
DDC FILE COPY

DDC
DEC 16 1977

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE ✓

DUPLICATE
[illegible text]

(15) N00014-76-C-0277
F49620-77-C-0037

(12)

(9) Research rpt.

(11) Aug 77

(12) 62 p.

(14) RR-114

407 151

AUG. 1977

(6) Yale Sparse Matrix Package,
II. The Nonsymmetric Codes¹
(10) S. C./Eisenstat,² M. C./Gursky,³
M. H./Schultz,² A. H./Sherman⁴
Research Report #114 ✓

DEC 10 1977

¹This research was supported in part by ONR Grant N00014-76-~~FF~~^{Vc-0277},
NSF Grant MCS 76-11460, AFOSR Grant F49620-77-C-0037, and the Chevron Oil
Field Research Company.

²Department of Computer Science, Yale University.

³Department of Electrical Engineering and Computer Science, University
of California, Berkeley.

⁴Department of Computer Science, The University of Texas at Austin.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

1. Introduction

Consider the $N \times N$ system of linear equations

$$(1) \quad Mx = b,$$

where the coefficient matrix M is large, sparse, and nonsymmetric.

Assume that M can be factored in the form

$$M = LDU,$$

where L is a lower triangular matrix, D is a diagonal matrix, and U is a unit upper triangular matrix. Such systems arise frequently in scientific computation, e.g., in finite difference and finite element approximations to non-self-adjoint elliptic boundary value problems. In this report, we present a package of efficient, reliable, well-documented, and portable FORTRAN subroutines for solving these systems. See [3] for a corresponding package for symmetric problems.

Direct methods for solving (1) are generally variations of Gaussian elimination. We form the LDU decomposition of A , and successively solve the triangular systems

$$(2) \quad Ly = b, \quad Dz = y, \quad Ux = z.$$

When M is large ($N \gg 1$), (dense) Gaussian elimination is prohibitively expensive in terms of both the work ($\sim 2/3 N^3$ multiplies) and storage (N^2 words) required. But, since M is sparse, most entries of M , L , and U are zero and there are significant advantages to factoring M without

Availability Codes		
DISC.	AVAIL.	AND IN SPECIAL
A		

storing or operating on these zeroes. Recently, a number of implementations of sparse Gaussian elimination have appeared based on this idea, cf., [2, 6, 7, 8].

In section 2, we describe the scheme used for storing sparse matrices, while, in section 3, we give an overview of the package from the point of view of the user; for further details of the algorithms employed, see [4, 5]. In section 4, we illustrate the performance of the package on a typical model problem. Listings of the three sets of subroutines for factoring and solving the class of sparse nonsymmetric systems under consideration appear in Appendices 1, 2, and 3. These three sets of subroutines have different storage schemes and basically trade-off run-time efficiency for storage. Appendix 4 contains a test driver which sets up a problem and calls all three sets of subroutines for solution. A sample output appears as Appendix 5.

2. Sparse Matrix Storage Schemes

Since the coefficient matrix M and the triangular factors L and U are large and sparse, it is inefficient to store them as dense matrices. The package has two schemes for storing sparse matrices, called the "uncompressed storage scheme" and the "compressed storage scheme." The input matrix M is always stored using the first of these, while the triangular factors L and U may be stored using either one, depending on which subroutines are used. The subroutine NDRV uses the "uncompressed storage scheme" for L and U while the subroutines TDRV and CDRV use the "compressed storage scheme."

The uncompressed storage scheme has been used previously in various forms, cf. [1, 6]. To use it to store the input matrix M requires three one-dimensional arrays: IA , JA , and A . The nonzero entries of M are stored row-by-row in the REAL array A . To identify the individual nonzero entries in a row, we need to know in which column each entry lies. The INTEGER array JA contains the column indices which correspond to the nonzero entries of M , i.e., if $A(K) = M(I, J)$, then $JA(K) = J$. In addition, we need to know where each row starts and how long it is. The INTEGER array IA contains the index positions in JA and A where the rows of M begin, i.e., if $M(I, J)$ is the first (leftmost) entry of the I -th row and $A(K) = M(I, J)$, then $IA(I) = K$. Moreover, $IA(N+1)$ is defined as the index in JA and A of the first location following the last element in the last row. Thus, the number of entries in the I -th row is given by $IA(I+1) - IA(I)$, the nonzero entries of the I -th row are stored

consecutively in

$A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1),$

and the corresponding column indices are stored consecutively in

$JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1).$

For example, the 5x5 matrix

```

      1.  0.  2.  0.  0.
      0.  3.  0.  0.  0.
M =   0.  4.  5.  6.  0.
      0.  0.  0.  7.  0.
      0.  0.  0.  8.  9.
  
```

is stored as

```

      | 1  2  3  4  5  6  7  8  9
      +-----+
IA | 1  3  4  7  8 10
JA | 1  3  2  2  3  4  4  4  5
A  | 1. 2. 3. 4. 5. 6. 7. 8. 9. .
  
```

The overhead in this storage scheme is the storage required for the INTEGER arrays IA and JA. But since IA has $N+1$ entries and JA has one entry for each element of A, the total overhead is approximately equal to the number of nonzero entries in M.

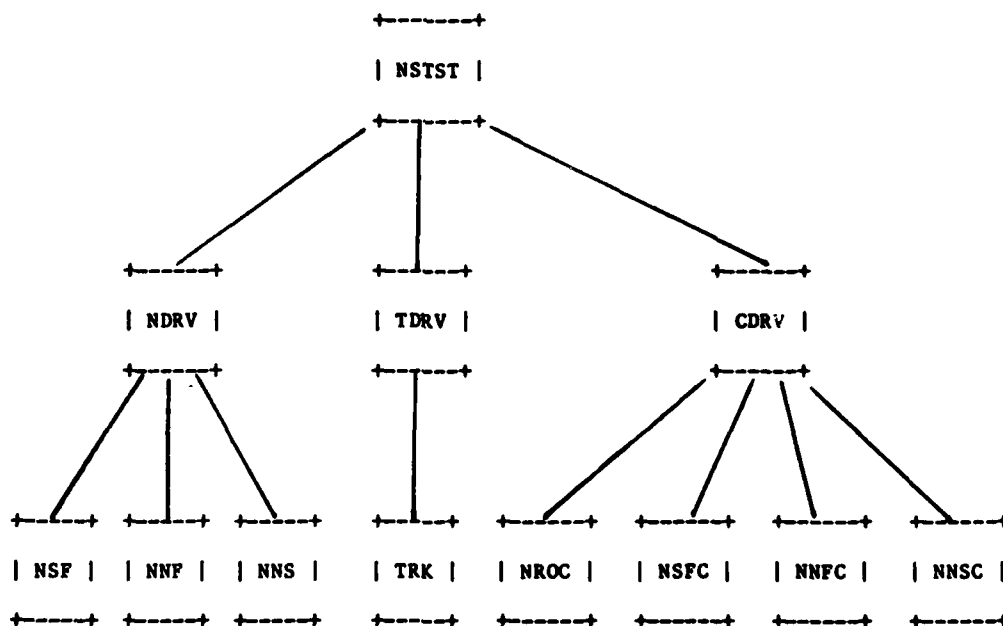
The triangular matrices L and U are stored in basically the same fashion using the arrays IL , JL , L and IU , JU , U respectively, except that the diagonal entries are not stored in these arrays. The diagonal entries of L and U are known to be ones and are not stored and the array D is used to store the reciprocals of the diagonal entries of the diagonal matrix D .

In certain situations, where storage is at a premium, it is essential to reduce storage overhead, even at the cost of decreased runtime efficiency. This can be done by storing L and U with the more complex compressed storage scheme. This scheme incurs more operational overhead than the uncompressed storage scheme, but in many important cases the storage requirement can be substantially reduced. For a detailed description, see [4, 5, 9].

3. A Sparse Nonsymmetric Matrix Package

The package consists of a test driver, three driver subroutines, and eight subroutines (see Figure 1). The three drivers (subroutines NDRV, TDRV, and CDRV) are specific implementation designs which illustrate the space-time tradeoff mentioned in section 2. The test driver (subroutine NSTST) sets up a model sparse nonsymmetric system of linear equations and calls each of the three driver subroutines to solve the linear system. In the remainder of this section, we describe each of these subroutines in somewhat greater detail. The codes themselves are extensively documented; for further details about the algorithms employed see [4, 5].

Figure 1: A schematic overview of the sparse symmetric matrix package



Our basic design for the implementation of sparse elimination follows that of Chang [1], which has proved to be especially robust. The first implementation, NDRV, is designed for speed. It uses the uncompressed storage scheme for M, L, and U because of the smaller operational overhead associated with it. We break the computation up into three distinct phases: symbolic factorization (subroutine NSF), numeric factorization and the solution for one right-hand side (subroutine NNF), and forward and back solution for additional right-hand sides (subroutine NNS). The subroutine NSF computes the zero structures of L and U from that of M (disregarding the numerical entries in M). The subroutine NNF then uses the structural information generated by NSF to compute the numerical entries of L and U and to solve for one right-hand side.

The main advantage of splitting up the computation in this way is flexibility. To solve a single system of equations, it suffices to use NSF and NNF (PATH=1 in NDRV). It should be pointed out here that a one line modification of NNF can be made to allow the solution of a single system without storing L: simply comment out the line

$$L(I) = - LI,$$

as indicated in the code. This change will yield substantial storage savings without the loss in efficiency incurred by TDRV. To solve several systems in which the coefficient matrices have the same zero structure, it suffices to use NSF and NNF only once each for the first system and then to use NNF once for each subsequent system (PATH=2 in

NDRV). Finally, to solve several systems with the same coefficient matrix but different right hand sides, it suffices to use NSF and NNF only once each for the first system, and then to use NNS once for each subsequent system (PATH=3 in NDRV).

A drawback to the multi-phase design of NDRV is that it is necessary to store the description of the zero structures of both L and U. By giving up some flexibility, the second implementation, TDRV, greatly reduces the storage requirements. The entire computation is performed in a single phase (subroutine TRK) to avoid storing either the description or the numerical entries of L. Moreover, U is stored with the compressed storage scheme to reduce the storage overhead. This subroutine incurs more operational overhead than NDRV, and we lose the ability to efficiently solve a sequence of related systems. However the total storage requirements are usually significantly smaller (see Tables 4-6).

Finally, the third implementation, CDRV, attempts to balance the design goals of speed, flexibility, and storage economy. It splits the computation as in NDRV to allow flexibility and efficiency, but it uses the compressed storage scheme as in TDRV to reduce storage overhead. The rows and columns of the original matrix M can be reordered (e.g., to reduce fillin or ensure numerical stability) before calling CDRV. If no reordering is done, then set $R(I)=C(I)=IC(I) = I$ for $I=1, \dots, N$. The solution Z is returned in the original order. If the columns have been reordered (i.e., $C(I).NE.I$ for some I), then CDRV will call a subroutine

NROC which rearranges each row of JA and A, leaving the rows in the original order, but placing the elements of each row in increasing order with respect to the new ordering. If PATH.NE.1, then NROC is assumed to have been called already.

To solve a single system of equations, it suffices to use NROC (if the columns of M have been reordered), NSFC, and NNFC (PATH=1 in CDRV). It should be pointed out here that a one line modification of NNFC can be made to allow the solution of a single system without storing L: simply comment out the line

$$L(IRL(I)) = - LKI,$$

as indicated in the code. This change will yield substantial storage savings without the loss in efficiency incurred in TDRV. To solve several systems in which the coefficient matrices have the same zero structure, it suffices to use NROC, NSFC, and NNFC only once each for the first system and then to use NNFC once for each subsequent system (PATH=2 in CDRV). Finally, to solve several systems with the same coefficient matrix but different right hand sides, it suffices to use NROC, NSFC, and NNFC only once each for the first system, and then to use NNSC once for each subsequent system (PATH=3 in CDRV).

The test driver (program NSTST) is used to verify the performance of the package on a particular computer system, and may be used as a guide to understanding how to use the package. It generates the coefficient matrix for a nonsymmetric five-point difference equation on

a 3x3 grid and chooses the right-hand side so that the solution vector x is (1,2,3,4,5,6,7,8,9) (see Appendix 4). The grid points are given in the natural row-by-row ordering. At each stage the values of all relevant variables are printed out, and a sample output appears as Appendix 5.

4. Performance

One of the most important aspects of any package is its performance in terms of both the time and storage required to solve a typical problem. In Tables 1-6, we present the time and storage required to solve a nonsymmetric five-point difference equation on an $n \times n$ grid for several values of n . These computations were performed in single precision on an IBM 370/158 using the FORTRAN IV Level H Extended compiler.

Table 1: Times for 5-point operator on a 20x20 mesh

Code	NSF(C)	NNF(C)	sec/*	NNS(C)	Total
NDRV	0.213	0.560	9.978	0.063	0.773
TDRV			15.561		0.873
CDRV	0.267	0.790	14.077	0.087	1.057

Table 2: Times for 5-point operator on a 30x30 mesh

Code	NSF(C)	NNF(C)	sec/*	NNS(C)	Total
NDRV	0.583	2.050	9.503	0.170	2.633
TDRV			14.046		3.030
CDRV	0.650	2.810	13.026	0.233	3.460

Table 3: Times for 5-point operator on a 40x40 mesh

Code	NSF(C)	NNF(C)	sec/*	NNS(C)	Total
NDRV	1.197	5.430	9.250	0.347	6.626
TDRV			13.134		7.710
CDRV	1.243	7.313	12.459	0.480	8.556

Table 4: Storage for 5-point operator on a 20x20 mesh

Code	A/JA	L	JL	U	JU	Total ⁺	Mults.
NDRV	1,920	3,368	3,368	3,368	3,368	15,47	56,118
TDRV	1,920			3,368	1,889	7,660	56,118
CDRV	1,920	3,368	1,889	3,368	1,889	13,716	56,118

Table 5: Storage for 5-point operator on a 30x30 mesh

Code	A/JA	L	JL	U	JU	Total ⁺	Mults.
NDRV	4,380	9,456	9,456	9,456	9,456	42,327	215,708
TDRV	4,380			9,456	4,538	19,397	215,708
CDRV	4,380	9,456	4,538	9,456	4,538	35,190	215,708

+Total storage required by driver

Table 6: Storage for 5-point operator on a 40x40 mesh

Code	A/JA	L	JL	U	JU	⁺ Total	Mults.
NDRV	7,840	19,926	19,926	19,926	19,926	87,707	586,970
TDRV	7,840			19,926	8,423	37,952	586,970
CDRV	7,840	19,926	8,423	19,926	8,423	69,500	586,970

REFERENCES

- [1] A. Chang.

Application of sparse matrix methods in electric power system analysis. In Sparse Matrix Proceedings, Report RA1, IBM Research. R. A. Willoughby, editor, Yorktown Heights, New York. 1968.

- [2] A. R. Curtis and J. K. Reid

The solution of large sparse unsymmetric systems of linear equations. JIMA 8:344-353, 1971.

- [3] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman.

The Yale Sparse Matrix Package. I. Symmetric Problems. Yale University, Department of Computer Science, Technical Report #112, May 1977.

- [4] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman.

Considerations in the design of software for sparse Gaussian elimination. In J. R. Bunch and D. J. Rose, editors, Sparse Matrix Computations, Academic Press, 1976, pp. 263-273.

- [5] S. C. Eisenstat and A. H. Sherman.

Efficient implementation of sparse nonsymmetric Gaussian elimination without pivoting. In Proceedings of the SIGNUM Meeting on Software for Partial Differential Equations, ACM SIGNUM Newsletter, December 1975, pp. 26-29.

- [6] F. G. Gustavson.

Some basic techniques for solving sparse systems of linear equations. In D. J. Rose and R. A. Willoughby, editors, Sparse Matrices and Their Applications, Plenum Press, 1972, pp.41-52.

- [7] J. E. Key

Computer Programs for Solution of Large, Sparse, Unsymmetric Systems of Linear Equations. International Journal for Numerical Methods in Engineering 6: 497-509, 1973.

- [8] W. C. Rheinboldt and C. K. Mesztenyi.

Programs for the solution of large sparse matrix problems based on the arc-graph structure. University of Maryland Computer Science Technical Report TR-262, 1973.

- [9] A. H. Sherman.

On the Efficient Solution of Sparse Systems of Linear and Nonlinear Equations. Ph.D. dissertation, Department of Computer Science, Yale University, 1975.

- [10] A. H. Sherman.

Yale sparse matrix package user's guide. Lawrence Livermore Laboratory Report UCID-30114, 1975.

```

C
C
C      Subroutines for Solving Sparse Nonsymmetric Systems
C      of Linear Equations (Uncompressed Pointer Storage)
C
C
C*** Subroutine NDRV
C*** Driver for subroutines for solving sparse nonsymmetric systems of
C      linear equations (uncompressed pointer storage)
C
      SUBROUTINE NDRV
      *      (N, R,C,IC, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, PATH, FLAG)
C
C      PARAMETERS
C      Class abbreviations are --
C      n - INTEGER variable
C      f - REAL variable
C      v - supplies a VALUE to the driver
C      r - returns a RESULT from the driver
C      i - used INTERNALLY by the driver
C      a - ARRAY
C
C      Class | Parameter
C      -----
C      |
C      The nonzero entries of the coefficient matrix M are stored
C      row-by-row in the array A. To identify the individual nonzero
C      entries in each row, we need to know in which column each entry
C      lies. The column indices which correspond to the nonzero entries
C      of M are stored in the array JA; i.e., if  $A(K) = M(I,J)$ , then
C       $JA(K) = J$ . In addition, we need to know where each row starts and
C      how long it is. The index positions in JA and A where the rows of
C      M begin are stored in the array IA; i.e., if  $M(I,J)$  is the first
C      nonzero entry (stored) in the I-th row and  $A(K) = M(I,J)$ , then
C       $IA(I) = K$ . Moreover, the index in JA and A of the first location
C      following the last element in the last row is stored in  $IA(N+1)$ .
C      Thus, the number of entries in the I-th row is given by
C       $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C      consecutively in
C       $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C      and the corresponding column indices are stored consecutively in
C       $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C      For example, the 5 by 5 matrix
C      ( 1. 0. 2. 0. 0.)
C      ( 0. 3. 0. 0. 0.)
C      M = ( 0. 4. 5. 6. 0.)
C      ( 0. 0. 0. 7. 0.)
C      ( 0. 0. 0. 8. 9.)
C      would be stored as
C      | 1 2 3 4 5 6 7 8 9
C      -----
C      IA | 1 3 4 7 8 10
C      JA | 1 3 2 2 3 4 4 4 5
C      A  | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C      nv | N      - number of variables/equations.
C      fva | A      - nonzero entries of the coefficient matrix M, stored
C      |          by rows.
C      |          Size = number of nonzero entries in M.
C      nva | IA     - pointers to delimit the rows in A.

```

```

C      |      Size = N+1.
C nva  | JA   - column numbers corresponding to the elements of A.
C      |      Size = size of A.
C fva  | B    - right-hand side b; B and Z can be the same array.
C      |      Size = N.
C fra  | Z    - solution x; B and Z can be the same array.
C      |      Size = N.
C
C      The rows and columns of the original matrix M can be
C      reordered (e.g., to reduce fillin or ensure numerical stability)
C      before calling the driver. If no reordering is done, then set
C      R(I) = C(I) = IC(I) = I for I=1,...,N. The solution Z is returned
C      in the original order.
C
C nva  | R    - ordering of the rows of M.
C      |      Size = N.
C nva  | C    - ordering of the columns of M.
C      |      Size = N.
C nva  | IC   - inverse of the ordering of the columns of M; i.e.,
C      |      IC(C(I)) = I for I=1,...,N.
C      |      Size = N.
C
C      The solution of the system of linear equations is divided into
C      three stages --
C      NSF -- The matrix M is processed symbolically to determine where
C              fillin will occur during the numeric factorization.
C      NNF -- The matrix M is factored numerically into the product LDU
C              of a unit lower triangular matrix L, a diagonal matrix D,
C              and a unit upper triangular matrix U, and the system
C              Mx = b is solved.
C      NNS -- The linear system Mx = b is solved using the LDU
C              factorization from NNF.
C      For several systems whose coefficient matrices have the same
C      nonzero structure, NSF need be done only once (for the first
C      system); then NNF is done once for each additional system. For
C      several systems with the same coefficient matrix, NSF and NNF need
C      be done only once (for the first system); then NNS is done once
C      for each additional right-hand side.
C
C nv   | PATH - path specification; values and their meanings are --
C      |      1 perform NSF and NNF.
C      |      2 perform NNF only (NSF is assumed to have been
C      |         done in a manner compatible with the storage
C      |         allocation used in the driver).
C      |      3 perform NNS only (NSF and NNF are assumed to
C      |         have been done in a manner compatible with the
C      |         storage allocation used in the driver).
C
C      Various errors are detected by the driver and the individual
C      subroutines.
C
C nr   | FLAG - error flag; values and their meanings are --
C      |      0 No Errors Detected
C      |      N+K Null Row in A -- Row = K
C      |      2N+K Duplicate Entry in A -- Row = K
C      |      3N+K Insufficient Storage in NSF -- Row = K
C      |      4N+1 Insufficient Storage in NNF
C      |      5N+K Null Pivot -- Row = K
C      |      6N+K Insufficient Storage in NSF -- Row = K

```

```

C      |              7N+1  Insufficient Storage in NNF
C      |              8N+K  Zero Pivot -- Row = K
C      |              10N+1  Insufficient Storage in NDRV
C      |              11N+1  Illegal PATH Specification
C
C      Working storage is needed for the factored form of the matrix
C      M plus various temporary vectors. The arrays ISP and RSP should be
C      the same; integer storage is allocated from the beginning of ISP
C      and real storage from the end of RSP.
C
C nv   | NSP   - declared dimension of ISP and RSP; NSP generally must
C      |       be larger than 5N+3 + 2K (where K = (number of
C      |       nonzero entries in M)).
C nvira | ISP   - integer working storage divided up into various arrays
C      |       needed by the subroutines; ISP and RSP should be
C      |       the same array.
C      |       Size = NSP.
C fvira | RSP   - real working storage divided up into various arrays
C      |       needed by the subroutines; ISP and RSP should be
C      |       the same array.
C      |       Size = NSP.
C nr   | ESP   - if sufficient storage was available to perform the
C      |       symbolic factorization (NSF), then ESP is set to the
C      |       amount of excess storage provided (negative if
C      |       insufficient storage was available to perform the
C      |       numeric factorization (NNF)).
C
C      INTEGER R(1), C(1), IC(1), IA(1), JA(1), ISP(1), ESP,
C      *      PATH, FLAG, Q, IM, D, U, ROW, TMP, UMAX
C      REAL A(1), B(1), Z(1), RSP(1)
C
C      IF (PATH.LT.1 .OR. PATH.GT.3) GO TO 111
C ***** Initialize and divide up temporary storage *****
C      IL = 1
C      IU = IL + N+1
C      JL = IU + N+1
C      FLAG = 0
C
C ***** Call NSF if flag is set *****
C      IF (PATH.GT.1) GO TO 2
C      IM = NSP - N
C      Q = IM - (N+1)
C      MAX = Q - JL
C      IF (MAX.LT.0) GO TO 110
C      JLMAX = MAX/2
C      JUTMP = JL + JLMAX
C      JUMAX = Q - JUTMP
C      CALL NSF
C
C      (N, R, IC, IA, JA,
C      *      ISP(IL), ISP(JL), JLMAX, ISP(IU), ISP(JUTMP), JUMAX,
C      *      RSP(Q), RSP(IM), FLAG)
C      IF (FLAG.NE.0) GO TO 100
C ***** Move JU next to JL *****
C      JLMAX = ISP(IL+N)-1
C      JU = JL + JLMAX
C      JUMAX = ISP(IU+N)-1
C      IF (JUMAX.LE.0) GO TO 2
C      DO 1 J=1,JUMAX
C      1      ISP(JU+J-1) = ISP(JUTMP+J-1)

```

```

C
C ***** Call remaining subroutines *****
2  JLMAX = ISP(IL+N)-1
   JU = JL + JLMAX
   JUMAX = ISP(IU+N)-1
   L = JU + JUMAX
   LMAX = JLMAX
   D = L + LMAX
   U = D + N
   ROW = NSP - N
   TMP = ROW - N
   UMAX = TMP - U
   ESP = UMAX - JUMAX
C
   IF (PATH.GT.2) GO TO 3
   CALL NNF
   *   (N, R, C, IC, IA, JA, A, Z, B,
   *   ISP(IL), ISP(JL), RSP(L), LMAX, RSP(D),
   *   ISP(IU), ISP(JU), RSP(U), UMAX,
   *   RSP(ROW), RSP(TMP), FLAG)
   IF (FLAG.NE.0) GO TO 100
   RETURN
C
3  CALL NNS
   *   (N, R, C,
   *   ISP(IL), ISP(JL), RSP(L), RSP(D),
   *   ISP(IU), ISP(JU), RSP(U),
   *   Z, B, RSP(TMP))
   RETURN
C
C ** ERROR: Error Detected in NSF, NNF, or NNS
100 RETURN
C ** ERROR: Insufficient Storage
110 FLAG = 10*N + 1
   RETURN
C ** ERROR: Illegal PATH Specification
111 FLAG = 11*N + 1
   RETURN
   END

```

```

C
C
C -----
C
C      YALE SPARSE MATRIX PACKAGE - NONSYMMETRIC CODES
C      SOLVING THE SYSTEM OF EQUATIONS  $Mx = b$ 
C      (UNCOMPRESSED POINTER STORAGE)
C
C
C I.   SUBROUTINE NAMES
C      Subroutine names are of the form Nxx where --
C      (1) the first letter is N for nonsymmetric matrices;
C      (2) the second letter is either S for symbolic processing or N
C          for numeric processing;
C      (3) the third letter is either F for factorization or S for
C          solution.
C
C
C II.  CALLING SEQUENCES
C      The coefficient matrix can be processed by an ordering routine
C      (e.g., to reduce fillin or ensure numerical stability) before using
C      the remaining subroutines. If no reordering is done, then set
C       $R(I) = C(I) = IC(I) = I$  for  $I=1, \dots, N$ . The calling sequence is --
C      (
C          (matrix ordering))
C      NSF (symbolic factorization to determine where fillin will
C          occur during numeric factorization)
C      NNF (numeric factorization into product LDU of unit lower
C          triangular matrix L, diagonal matrix D, and unit upper
C          triangular matrix U, and solution of linear system)
C      NNS (solution of linear system for additional right-hand
C          side using LDU factorization from NNF)
C
C
C III. STORAGE OF SPARSE MATRICES
C      The nonzero entries of the coefficient matrix M are stored
C      row-by-row in the array A. To identify the individual nonzero
C      entries in each row, we need to know in which column each entry
C      lies. The column indices which correspond to the nonzero entries
C      of M are stored in the array JA; i.e., if  $A(K) = M(I, J)$ , then
C       $JA(K) = J$ . In addition, we need to know where each row starts and
C      how long it is. The index positions in JA and A where the rows of
C      M begin are stored in the array IA; i.e., if  $M(I, J)$  is the first
C      nonzero entry (stored) in the I-th row and  $A(K) = M(I, J)$ , then
C       $IA(I) = K$ . Moreover, the index in JA and A of the first location
C      following the last element in the last row is stored in  $IA(N+1)$ .
C      Thus, the number of entries in the I-th row is given by
C       $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C      consecutively in
C       $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C      and the corresponding column indices are stored consecutively in
C       $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C      For example, the 5 by 5 matrix
C      ( 1. 0. 2. 0. 0.)
C      ( 0. 3. 0. 0. 0.)
C      M = ( 0. 4. 5. 6. 0.)
C          ( 0. 0. 0. 7. 0.)
C          ( 0. 0. 0. 8. 9.)

```

```

C      would be stored as
C      | 1 2 3 4 5 6 7 8 9
C      -----
C      IA | 1 3 4 7 8 10
C      JA | 1 3 2 2 3 4 4 4 5
C      A | 1. 2. 3. 4. 5. 6. 7. 8. 9.

```

The strict triangular portions of the matrices L and U are stored in the same fashion using the arrays IL, JL, L and IU, JU, U respectively. The diagonal entries of L and U are assumed to be equal to one and are not stored. The array D contains the reciprocals of the diagonal entries of the matrix D.

IV. ADDITIONAL STORAGE SAVINGS

In NSF, R and IC can be the same array in the calling sequence if no reordering of the coefficient matrix has been done. In NNF, R, C and IC can all be the same array if no reordering has been done. If only the rows have been reordered, then C and IC can be the same array. If the row and column orderings are the same, then R and C can be the same array. Z and ROW can be the same array.

In NNS, R and C can be the same array if no reordering has been done or if the row and column orderings are the same. Z and B can be the same array; however, then B will be destroyed.

V. PARAMETERS

Following is a list of parameters to the programs. Names are uniform among the various subroutines. Class abbreviations are --

```

C      n - INTEGER variable
C      f - REAL variable
C      v - supplies a VALUE to a subroutine
C      r - returns a RESULT from a subroutine
C      i - used INTERNALLY by a subroutine
C      a - ARRAY

```

Class	Parameter
fva	A - nonzero entries of the coefficient matrix M, stored by rows. Size = number of nonzero entries in M.
fva	B - right-hand side b. Size = N.
nva	C - ordering of the columns of M. Size = N.
fvra	D - reciprocals of the diagonal entries of the matrix D. Size = N.
nr	FLAG - error flag; values and their meanings are -- 0 No Errors Detected N+K Null Row in A -- Row = K 2N+K Duplicate Entry in A -- Row = K 3N+K Insufficient Storage for JL -- Row = K 4N+1 Insufficient Storage for L 5N+K Null Pivot -- Row = K 6N+K Insufficient Storage for JU -- Row = K 7N+1 Insufficient Storage for U 8N+K Zero Pivot -- Row = K
nva	IA - pointers to delimit the rows in A. Size = N+1.
nva	IC - inverse of the ordering of the columns of M; i.e., IC(C(I)) = 1 for I=1,...N. Size = N.
nvra	IL - pointers to delimit the rows in L. Size = N+1.

```

C nvra | IU   - pointers to delimit the rows in U.
C       |      Size = N+1.
C nva   | JA   - column numbers corresponding to the elements of A.
C       |      Size = size of A.
C nvra  | JL   - column numbers corresponding to the elements of L.
C       |      Size = JLMAX.
C nv    | JLMAX - declared dimension of JL; JLMAX must be larger than
C       |      the number of nonzero entries in the strict lower
C       |      triangle of M plus fillin (IL(N+1)-1 after NSF).
C nvra  | JU   - column numbers corresponding to the elements of U.
C       |      Size = JUMAX.
C nv    | JUMAX - declared dimension of JU; JUMAX must be larger than
C       |      the number of nonzero entries in the strict upper
C       |      triangle of M plus fillin (IU(N+1)-1 after NSF).
C fvra  | L    - nonzero entries in the strict lower triangular portion
C       |      of the matrix L, stored by rows.
C       |      Size = LMAX
C nv    | LMAX - declared dimension of L; LMAX must be larger than
C       |      the number of nonzero entries in the strict lower
C       |      triangle of M plus fillin (IL(N+1)-1 after NSF).
C nv    | N    - number of variables/equations.
C nva   | R    - ordering of the rows of M.
C       |      Size = N.
C fvra  | U    - nonzero entries in the strict upper triangular portion
C       |      of the matrix U, stored by rows.
C       |      Size = UMAX.
C nv    | UMAX - declared dimension of U; UMAX must be larger than
C       |      the number of nonzero entries in the strict upper
C       |      triangle of M plus fillin (IU(N+1)-1 after NSF).
C fra   | Z    - solution x.
C       |      Size = N.

```

```

C
C
C
C
C *** Subroutine NSF

```

```

C *** Symbolic LDU-factorization of a nonsymmetric sparse matrix
C      (uncompressed pointer storage)

```

```

C      SUBROUTINE NSF
C      *      (N, R, IC, IA, JA, IL, JL, JLMAX, IU, JU, JUMAX, Q, IM, FLAG)

```

```

C      Input variables:  N, R, IC, IA, JA, JLMAX, JUMAX.

```

```

C      Output variables: IL, JL, IU, JU, FLAG.

```

```

C      Parameters used internally:

```

```

C nia   | Q      - suppose M' is the result of reordering M; if
C       |      processing of the Kth row of M' (hence the Kth rows
C       |      of L and U) is being done, then Q(J) is initially
C       |      nonzero if M'(K,J) is nonzero; since values need
C       |      not be stored, each entry points to the next
C       |      nonzero; for example, if N=9 and the 5th row of
C       |      M' is
C       |      0 x x 0 x 0 0 x 0,

```

```

C      |      then Q will initially be
C      |      a 3 5 a 8 a a 10 a 2      (a - arbitrary);
C      |      Q(N+1) points to the first nonzero in the row and
C      |      the last nonzero points to N+1; as the algorithm
C      |      proceeds, other elements of Q are inserted in the
C      |      list because of fillin.
C      |      Size = N+1.
C nia | IM - at each step in the factorization, IM(1) is the last
C      |      element in the Ith row of U which needs to be
C      |      considered in computing fillin.
C      |      Size = N.
C
C Internal variables--
C      JLPTR - points to the last position used in JL.
C      JUPTR - points to the last position used in JU.
C
C      INTEGER R(1), IC(1), IA(1), JA(1), IL(1), JL(1),
C      *      IU(1), JU(1), Q(1), IM(1), FLAG, QM, VJ
C
C ***** Initialize pointers *****
C      JLPTR = 0
C      IL(1) = 1
C      JUPTR = 0
C      IU(1) = 1
C
C ***** For each row of L and U *****
C      DO 10 K=1,N
C ***** Set Q to the reordered row of A *****
C      Q(N+1) = N+1
C      JMIN = IA(R(K))
C      JMAX = IA(R(K)+1) - 1
C      IF (JMIN.GT.JMAX) GO TO 101
C      DO 2 J=JMIN,JMAX
C      VJ = IC(JA(J))
C      QM = N+1
C      1 M = QM
C      QM = Q(M)
C      IF (QM.LT.VJ) GO TO 1
C      IF (QM.EQ.VJ) GO TO 102
C      Q(M) = VJ
C      Q(VJ) = QM
C      2 CONTINUE
C
C ***** For each entry in the lower triangle *****
C      I = N+1
C      3 I = Q(I)
C      IF (I.GE.K) GO TO 7
C ***** L(K,I) will be nonzero, so add it to JL *****
C      JLPTR = JLPTR+1
C      IF (JLPTR.GT.JLMAX) GO TO 03
C      JL(JLPTR) = I
C      QM = I

```

```

C ***** Inspect Ith row for fillin, adjust LM if possible *****
      JMIN = IU(I)
      JMAX = IM(I)
      IF (JMIN.GT.JMAX) GO TO 6
      DO 5 J=JMIN,JMAX
        VJ = JU(J)
        IF (VJ.EQ.K) IM(I) = J
4       M = QM
        QM = Q(M)
        IF (QM.LT.VJ) GO TO 4
        IF (QM.EQ.VJ) GO TO 5
        Q(M) = VJ
        Q(VJ) = QM
        QM = VJ
5       CONTINUE
6       GO TO 3

C ***** Check for null pivot *****
C 7       IF (I.NE.K) GO TO 105
C ***** Remaining elements of Q define structure of U(K, ) *****
8       I = Q(I)
        IF (I.GT.N) GO TO 9
        JUPTR = JUPTR+1
        IF (JUPTR.GT.JUMAX) GO TO 106
        JU(JUPTR) = I
        GO TO 8

C ***** Get ready for next row *****
9       IM(K) = JUPTR
        IL(K+1) = JLPTR+1
10      IU(K+1) = JUPTR+1
C
        FLAG = 0
        RETURN

C
C ** ERROR: Null Row in A
101     FLAG = N + R(K)
        RETURN
C ** ERROR: Duplicate Entry in A
102     FLAG = 2*N + R(K)
        RETURN
C ** ERROR: Insufficient Storage for JL
103     FLAG = 3*N + K
        RETURN
C ** ERROR: Null Pivot
105     FLAG = 5*N + K
        RETURN
C ** ERROR: Insufficient Storage for JU
106     FLAG = 6*N + K
        RETURN
      END

```

```

C
C -----
C
C*** Subroutine NNF
C*** Numeric LDU-factorization of sparse nonsymmetric matrix and
C      solution of system of linear equations (uncompressed pointer
C      storage)
C
      SUBROUTINE NNF
        * (N, R,C,IC, IA,JA,A, Z, B, IL,JL,L,LMAX, D, IU,JU,U,UMAX,
        *   ROW, TMP, FLAG)
C
C      Input variables:  N, R,C,IC, IA,JA,A, B, IL,JL,LMAX, IU,JU,UMAX
C      Output variables: Z, L,D,U, FLAG
C
C      Parameters used internally:
C fia | ROW - holds intermediate values in calculation of L, D, U.
C      |      Size = N.
C fia | TMP - holds new right-hand side b' for solution of the
C      |      equation  $Ux = b'$ .
C      |      Size = N.
C
      INTEGER R(1), C(1), IC(1), IA(1), JA(1),
        * IL(1), JL(1), LMAX, IU(1), JU(1), UMAX, FLAG
      REAL A(1), Z(1), B(1), L(1), D(1), U(1), ROW(1), TMP(1), LI
C
C ***** Check storage *****
      IF (IL(N+1)-1 .GT. LMAX) GO TO 104
      IF (IU(N+1)-1 .GT. UMAX) GO TO 107
C
C ***** For each row *****
      DO 10 K=1,N
C ***** Set the initial structure of ROW *****
        JMIN = IL(K)
        JMAX = IL(K+1) - 1
        IF (JMIN.GT.JMAX) GO TO 2
C ***** If L(K,M) .NE. 0, ROW(M)=0 *****
        DO 1 J=JMIN,JMAX
          1 ROW(JL(J)) = 0
          2 ROW(K) = 0
          JMIN = IU(K)
          JMAX = IU(K+1) - 1
          IF (JMIN.GT.JMAX) GO TO 4
C ***** If U(K,M) .NE. 0, ROW(M)=0 *****
          DO 3 J=JMIN,JMAX
            3 ROW(JU(J)) = 0
            4 JMIN = IA(R(K))
            JMAX = IA(R(K)+1) - 1
C ***** Set ROW to Kth row of reordered A *****
            DO 5 J=JMIN,JMAX
              5 ROW(IC(JA(J))) = A(J)
C ***** Initialize SUM *****
            SUM = B(R(K))

```

```

C
C ***** Assign the Kth row of L and adjust ROW, SUM *****
      IMIN = IL(K)
      IMAX = IL(K+1) - 1
      IF (IMIN.GT.IMAX) GO TO 8
      DO 7 I=IMIN,IMAX
        LI = - ROW(JL(I))
C ***** If L is not required, then comment out the following line **
        L(I) = - LI
        SUM = SUM + LI * TMP(JL(I))
        JMIN = IU(JL(I))
        JMAX = IU(JL(I)+1) - 1
        IF (JMIN.GT.JMAX) GO TO 7
        DO 6 J=JMIN,JMAX
          ROW(JU(J)) = ROW(JU(J)) + LI * U(J)
6      CONTINUE
7
C
C ***** Assign diagonal D and Kth row of U, set TMP(K) *****
8      IF (ROW(K).EQ.0) GO TO 108
      DK = 1 / ROW(K)
      D(K) = DK
      TMP(K) = SUM * DK
      JMIN = IU(K)
      JMAX = IU(K+1) - 1
      IF (JMIN.GT.JMAX) GO TO 10
      DO 9 J=JMIN,JMAX
9      U(J) = ROW(JU(J)) * DK
10     CONTINUE
C
C ***** Solve  $Ux = TMP$  by back substitution *****
      K = N
      DO 13 I=1,N
        SUM = TMP(K)
        JMIN = IU(K)
        JMAX = IU(K+1) - 1
        IF (JMIN.GT.JMAX) GO TO 12
        DO 11 J=JMIN,JMAX
11         SUM = SUM - U(J) * TMP(JU(J))
12         TMP(K) = SUM
        Z(C(K)) = SUM
13         K = K-1
C
      FLAG = 0
      RETURN
C
C ** ERROR: Insufficient Storage for L
104     FLAG = 4*N + 1
      RETURN
C ** ERROR: Insufficient Storage for U
107     FLAG = 7*N + 1
      RETURN
C ** ERROR: Zero Pivot
108     FLAG = 8*N + K
      RETURN
      END

```

```

C
C -----
C
C*** Subroutine NNS
C*** Numeric solution of a sparse nonsymmetric system of linear
C      equations given LDU-factorization (uncompressed pointer storage)
C
C      SUBROUTINE NNS
C      *      (N, R,C, IL,JL,L, D, IU,JU,U, Z, B, TMP)
C
C      Input variables:  N, R,C, IL,JL,L, D, IU,JU,U, B
C      Output variables: Z
C
C      Parameters used internally:
C fia | TMP - holds new right-hand side b' for solution of the
C      |      equation  $Ux = b'$ .
C      |      Size = N.
C
C      INTEGER R(1), C(1), IL(1), JL(1), IU(1), JU(1)
C      REAL L(1), D(1), U(1), Z(1), B(1), TMP(1)
C
C ***** Solve  $LDy = b$  by forward substitution *****
C      DO 2 K=1,N
C          SUM = B(R(K))
C          JMIN = IL(K)
C          JMAX = IL(K+1) - 1
C          IF (JMIN.GT.JMAX) GO TO 2
C          DO 1 J=JMIN,JMAX
C              SUM = SUM - L(J) * TMP(JL(J))
C          1      TMP(K) = SUM * D(K)
C
C ***** Solve  $Ux = y$  by back substitution *****
C      K = N
C      DO 5 I=1,N
C          SUM = TMP(K)
C          JMIN = IU(K)
C          JMAX = IU(K+1) - 1
C          IF (JMIN.GT.JMAX) GO TO 4
C          DO 3 J=JMIN,JMAX
C              SUM = SUM - U(J) * TMP(JU(J))
C          3      TMP(K) = SUM
C          4      Z(C(K)) = SUM
C          5      K = K-1
C      RETURN
C      END

```

```

C                                     Appendix 2                                7/31/77
C
C      Subroutines for Solving Sparse Nonsymmetric Systems
C      of Linear Equations (Track Nonzeroes Dynamically)
C
C
C*** Subroutine TDRV
C*** Driver for subroutine for solving sparse nonsymmetric systems of
C      linear equations (track nonzeroes dynamically)
C
C      SUBROUTINE TDRV
C      *      (N, R, IC, IA, JA, A, B, Z, NSP, ISP, RSP, ESP, FLAG)
C
C      PARAMETERS
C      Class abbreviations are --
C      n - INTEGER variable
C      f - REAL variable
C      v - supplies a VALUE to the driver
C      r - returns a RESULT from the driver
C      i - used INTERNALLY by the driver
C      a - ARRAY
C
C      Class | Parameter
C      -----
C      |
C      The nonzero entries of the coefficient matrix M are stored
C      row-by-row in the array A. To identify the individual nonzero
C      entries in each row, we need to know in which column each entry
C      lies. The column indices which correspond to the nonzero entries
C      of M are stored in the array JA; i.e., if  $A(K) = M(I, J)$ , then
C       $JA(K) = J$ . In addition, we need to know where each row starts and
C      how long it is. The index positions in JA and A where the rows of
C      M begin are stored in the array IA; i.e., if  $M(I, J)$  is the first
C      nonzero entry (stored) in the I-th row and  $A(K) = M(I, J)$ , then
C       $IA(I) = K$ . Moreover, the index in JA and A of the first location
C      following the last element in the last row is stored in  $IA(N+1)$ .
C      Thus, the number of entries in the I-th row is given by
C       $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C      consecutively in
C       $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C      and the corresponding column indices are stored consecutively in
C       $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C      For example, the 5 by 5 matrix
C      ( 1. 0. 2. 0. 0.)
C      ( 0. 3. 0. 0. 0.)
C      M = ( 0. 4. 5. 6. 0.)
C      ( 0. 0. 0. 7. 0.)
C      ( 0. 0. 0. 8. 9.)
C      would be stored as
C      | 1 2 3 4 5 6 7 8 9
C      -----
C      IA | 1 3 4 7 8 10
C      JA | 1 3 2 2 3 4 4 4 5
C      A  | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C      nv | N      - number of variables/equations.
C      fva | A      - nonzero entries of the coefficient matrix M, stored
C      |           by rows.
C      |           Size = number of nonzero entries in M.
C      nva | IA     - pointers to delimit the rows in A.

```

```

C      |      Size = N+1.
C nva  | JA    - column numbers corresponding to the elements of A.
C      |      Size = size of A.
C fva  | B     - right-hand side b; B and Z can be the same array.
C      |      Size = N.
C fra  | Z     - solution x; B and Z can be the same array.
C      |      Size = N.
C
C      The rows and columns of the original matrix M can be
C      reordered (e.g., to reduce fillin or ensure numerical stability)
C      before calling the driver. If no reordering is done, then set
C      R(I) = C(I) = IC(I) = I for I=1,...,N. The solution Z is returned
C      in the original order.
C
C nva  | R     - ordering of the rows of M.
C      |      Size = N.
C nva  | IC    - inverse of the ordering of the columns of M; i.e.,
C      |      IC(C(I)) = I for I=1,...,N, where C is the
C      |      ordering of the columns of M.
C      |      Size = N.
C
C      Various errors are detected by the driver and the individual
C      subroutines.
C
C nr    | FLAG  - error flag; values and their meanings are --
C      |      0      No Errors Detected
C      |      N+K    Null Row in A -- Row = K
C      |      2N+K   Duplicate Entry in A -- Row = K
C      |      5N+K   Null Pivot -- Row = K
C      |      8N+K   Zero Pivot -- Row = K
C      |      10N+1  Insufficient Storage in TDRV
C      |      12N+K  Insufficient Storage in TRK
C
C      Working storage is needed for the factored form of the matrix
C      M plus various temporary vectors. The arrays ISP and RSP should be
C      the same; integer storage is allocated from the beginning of ISP
C      and real storage from the end of RSP.
C
C nv    | NSP   - declared dimension of ISP and RSP; NSP generally must
C      |      be larger than  $6N+2 + 2*K$  (where  $K$  = (number of
C      |      nonzero entries in the upper triangle of M)).
C nvira | ISP   - integer working storage divided up into various arrays
C      |      needed by the subroutines; ISP and RSP should be
C      |      the same array.
C      |      Size = NSP.
C fvira | RSP   - real working storage divided up into various arrays
C      |      needed by the subroutines; ISP and RSP should be
C      |      the same array.
C      |      Size = NSP.
C nr    | ESP   - if NSP is sufficiently large to allocate space, then
C      |      ESP is set to the amount of excess storage provided.
C
C      INTEGER R(1), IC(1), IA(1), JA(1), ISP(1), ESP, FLAG,
C      *      U, ROW, TMP, Q
C      REAL A(1), B(1), Z(1), RSP(1)

```

```

C
C ***** Initialize and divide up temporary storage *****
      IJU = 1
      IU  = IJU + N
      Q   = IU + N+1
      IM  = Q  + N+1
      JU  = IM + N
      U   = JU
      ROW = NSP - N
      TMP = ROW - N
      MAX = TMP - JU
      IF (MAX.LT.0) GO TO 110
C
C ***** Call zero-tracking subroutine *****
      FLAG = 0
      CALL TRK
      *      (N, R, IC, IA, JA, A, Z, B,
      *      ISP(IJU), ISP(JU), ISP(IU), RSP(U), MAX,
      *      ISP(Q),  ISP(IM), RSP(ROW), RSP(TMP), FLAG, ESP)
      IF (FLAG.NE.0) GO TO 100
      RETURN
C
C ** ERROR: Error Detected in TRK
100  RETURN
C ** ERROR: Insufficient Storage
110  FLAG = 10*N + 1
      RETURN
      END

```

```

D
C -----
C
C YALE SPARSE MATRIX PACKAGE - ZERO-TRACKING CODE
C SOLVING THE SYSTEM OF EQUATIONS  $Mx = b$ 
C
C I. SUBROUTINE NAMES
C TRK performs an LDU-decomposition of the matrix M, without
C storing L or D, and solves the linear system of equations.
C
C II. CALLING SEQUENCES
C The coefficient matrix can be processed by an ordering routine
C (e.g., to reduce fillin or ensure numerical stability) before using
C the remaining subroutines. If no reordering is done, then set
C  $R(I) = C(I) = IC(I) = I$  for  $I=1, \dots, N$ . The calling sequence is --
C ( (matrix ordering))
C TRK (solution of linear system of equations)
C (If several systems with the same coefficient matrix but different
C right-hand sides or several systems whose coefficient matrices have
C the same nonzero structure are to be solved, and sufficient space
C is available, other subroutines should be used.)
C
C III. STORAGE OF SPARSE MATRICES
C The nonzero entries of the coefficient matrix M are stored
C row-by-row in the array A. To identify the individual nonzero
C entries in each row, we need to know in which column each entry
C lies. The column indices which correspond to the nonzero entries
C of M are stored in the array JA; i.e., if  $A(K) = M(I, J)$ , then
C  $JA(K) = J$ . In addition, we need to know where each row starts and
C how long it is. The index positions in JA and A where the rows of
C M begin are stored in the array IA; i.e., if  $M(I, J)$  is the first
C nonzero entry (stored) in the I-th row and  $A(K) = M(I, J)$ , then
C  $IA(I) = K$ . Moreover, the index in JA and A of the first location
C following the last element in the last row is stored in  $IA(N+1)$ .
C Thus, the number of entries in the I-th row is given by
C  $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C consecutively in
C  $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C and the corresponding column indices are stored consecutively in
C  $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C For example, the 5 by 5 matrix
C ( 1. 0. 2. 0. 0.)
C ( 0. 3. 0. 0. 0.)
C  $M = ( 0. 4. 5. 6. 0.)$ 
C ( 0. 0. 0. 7. 0.)
C ( 0. 0. 0. 8. 9.)
C would be stored as
C | 1 2 3 4 5 6 7 8 9
C -----
C IA | 1 3 4 7 8 10
C JA | 1 3 2 2 3 4 4 4 5
C A | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C The strict upper triangular portion of the matrix U is stored
C in a similar fashion using the arrays IU, JU, U, except that an
C additional array IJU is used to compress storage of JU by allowing
C some of the column indices to be used for more than one row.
C IJU(K) points to the starting location in JU of entries for the Kth
C row. Compression in JU occurs in two ways. First, if a row I was
C merged into the current row K, and the number of elements merged in
C from (the tail portion of) row I is the same as the final length of

```

C row K, then the Kth row and the tail of row I are identical and
 C IJU(K) may point to the start of the tail. Second, if some tail
 C portion of the (K-1)st row is identical to the head of the Kth row,
 C then IJU(K) may point to the start of that tail portion. For
 C example, the nonzero structure of the strict upper triangular part
 C of the matrix

```

C      d 0 x x x
C      0 d 0 x x
C      0 0 d x 0
C      0 0 0 d x
C      0 0 0 0 d
  
```

C would be represented as

```

C      | 1 2 3 4 5 6
C      -----
C      IU | 1 4 6 7 8 8
C      JU | 3 4 5 4
C      IJU | 1 2 4 3
  
```

C IV. ADDITIONAL STORAGE SAVINGS

C JU and U should be the same array. TRK fills JU from the
 C beginning of the array and U from the end of the array.

C R and IC can be the same array in the calling sequence if no
 C reordering of the coefficient matrix has been done. Z and ROW can
 C be the same array.

C V. PARAMETERS

C Following is a list of parameters to TRK. Class abbreviations
 C are --

```

C      n - INTEGER variable
C      f - REAL variable
C      v - supplies a VALUE to a subroutine
C      r - returns a RESULT from a subroutine
C      i - used INTERNALLY by a subroutine
C      a - ARRAY
  
```

C Class | Parameter

Class	Parameter	Description
fva	A	- nonzero entries of the coefficient matrix M, stored by rows. Size = number of nonzero entries in M.
fva	B	- right-hand side b. Size = N.
nr	ESP	- if enough storage was provided for JU and U, then ESP is set to amount of excess storage provided.
nr	FLAG	- error flag; values and their meanings are -- 0 No Errors Detected N+K Null Row in A -- Row = K 2N+K Duplicate Entry in A -- Row = K 5N+K Null Pivot -- Row = K 8N+K Zero Pivot -- Row = K 12N+K Insufficient Storage for JU/U -- Row = K
nva	IA	- pointers to delimit the rows in A. Size = N+1.
nva	IC	- inverse of the ordering of the columns of M; i.e., IC(C(I)) = I for I=1,...,N, where C is the

```

C          ordering of the columns of M.
C          Size = N.
C nia      IJU - pointers to the first element in each row in JU, used
C            to compress storage in JU.
C            Size = N.
C nia      IU  - pointers to delimit the rows in U.
C            Size = N+1.
C nva      JA  - column numbers corresponding to the elements of A.
C            Size = size of A.
C nia      JU  - column numbers corresponding to the elements of U;
C            JU and U should be the same array.
C            Size = MAX.
C nv       MAX - declared dimension of JU and U; MAX must be larger
C            than the size of U (the number of nonzero entries
C            in the strict upper triangle of M plus fillin) plus
C            the size of JU (the size of U minus compression).
C nv       N   - number of variables/equations.
C nva      R   - ordering of the rows of M.
C            Size = N.
C fia      U   - nonzero entries in the strict upper triangular portion
C            of U, stored by rows; JU and U should be the same
C            array.
C            Size = MAX.
C fra      Z   - solution x.
C            Size = N.

```

C
C
C
C*** Subroutine TRK

C*** Numerical solution of sparse nonsymmetric system of linear
C equations (track zeroes dynamically)

C

SUBROUTINE TRK

```

*      (N, R, IC, IA, JA, A, Z, B, IJU, JU, IU, U, MAX,
*      Q, IM, ROW, TMP, FLAG, ESP)

```

C

Input variables: N, R, IC, IA, JA, A, B, MAX

C

Output variables: Z, FLAG

C

Parameters used internally:

C nia

Q - suppose M' is the result of reordering M; if
processing of the Kth row of M' (hence the Kth rows
of L and U) is being done, then Q(J) is initially
nonzero if M'(K,J) is nonzero; since values need
not be stored, each entry points to the next
nonzero; for example, if N=9 and the 5th row of
M' is

0 x x 0 x 0 0 x 0,

then Q will initially be

a 3 5 a 8 a a 10 a 2 (a - arbitrary);

Q(N+1) points to the first nonzero in the row and
the last nonzero points to N+1; as the algorithm
proceeds, other elements of Q are inserted in the
list because of fillin.

Size = N+1.

C nia

IM - at each step in the factorization, IM(I) is the last
element of the Ith row of U which needs to be
considered in computing fillin.

C

Size = N.

```

C fia | ROW - holds intermediate values in calculation of U.
C      |      Size = N.
C fia | TMP - holds new right-hand side b' for solution of the
C      |      equation  $Ux = b'$ .
C      |      Size = N.
C
      INTEGER R(1), IC(1), IA(1), JA(1),
      * IJU(1), JU(1), IU(1), Q(1), IM(1), FLAG, ESP, VJ, QM
      REAL A(1), Z(1), B(1), U(1), ROW(1), TMP(1)
C
C ***** Initialize *****
JUMIN = 1
JUMAX = 0
IU(1) = MAX
C
C ***** For each row *****
DO 20 K=1,N
C ***** Initialize Q and ROW to the Kth row of reordered A *****
      LUK = 0
      Q(N+1) = N+1
      JMIN = IA(R(K))
      JMAX = IA(R(K)+1) - 1
      IF (JMIN.GT.JMAX) GO TO 101
      DO 2 J=JMIN,JMAX
        VJ = IC(JA(J))
        QM = N+1
1      M = QM
        QM = Q(M)
        IF (QM.LT.VJ) GO TO 1
        IF (QM.EQ.VJ) GO TO 102
        LUK = LUK+1
        Q(M) = VJ
        Q(VJ) = QM
        ROW(VJ) = A(J)
2      CONTINUE
C
C ***** Link through Q *****
      LMAX = 0
      IJU(K) = JUMAX
      I = N+1
3      I = Q(I)
      LUK = LUK-1
      IF (I.GE.K) GO TO 8
      QM = I
      JMIN = IJU(I)
      JMAX = IM(I)
      LUI = 0
      IF (JMIN.GT.JMAX) GO TO 7
C ***** and find nonzero structure of Kth row of L and U *****
      DO 5 J=JMIN,JMAX
        VJ = JU(J)
        IF (VJ.GT.K) LUI = LUI+1
4      M = QM
        QM = Q(M)
        IF (QM.LT.VJ) GO TO 4
        IF (QM.EQ.VJ) GO TO 5
        LUK = LUK+1
        Q(M) = VJ
        Q(VJ) = QM
        ROW(VJ) = 0
        QM = VJ
5      CONTINUE

```

```

C ***** Adjust IJU and IM *****
      JTMP = JMAX - LUI
      IF (LUI.LE.LMAX) GO TO 6
      LMAX = LUI
      IJU(K) = JTMP+1
6      IF (JTMP.LT.JMIN) GO TO 7
      IF (JU(JTMP).EQ.K) IM(I) = JTMP
7      GO TO 3
C
C ***** See if JU storage can be compressed *****
8      IF (I.NE.K) GO TO 105
      IF (LUK.EQ.LMAX) GO TO 14
      IF (JUMIN.GT.JUMAX) GO TO 12
      I = Q(K)
      DO 9 JMIN=JUMIN,JUMAX
        IF (JU(JMIN)-I) 9, 10, 12
9      CONTINUE
      GO TO 12
10     IJU(K) = JMIN
      DO 11 J=JMIN,JUMAX
        IF (JU(J).NE.I) GO TO 12
        I = Q(I)
        IF (I.GT.N) GO TO 14
11     CONTINUE
      JUMAX = JMIN - 1
C ***** Store pointers in JU *****
12     JUMIN = JUMAX + 1
      JUMAX = JUMAX + LUK
      IF (JUMAX.GT.IU(K)) GO TO 112
      I = K
      DO 13 J=JUMIN,JUMAX
        I = Q(I)
13     JU(J) = I
      IJU(K) = JUMIN
14     IU(K+1) = IU(K) - LUK
      IF (JUMAX.GT.IU(K+1)) GO TO 112
      IM(K) = IJU(K) + LUK - 1
C
C ***** Calculate numerical values for Kth row *****
      SUM = B(R(K))
      I = N+1
15     I = Q(I)
      IF (I.GE.K) GO TO 18
      AKI = - ROW(I)
      SUM = SUM + AKI * TMP(I)
      JMIN = IU(I+1) + 1
      JMAX = IU(I)
      IF (JMIN.GT.JMAX) GO TO 17
      MU = IJU(I) - JMIN
      DO 16 J=JMIN,JMAX
16     ROW(JU(MU+J)) = ROW(JU(MU+J)) + AKI * U(J)
17     GO TO 15

```

```

C ***** Store values in TMP and U *****
18  IF (ROW(K).EQ.0) GO TO 108
    DK = 1 / ROW(K)
    TMP(K) = SUM * DK
    JMIN = IU(K+1) + 1
    JMAX = IU(K)
    IF (JMIN.GT.JMAX) GO TO 20
    MU = IJU(K) - JMIN
    DO 19 J=JMIN,JMAX
19      U(J) = ROW(JU(MU+J)) * DK
20  CONTINUE
    ESP = IU(N+1) - JMAX
C
C ***** Solve Ux = TMP by back substitution *****
K = N
DO 23 I=1,N
    SUM = TMP(K)
    JMIN = IU(K+1) + 1
    JMAX = IU(K)
    IF (JMIN.GT.JMAX) GO TO 22
    MU = IJU(K) - JMIN
    DO 21 J=JMIN,JMAX
21      SUM = SUM - U(J) * TMP(JU(MU+J))
22      TMP(K) = SUM
23      K = K-1
    DO 24 K=1,N
24      Z(K) = TMP(IC(K))
C
    FLAG = 0
    RETURN
C
C ** ERROR: Null Row in A
101  FLAG = N + R(K)
    RETURN
C ** ERROR: Duplicate Entry in A
102  FLAG = 2*N + R(K)
    RETURN
C ** ERROR: Null Pivot
105  FLAG = 5*N + K
    RETURN
C ** ERROR: Zero Pivot
108  FLAG = 8*N + K
    RETURN
C ** ERROR: Insufficient Storage for JU and U
112  FLAG = 12*N + K
    RETURN
END

```

```

C
C
C      Subroutines for Solving Sparse Nonsymmetric Systems
C      of Linear Equations (Compressed Pointer Storage)
C
C
C*** Subroutine CDRV
C*** Driver for subroutines for solving sparse nonsymmetric systems of
C      linear equations (compressed pointer storage)
C
C      SUBROUTINE CDRV
C      *      (N, R,C,IC, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, PATH, FLAG)
C
C      PARAMETERS
C      Class abbreviations are--
C      n - INTEGER variable
C      f - REAL variable
C      v - supplies a VALUE to the driver
C      r - returns a RESULT from the driver
C      i - used INTERNALLY by the driver
C      a - ARRAY
C
C      Class | Parameter
C      -----
C      |
C      The nonzero entries of the coefficient matrix M are stored
C      row-by-row in the array A. To identify the individual nonzero
C      entries in each row, we need to know in which column each entry
C      lies. The column indices which correspond to the nonzero entries
C      of M are stored in the array JA; i.e., if  $A(K) = M(I,J)$ , then
C       $JA(K) = J$ . In addition, we need to know where each row starts and
C      how long it is. The index positions in JA and A where the rows of
C      M begin are stored in the array IA; i.e., if  $M(I,J)$  is the first
C      nonzero entry (stored) in the I-th row and  $A(K) = M(I,J)$ , then
C       $IA(I) = K$ . Moreover, the index in JA and A of the first location
C      following the last element in the last row is stored in  $IA(N+1)$ .
C      Thus, the number of entries in the I-th row is given by
C       $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C      consecutively in
C       $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C      and the corresponding column indices are stored consecutively in
C       $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C      For example, the 5 by 5 matrix
C      ( 1. 0. 2. 0. 0.)
C      ( 0. 3. 0. 0. 0.)
C      M = ( 0. 4. 5. 6. 0.)
C      ( 0. 0. 0. 7. 0.)
C      ( 0. 0. 0. 8. 9.)
C      would be stored as
C      | 1 2 3 4 5 6 7 8 9
C      -----
C      IA | 1 3 4 7 8 10
C      JA | 1 3 2 2 3 4 4 4 5
C      A  | 1. 2. 3. 4. 5. 6. 7. 8. 9.
C
C      nv | N      - number of variables/equations.
C      fva | A      - nonzero entries of the coefficient matrix M, stored
C      |          by rows.
C      |          Size = number of nonzero entries in M.
C      nva | IA     - pointers to delimit the rows in A.

```

```

C      |      Size = N+1.
C nva  | JA   - column numbers corresponding to the elements of A.
C      |      Size = size of A.
C nva  | B    - right-hand side b; B and Z can be the same array.
C      |      Size = N.
C fra  | Z    - solution x; B and Z can be the same array.
C      |      Size = N.
C
C      The rows and columns of the original matrix M can be
C reordered (e.g., to reduce fillin or ensure numerical stability)
C before calling the driver. If no reordering is done, then set
C  $R(I) = C(I) = IC(I) = I$  for  $I=1, \dots, N$ . The solution Z is returned
C in the original order.
C      If the columns have been reordered (i.e.,  $C(I).NE.I$  for some
C I), then the driver will call a subroutine (NROC) which rearranges
C each row of JA and A, leaving the rows in the original order, but
C placing the elements of each row in increasing order with respect
C to the new ordering. If  $PATH.NE.1$ , then NROC is assumed to have
C been called already.
C
C nva  | R    - ordering of the rows of M.
C      |      Size = N.
C nva  | C    - ordering of the columns of M.
C      |      Size = N.
C nva  | IC   - inverse of the ordering of the columns of M; i.e.,
C      |       $IC(C(I)) = I$  for  $I=1, \dots, N$ .
C      |      Size = N.
C
C      The solution of the system of linear equations is divided into
C three stages --
C      NSFC -- The matrix M is processed symbolically to determine where
C              fillin will occur during the numeric factorization.
C      NNFC -- The matrix M is factored numerically into the product LDU
C              of a unit lower triangular matrix L, a diagonal matrix
C              D, and a unit upper triangular matrix U, and the system
C               $Mx = b$  is solved.
C      NNSC -- The linear system  $Mx = b$  is solved using the LDU
C              factorization from NNFC.
C      For several systems whose coefficient matrices have the same
C      nonzero structure, NSFC need be done only once (for the first
C      system); then NNFC is done once for each additional system. For
C      several systems with the same coefficient matrix, NSFC and NNFC
C      need be done only once (for the first system); then NNSC is done
C      once for each additional right-hand side.
C
C nv   | PATH - path specification; values and their meanings are --
C      |      1 perform NROC, NSFC, and NNFC.
C      |      2 perform NNFC only (NSFC is assumed to have been
C      |         done in a manner compatible with the storage
C      |         allocation used in the driver).
C      |      3 perform NNSC only (NSFC and NNFC are assumed to
C      |         have been done in a manner compatible with the
C      |         storage allocation used in the driver).

```

```

C
C      Various errors are detected by the driver and the individual
C      subroutines.
C
C nr   | FLAG - error flag; values and their meanings are --
C       |      0      No Errors Detected
C       |      N+K     Null Row in A -- Row = K
C       |      2N+K    Duplicate Entry in A -- Row = K
C       |      3N+K     Insufficient Storage in NSFC -- Row = K
C       |      4N+1     Insufficient Storage in NNFC
C       |      5N+K     Null Pivot -- Row = K
C       |      6N+K     Insufficient Storage in NSFC -- Row = K
C       |      7N+1     Insufficient Storage in NNFC
C       |      8N+K     Zero Pivot -- Row = K
C       |      10N+1    Insufficient Storage in CDRV
C       |      11N+1    Illegal PATH Specification
C
C      Working storage is needed for the factored form of the matrix
C      M plus various temporary vectors. The arrays ISP and RSP should be
C      the same; integer storage is allocated from the beginning of ISP
C      and real storage from the end of RSP.
C
C nv   | NSP - declared dimension of ISP and RSP; NSP generally must
C       |       be larger than  $8N+2 + 2K$  (where  $K = (\text{number of}$ 
C       |       nonzero entries in  $M)$ ).
C nvira | ISP - integer working storage divided up into various arrays
C       |       needed by the subroutines; ISP and RSP should be
C       |       the same array.
C       |       Size = NSP.
C fvira | RSP - real working storage divided up into various arrays
C       |       needed by the subroutines; ISP and RSP should be
C       |       the same array.
C       |       Size = NSP.
C nr   | ESP - if sufficient storage was available to perform the
C       |       symbolic factorization (NSFC), then ESP is set to
C       |       the amount of excess storage provided (negative if
C       |       insufficient storage was available to perform the
C       |       numeric factorization (NNFC)).
C
C      INTEGER R(1), C(1), IC(1), IA(1), JA(1), ISP(1), ESP, PATH,
C      *      FLAG, TMP, D, Q, U, RMN, ADD, UMAX
C      REAL A(1), B(1), Z(1), RSP(1)
C
C      IF(PATH.LE.0 .OR. PATH.GT.3) GO TO 111
C***** Initialize and divide up temporary storage *****
C      FLAG = 0
C      IL = 1
C      IJL = IL + N + 1
C      IU = IJL + N
C      IJU = IU + N + 1
C      IRL = IJU + N
C      JRL = IRL + N
C      JL = JRL + N
C      IRA = NSP + 1 - N
C      D = IRA
C      JRA = D - N
C      TMP = JRA
C      Q = TMP - (N + 1)
C      JRU = Q - N
C      IRU = JRU - N
C      IF(JL .GE. IRU) GO TO 110
C      IF(PATH .GT. 1) GO TO 10

```

```

C
C***** Reorder A if necessary, call NSFC if flag is set *****
      RMN = IRU - JL
      ADD = RMN/2
      JU = JL + ADD
      JLMAX = ADD
      JUMAX = RMN - ADD
      DO 5 II=1,N
        IF(C(II) .NE. II) GO TO 6
5      CONTINUE
      GO TO 7
C
6      CALL NROC (N, IC, IA, JA, A, ISP(IL), RSP(Q), ISP(IU), FLAG)
      IF(FLAG .NE. 0) GO TO 100
C
7      CALL NSFC
      *      (N, R, IC, IA, JA, JLMAX, ISP(IL), ISP(JL), ISP(IJL), JUMAX,
      *      ISP(IU), ISP(JU), ISP(IJU), RSP(Q), RSP(IRA), RSP(JRA), Z,
      *      ISP(IRL), ISP(JRL), RSP(IRU), RSP(JRU), FLAG)
      IF(FLAG .NE. 0) GO TO 100
C***** See if enough space remains, move JU next to JL *****
10     JLMAX = ISP(IJL+N-1)
      JUMAX = ISP(IJU+N-1)
      LMAX = ISP(IL+N) - 1
      UMAX = ISP(IU+N) - 1
      IF(PATH .GT. 1) GO TO 20
      NEED = JLMAX + JUMAX + LMAX + UMAX
      RMN = RMN + 3*N + 1
      ESP = RMN - NEED
      IF(NEED .GT. RMN) GO TO 110
      JUOLD = JU - 1
      JU = JL + JLMAX - 1
      IF (JUMAX.LE.0) GO TO 20
      DO 15 II=1,JUMAX
15     ISP(JU+II) = ISP(JUOLD+II)
C***** Call remaining subroutines *****
20     JU = JL + JLMAX
      L = JU + JUMAX
      U = L + LMAX
C
      IF(PATH .EQ. 3) GO TO 30
      CALL NNFC
      *      (N, R, C, IC, IA, JA, A, LMAX, ISP(IU), ICF(JL), ISP(IJL), RSP(L),
      *      RSP(D), UMAX, ISP(IU), ISP(JU), ISP(IJU), RSP(U), Z, L,
      *      RSP(TMP), ISP(IRL), ISP(JRL), FLAG)
      IF(FLAG .NE. 0) GO TO 100
      RETURN
C
30     CALL NNSC
      *      (N, R, C, ISP(IL), ISP(JL), ISP(IJL), RSP(L), RSP(D), ISP(IU),
      *      ISP(JU), ISP(IJU), RSP(U), Z, B, RSP(TMP))
      RETURN
C
C ** ERROR: Error Detected in NROC, NSFC, NNFC, or NNSC
100    RETURN
C ** ERROR: Insufficient Storage
110    FLAG = 10*N + 1
      RETURN
C ** ERROR: Illegal PATH Specification
111    FLAG = 11*N + 1
      RETURN
      END

```

```

C
C
C -----
C
C      YALE SPARSE MATRIX PACKAGE - NONSYMMETRIC CODES
C      SOLVING THE SYSTEM OF EQUATIONS  $Mx = b$ 
C
C
C I.   SUBROUTINE NAMES
C      Subroutine names and functions are --
C      (1) NROC for reordering;
C      (2) NSFC for symbolic factorization;
C      (3) NNFC for numeric factorization and solution;
C      (4) NNSC for solution.
C
C II.  CALLING SEQUENCES
C      The coefficient matrix can be processed by an ordering routine
C      (e.g., to reduce fillin or ensure numerical stability) before using
C      the remaining subroutines. If no reordering is done, then set
C       $R(I) = C(I) = IC(I) = 1$  for  $I=1, \dots, N$ . If an ordering subroutine
C      is used, then NROC should be used to reorder the coefficient matrix
C      The calling sequence is --
C      (
C      (matrix ordering))
C      (NROC (matrix reordering))
C      NSFC (symbolic factorization to determine where fillin will
C            occur during numeric factorization)
C      NNFC (numeric factorization into product LDU of unit lower
C            triangular matrix L, diagonal matrix D, and unit
C            upper triangular matrix U, and solution of linear
C            system)
C      NNSC (solution of linear system for additional right-hand
C            side using LDU factorization from NNFC)
C      (If only one system of equations is to be solved, then the
C      subroutine TRK should be used.)
C
C III. STORAGE OF SPARSE MATRICES
C      The nonzero entries of the coefficient matrix M are stored
C      row-by-row in the array A. To identify the individual nonzero
C      entries in each row, we need to know in which column each entry
C      lies. The column indices which correspond to the nonzero entries
C      of M are stored in the array JA; i.e., if  $A(K) = M(I, J)$ , then
C       $JA(K) = J$ . In addition, we need to know where each row starts and
C      how long it is. The index positions in JA and A where the rows of
C      M begin are stored in the array IA; i.e., if  $M(I, J)$  is the first
C      (leftmost) entry in the I-th row and  $A(K) = M(I, J)$ , then
C       $IA(I) = K$ . Moreover, the index in JA and A of the first location
C      following the last element in the last row is stored in  $IA(N+1)$ .
C      Thus, the number of entries in the I-th row is given by
C       $IA(I+1) - IA(I)$ , the nonzero entries of the I-th row are stored
C      consecutively in
C       $A(IA(I)), A(IA(I)+1), \dots, A(IA(I+1)-1)$ ,
C      and the corresponding column indices are stored consecutively in
C       $JA(IA(I)), JA(IA(I)+1), \dots, JA(IA(I+1)-1)$ .
C      For example, the 5 by 5 matrix
C      ( 1. 0. 2. 0. 0.)
C      ( 0. 3. 0. 0. 0.)
C      M = ( 0. 4. 5. 6. 0.)
C          ( 0. 0. 0. 7. 0.)
C          ( 0. 0. 0. 8. 9.)
C      would be stored as
C      | 1 2 3 4 5 6 7 8 9
C      -----
C      IA | 1 3 4 7 8 10
C      JA | 1 3 2 2 3 4 4 4 5
C      A  | 1 2 3 4 5 6 7 8 9.

```

C The strict upper (lower) triangular portion of the matrix
 C U (L) is stored in a similar fashion using the arrays IU, JU, U
 C (IL, JL, L) except that an additional array IJU (IJL) is used to
 C compress storage of JU (JL) by allowing some of the column (row)
 C indices to be used for more than one row (column) (n.b., L is stored
 C by columns). IJU(K) (IJL(K)) points to the starting location in
 C JU (JL) of entries for the Kth row (column). Compression in JU
 C (JL) occurs in two ways. First, if a row (column) I was merged
 C into the current row (column) K, and the number of elements merged
 C in from (the tail portion of) row (column) I is the same as the
 C final length of row (column) K, then the Kth row (column) and the
 C tail of row (column) I are identical and IJU(K) (IJL(K)) may point
 C to the start of the tail. Second, if some tail portion of the
 C (K-1)st row (column) is identical to the head of the Kth row
 C (column), then IJU(K) (IJL(K)) may point to the start of that tail
 C portion. For example, the nonzero structure of the strict upper
 C triangular part of the matrix

```

C      d 0 x x x
C      0 d 0 x x
C      0 0 d x 0
C      0 0 0 d x
C      0 0 0 0 d
C
C      would be represented as
C      | 1 2 3 4 5 6
C      +-----+
C      IU | 1 4 6 7 8 8
C      JU | 3 4 5 4
C      IJU| 1 2 4 3
  
```

C The diagonal entries of L and U are assumed to be equal to one and
 C are not stored. The array D contains the reciprocals of the
 C diagonal entries of the matrix D.

C IV. ADDITIONAL STORAGE SAVINGS

C In NSFC, R and IC can be the same array in the calling
 C sequence if no reordering of the coefficient matrix has been done.
 C In NNFC, Z and ROW can be the same array. R, C and IC can all
 C be the same array if no reordering has been done. If only the
 C rows have been reordered, then C and IC can be the same array.
 C If the row and column orderings are the same, then R and C can be
 C the same array.

C In NNSC, R and C can be the same array if no reordering has
 C been done or if the row and column orderings are the same. Z and B
 C can be the same array; however, then B will be destroyed.

C V. PARAMETERS

C Following is a list of parameters to the programs. Names are
 C uniform among the various subroutines. Class abbreviations are --

```

C   n - INTEGER variable
C   f - REAL variable
C   v - supplies a VALUE to a subroutine
C   r - returns a RESULT from a subroutine
C   i - used INTERNALLY by a subroutine
C   a - ARRAY
  
```

C Class | Parameter

```

C   -----+-----
C   fva | A   - nonzero entries of the coefficient matrix M, stored
C       |   by rows.
C       |   Size = number of nonzero entries in M.
  
```

```

C fva | B - right-hand side b.
C      |   Size = N.
C nva | C - ordering of the columns of M.
C      |   Size = N.
C fvra | D - reciprocals of the diagonal entries of the matrix D.
C      |   Size = N.
C nr | FLAG - error flag; values and their meanings are --
C      |   0 No Errors Detected
C      |   N+K Null Row in A -- Row = K
C      |   2N+K Duplicate Entry in A -- Row = K
C      |   3N+K Insufficient Storage for JL -- Row = K
C      |   4N+1 Insufficient Storage for L
C      |   5N+K Null Pivot -- Row = K
C      |   6N+K Insufficient Storage for JU -- Row = K
C      |   7N+1 Insufficient Storage for U
C      |   8N+K Zero Pivot -- Row = K
C nva | IA - pointers to delimit the rows of A.
C      |   Size = N+1.
C nvra | IJL - pointers to the first element in each column in JL,
C      |   used to compress storage in JL.
C      |   Size = N.
C nvra | IJU - pointers to the first element in each row in JU, used
C      |   to compress storage in JU.
C      |   Size = N.
C nvra | IL - pointers to delimit the columns of L.
C      |   Size = N+1.
C nvra | IU - pointers to delimit the rows of U.
C      |   Size = N+1.
C nva | JA - column numbers corresponding to the elements of A.
C      |   Size = size of A.
C nvra | JL - row numbers corresponding to the elements of L.
C      |   Size = JLMAX.
C nv | JLMAX - declared dimension of JL; JLMAX must be larger than
C      |   the number of nonzeros in the strict lower triangle
C      |   of M plus fillin minus compression.
C nvra | JU - column numbers corresponding to the elements of U.
C      |   Size = JUMAX.
C nv | JUMAX - declared dimension of JU; JUMAX must be larger than
C      |   the number of nonzeros in the strict upper triangle
C      |   of M plus fillin minus compression.
C fvra | L - nonzero entries in the strict lower triangular portion
C      |   of the matrix L, stored by columns.
C      |   Size = LMAX.
C nv | LMAX - declared dimension of L; LMAX must be larger than
C      |   the number of nonzeros in the strict lower triangle
C      |   of M plus fillin (IL(N+1)-1 after NSFC).
C nv | N - number of variables/equations.
C nva | R - ordering of the rcws of M.
C      |   Size = N.
C fvra | U - nonzero entries in the strict upper triangular portion
C      |   of the matrix U, stored by rows.
C      |   Size = UMAX.
C nv | UMAX - declared dimension of U; UMAX must be larger than
C      |   the number of nonzeros in the strict upper triangle
C      |   of M plus fillin (IU(N+1)-1 after NSFC).
C fra | Z - solution x.
C      |   Size = N.
C
C
C

```

```

C
C*** Subroutine NROC
C*** Reorders rows of A, leaving row order unchanged
C
      SUBROUTINE NROC (N, IC, IA, JA, A, JAR, AR, P, FLAG)
C
C      Input parameters: N, IC, IA, JA, A
C      Output parameters: JA, A, FLAG
C
C      Parameters used internally:
C nia | P      - at the Kth step, P is a linked list of the reordered
C      |         column indices of the Kth row of A; P(N+1) points
C      |         to the first entry in the list.
C      |         Size = N+1.
C nia | JAR    - at the Kth step, JAR contains the elements of the
C      |         reordered column indices of A.
C      |         Size = N.
C fia | AR     - at the Kth step, AR contains the elements of the
C      |         reordered row of A.
C      |         Size = N.
C
      INTEGER IC(1), IA(1), JA(1), JAR(1), P(1), FLAG
      REAL A(1), AR(1)
C
C ***** For each nonempty row *****
      DO 5 K=1,N
        JMIN = IA(K)
        JMAX = IA(K+1) - 1
        IF(JMIN .GT. JMAX) GO TO 5
        P(N+1) = N + 1
C ***** Insert each element in the list *****
        DO 3 J=JMIN,JMAX
          NEWJ = IC(JA(J))
          I = N + 1
1         IF(P(I) .GE. NEWJ) GO TO 2
          I = P(I)
          GO TO 1
2         IF(P(I) .EQ. NEWJ) GO TO 102
          P(NEWJ) = P(I)
          P(I) = NEWJ
          JAR(NEWJ) = JA(J)
          AR(NEWJ) = A(J)
3         CONTINUE
C ***** Replace old row in JA and A *****
          I = N + 1
          DO 4 J=JMIN,JMAX
            I = P(I)
            JA(J) = JAR(I)
4           A(J) = AR(I)
5         CONTINUE
          FLAG = 0
          RETURN
C
C ** ERROR: Duplicate entry in A
102  FLAG = N + K
      RETURN
      END
C

```

```

C -----
C
C*** Subroutine NSFC
C*** Symbolic LDU-factorization of nonsymmetric sparse matrix
C      (compressed pointer storage)
C
      SUBROUTINE NSFC
        *      (N, R, IC, IA, JA, JLMAX, IL, JL, IJL, JUMAX, IU, JU, IJU, Q, IRA,
        *      JRA, IRAC, IRL, JRL, IRU, JRU, FLAG)
C
C      Input variables: N, R, IC, IA, JA, JLMAX, JUMAX.
C      Output variables: IL, JL, IJL, IU, JU, IJU, FLAG.
C
C      Parameters used internally:
C nia | Q      - Suppose M' is the result of reordering M. If
C      |         processing of the Ith row of M' (hence the Ith
C      |         row of U) is being done, Q(J) is initially
C      |         nonzero if M'(I,J) is nonzero (J.GE.I). Since
C      |         values need not be stored, each entry points to the
C      |         next nonzero and Q(N+1) points to the first. N+1
C      |         indicates the end of the list. For example, if N=9
C      |         and the 5th row of M' is
C      |         0 x 0 x 0 0 x 0
C      |         then Q will initially be
C      |         a a a 8 a a 10 5      (a - arbitrary).
C      |         As the algorithm proceeds, other elements of Q
C      |         are inserted in the list because of fillin.
C      |         Q is used in an analogous manner to compute the
C      |         Ith column of L.
C      |         Size = N+1.
C nia | IRA,   - vectors used to find the columns of M. At the Kth
C      |         step of the factorization, IRAC(K) points to the
C      |         head of a linked list in JRA of row indices I
C      |         such that I.GE. K and M(I,K) is nonzero. Zero
C      |         indicates the end of the list. IRA(I) (I.GE.K)
C      |         points to the smallest J such that J.GE. K and
C      |         M(I,J) is nonzero.
C      |         Size of each = N.
C nia | IRL,   - vectors used to find the rows of L. At the Kth step
C      |         of the factorization, JRL(K) points to the head
C      |         of a linked list in JRL of column indices J
C      |         such that J.LT. K and L(K,J) is nonzero. Zero
C      |         indicates the end of the list. IRL(J) (J.LT.K)
C      |         points to the smallest I such that I.GE. K and
C      |         L(I,J) is nonzero.
C      |         Size of each = N.
C nia | IRU,   - vectors used in a manner analogous to IRL and JRL
C      |         to find the columns of U.
C      |         Size of each = N.
C
C      Internal variables:
C      JLPTR - points to the last position used in JL.
C      JUPTR - points to the last position used in JU.
C      JMIN, JMAX - are the indices in A or U of the first and last
C      |         elements to be examined in a given row.
C      |         For example, JMIN=IA(K), JMAX=IA(K+1)-1.
C
      INTEGER CEND, QM, REND, RV, VJ
      INTEGER IA(1), JA(1), IRA(1), JRA(1), IL(1), JL(1), IJL(1)
      INTEGER IU(1), JU(1), IJU(1), IRL(1), JRL(1), IRU(1), JRU(1)
      INTEGER R(1), IC(1), Q(1), IRAC(1), FLAG

```

```

C
C ***** Initialize pointers *****
      NP1 = N + 1
      JLMIN = 1
      JLPTR = 0
      IL(1) = 1
      JUMIN = 1
      JUPTR = 0
      IU(1) = 1
      DO 1 K=1,N
        IRAC(K) = 0
        JRA(K) = 0
        JRL(K) = 0
        JRU(K) = 0
1
C ***** Initialize column pointers for A *****
      DO 2 K=1,N
        RK = R(K)
        IAK = IA(RK)
        IF (IAK .GE. IA(RK+1)) GO TO 101
        JAIK = IC(JA(IAK))
        IF (JAIK .GT. K) GO TO 105
        JRA(K) = IRAC(JAIK)
        IRAC(JAIK) = K
2
        IRA(K) = IAK
C
C ***** For each column of L and row of U *****
      DO 41 K=1,N
C
C ***** Initialize Q for computing Kth column of L *****
      Q(NP1) = NP1
      LUK = -1
C ***** by filling in Kth column of A *****
      VJ = IRAC(K)
      IF (VJ .EQ. 0) GO TO 5
3
      QM = NP1
4
      M = QM
      QM = Q(M)
      IF (QM .LT. VJ) GO TO 4
      IF (QM .EQ. VJ) GO TO 102
      LUK = LUK + 1
      Q(M) = VJ
      Q(VJ) = QM
      VJ = JRA(VJ)
      IF (VJ .NE. 0) GO TO 3
C ***** Link through JRU *****
5
      LASTID = 0
      LASTI = 0
      IJL(K) = JLPTR
      I = K
6
      I = JRU(I)
      IF (I .EQ. 0) GO TO 10
      QM = NP1
      JMIN = IRL(I)
      JMAX = IJL(I) + IL(I+1) - IL(I) - 1
      LONG = JMAX - JMIN
      JTMP = JL(JMIN)
      IF (JTMP .NE. K) LONG = LONG + 1
      IF (JTMP .EQ. K) R(I) = -R(I)
      IF (LASTID .GE. LONG) GO TO 7
      LASTI = I
      LASTID = LONG
7
      IF (LONG .LE. 0) GO TO 6

```

```

C ***** And merge the corresponding columns into the Kth column ****
      DO 9 J=JMIN,JMAX
        VJ = JL(J)
      8      M = QM
          QM = Q(M)
          IF (QM .LT. VJ) GO TO 8
          IF (QM .EQ. VJ) GO TO 9
          LUK = LUK + 1
          Q(M) = VJ
          Q(VJ) = QM
          QM = VJ
      9      CONTINUE
          GO TO 6
C ***** LASTI is the longest column merged into the Kth *****
C ***** See if it equals the entire Kth column *****
      10     QM = Q(NP1)
          IF (QM .NE. K) GO TO 105
          IF (LUK .EQ. 0) GO TO 17
          IF (LASTID .NE. LUK) GO TO 11
C ***** If so, JL can be compressed *****
          IRL = IRL(LASTI)
          IJL(K) = IRL + 1
          IF (JL(IRL) .NE. K) IJL(K) = IJL(K) - 1
          GO TO 17
C ***** If not, see if Kth column can overlap the previous one *****
      11     IF (JLMIN .GT. JLPTR) GO TO 15
          QM = Q(QM)
          DO 12 J=JLMIN,JLPTR
            IF (JL(J) - QM) 12, 13, 15
      12     CONTINUE
          GO TO 15
      13     IJL(K) = J
          DO 14 I=J,JLPTR
            IF (JL(I) .NE. QM) GO TO 15
            QM = Q(QM)
            IF (QM .GT. N) GO TO 17
      14     CONTINUE
          JLPTR = J - 1
C ***** Move column indices from Q to JL, update vectors *****
      15     JLMIN = JLPTR + 1
          IJL(K) = JLMIN
          IF (LUK .EQ. 0) GO TO 17
          JLPTR = JLPTR + LUK
          IF (JLPTR .GT. JLMAX) GO TO 103
          QM = Q(NP1)
          DO 16 J=JLMIN,JLPTR
            QM = Q(QM)
      16     JL(J) = QM
      17     IRL(K) = IJL(K)
          IL(K+1) = IL(K) + LUK
C
C ***** Initialize Q for computing Kth row of U *****
      Q(NP1) = NP1
      LUK = -1

```

```

C ***** by filling in Kth row of reordered A *****
RK = R(K)
JMIN = IRA(K)
JMAX = IA(RK+1) - 1
IF (JMIN .GT. JMAX) GO TO 20
DO 19 J=JMIN,JMAX
  VJ = IC(JA(J))
  QM = NP1
18  M = QM
  QM = Q(M)
  IF (QM .LT. VJ) GO TO 18
  IF (QM .EQ. VJ) GO TO 102
  LUK = LUK + 1
  Q(M) = VJ
  Q(VJ) = QM
19  CONTINUE
C ***** Link through JRL, *****
20  LASTID = 0
  LASTI = 0
  IJU(K) = JUPTR
  I = K
  I1 = JRL(K)
21  I = I1
  IF (I .EQ. 0) GO TO 26
  I1 = JRL(I)
  QM = NP1
  JMIN = IRU(I)
  JMAX = IJU(I) + IU(I+1) - IU(I) - 1
  LONG = JMAX - JMIN
  JTMP = JU(JMIN)
  IF (JTMP .EQ. K) GO TO 22
C ***** Update IRL and JRL, *****
  LONG = LONG + 1
  CEND = IJL(I) + IL(I+1) - IL(I)
  IRL(I) = IRL(I) + 1
  IF (IRL(I) .GE. CEND) GO TO 22
  J = JL(IRL(I))
  JRL(I) = JRL(J)
  JRL(J) = I
22  IF (LASTID .GE. LONG) GO TO 23
  LASTI = 1
  LASTID = LONG
23  IF (LONG .LE. 0) GO TO 21
C ***** And merge the corresponding rows into the Kth row *****
DO 25 J=JMIN,JMAX
  VJ = JU(J)
24  M = QM
  QM = Q(M)
  IF (QM .LT. VJ) GO TO 24
  IF (QM .EQ. VJ) GO TO 25
  LUK = LUK + 1
  Q(M) = VJ
  Q(VJ) = QM
  QM = VJ
25  CONTINUE
  GO TO 21
C ***** Update JRL(K) and IRL(K) *****
26  IF (IL(K+1) .LE. IL(K)) GO TO 27
  J = JL(IRL(K))
  JRL(K) = JRL(J)
  JRL(J) = K

```

```

C ***** LASTI is the longest row merged into the Kth *****
C ***** See if it equals the entire Kth row *****
27      QM = Q(NP1)
        IF (QM .NE. K) GO TO 105
        IF (LUK .EQ. 0) GO TO 34
        IF (LASTID .NE. LUK) GO TO 28
C ***** If so, JU can be compressed *****
        IRUL = IRU(LASTI)
        IJU(K) = IRUL + 1
        IF (JU(IRUL) .NE. K) IJU(K) = IJU(K) - 1
        GO TO 34
C ***** If not, see if Kth row can overlap the previous one *****
28      IF (JUMIN .GT. JUPTR) GO TO 32
        QM = Q(QM)
        DO 29 J=JUMIN,JUPTR
          IF (JU(J) - QM) 29, 30, 32
29      CONTINUE
        GO TO 32
30      IJU(K) = J
        DO 31 I=J,JUPTR
          IF (JU(I) .NE. QM) GO TO 32
          QM = Q(QM)
          IF (QM .GT. N) GO TO 34
31      CONTINUE
        JUPTR = J - 1
C ***** Move row indices from Q to JU, update vectors *****
32      JUMIN = JUPTR + 1
        IJU(K) = JUMIN
        IF (LUK .EQ. 0) GO TO 34
        JUPTR = JUPTR + LUK
        IF (JUPTR .GT. JUMAX) GO TO 106
        QM = Q(NP1)
        DO 33 J=JUMIN,JUPTR
          QM = Q(QM)
33      JU(J) = QM
34      IRU(K) = IJU(K)
        IU(K+1) = IU(K) + LUK
C
C ***** Update IRU, JRU *****
        I = K
35      I1 = JRU(I)
        IF (R(I) .LT. 0) GO TO 36
        REND = IJU(I) + IU(I+1) - IU(I)
        IF (IRU(I) .GE. REND) GO TO 37
        J = JU(IRU(I))
        JRU(I) = JRU(J)
        JRU(J) = I
        GO TO 37
36      R(I) = -R(I)
37      I = I1
        IF (I .EQ. 0) GO TO 38
        IRU(I) = IRU(I) + 1
        GO TO 35

```

```

C
C ***** Update IRA, JRA, IRAC *****
38   I = IRAC(K)
      IF (I .EQ. 0) GO TO 41
39   I1 = JRA(I)
      IRA(I) = IRA(I) + 1
      IF (IRA(I) .GE. IA(R(I)+1)) GO TO 40
      IRAI = IRA(I)
      JAIRAI = IC(JA(IRAI))
      IF (JAIRAI .GT. I) GO TO 40
      JRA(I) = IRAC(JAIRAI)
      IRAC(JAIRAI) = I
40   I = I1
      IF (I .NE. 0) GO TO 39
41   CONTINUE
C
      IJL(N) = JLPTR
      IJU(N) = JUPTR
      FLAG = 0
      RETURN
C
C ** ERROR: Null Row in A
101  FLAG = N + RK
      RETURN
C ** ERROR: Duplicate entry in A
102  FLAG = 2*N + RK
      RETURN
C ** ERROR: Insufficient Storage for JL
103  FLAG = 3*N + K
      RETURN
C ** ERROR: Null pivot
105  FLAG = 5*N + K
      RETURN
C ** ERROR: Insufficient Storage for JU
106  FLAG = 6*N + K
      RETURN
      END
C
C -----
C
C*** Subroutine NNFC
C*** Numerical LDU-factorization of sparse nonsymmetric matrix and
C      solution of system of linear equations (compressed pointer
C      storage)
C
C      SUBROUTINE NNFC
C      * (N, R, C, IC, IA, JA, A, LMAX, IL, JL, IJL, L, D, UMAX, IU, JU, IJU,
C      *   U, Z, B, ROW, TMP, IRL, JRL, FLAG)
C
C      Input variables: N, R, C, IC, IA, JA, A, B, IL, JL, IJL,
C                      LMAX, IU, JU, IJU, UMAX
C      Output variables: Z, L, D, U, FLAG

```

```

C
C      Parameters used internally:
C nia | IRL, - vectors used to find the rows of L. At the Kth step
C nia | JRL   of the factorization, JRL(K) points to the head
C      | of a linked list in JRL of column indices J
C      | such J.LT. K and L(K,J) is nonzero. Zero
C      | indicates the end of the list. IRL(J) (J.LT.K)
C      | points to the smallest I such that I.GE. K and
C      | L(I,J) is nonzero.
C      | Size of each = N.
C fia | ROW - holds intermediate values in calculation of U and L.
C      | Size = N.
C fia | TMP - holds new right-hand side b' for solution of the
C      | equation  $Ux = b'$ .
C      | Size = N.
C
C      Internal variables:
C      JMIN, JMAX - indices of the first and last positions in a row to
C                  be examined.
C      SUM - used in calculating TMP.
C
C      INTEGER RK,UMAX
C      REAL LKI
C      INTEGER R(1), C(1), IC(1), IA(1), JA(1), IL(1), JL(1), IJU(1)
C      INTEGER IU(1), JU(1), IJU(1), IRL(1), JRL(1), FLAG
C      REAL A(1), L(1), D(1), U(1), Z(1), B(1), ROW(1), TMP(1)
C
C ***** Initialize pointers and test storage *****
C IF(IL(N+1)-1.GT. LMAX) GO TO 104
C IF(IU(N+1)-1.GT. UMAX) GO TO 107
C DO 1 K=1,N
C   IRL(K) = IL(K)
C   JRL(K) = 0
C 1  CONTINUE
C
C ***** For each row *****
C DO 19 K=1,N
C ***** Reverse JRL and zero ROW where Kth row of L will fill in ***
C   ROW(K) = 0
C   I1 = 0
C   IF (JRL(K) .EQ. 0) GO TO 3
C   I = JRL(K)
C 2   I2 = JRL(I)
C   JRL(I) = I1
C   I1 = I
C   ROW(I) = 0
C   I = I2
C   IF (I .NE. 0) GO TO 2
C ***** Set ROW to zero where U will fill in *****
C 3   JMIN = IJU(K)
C   JMAX = JMIN + IU(K+1) - IU(K) - 1
C   IF (JMIN .GT. JMAX) GO TO 5
C   DO 4 J=JMIN,JMAX
C     ROW(JU(J)) = 0
C 4 ***** Place Kth row of A in ROW *****
C 5   RK = R(K)
C   JMIN = IA(RK)
C   JMAX = IA(RK+1) - 1
C   DO 6 J=JMIN,JMAX
C     ROW(IC(JA(J))) = A(J)
C 6   CONTINUE

```

```

C ***** Initialize SUM, and link through JRL *****
SUM = B(RK)
I = I1
IF (I .EQ. 0) GO TO 10
C ***** Assign the Kth row of L and adjust ROW, SUM *****
7 LKI = -ROW(I)
C ***** If L is not required, then comment out the following line **
L(IRL(I)) = -LKI
SUM = SUM + LKI * TMP(I)
JMIN = IU(I)
JMAX = IU(I+1) - 1
IF (JMIN .GT. JMAX) GO TO 9
MU = IJU(I) - JMIN
DO 8 J=JMIN, JMAX
8 ROW(JU(MU+J)) = ROW(JU(MU+J)) + LKI * U(J)
9 I = JRL(I)
IF (I .NE. 0) GO TO 7

C
C ***** Assign Kth row of U and diagonal D, set TMP(K) *****
10 IF (ROW(K) .EQ. 0) GO TO 108
DK = 1 / ROW(K)
D(K) = DK
TMP(K) = SUM * DK
IF (K .EQ. N) GO TO 19
JMIN = IU(K)
JMAX = IU(K+1) - 1
IF (JMIN .GT. JMAX) GO TO 12
MU = IJU(K) - JMIN
DO 11 J=JMIN, JMAX
11 U(J) = ROW(JU(MU+J)) * DK
12 CONTINUE

C
C ***** Update IRL and JRL, keeping JRL in decreasing order *****
13 I = I1
IF (I .EQ. 0) GO TO 18
14 IRL(I) = IRL(I) + 1
I1 = JRL(I)
IF (IRL(I) .GE. IL(I+1)) GO TO 17
IJLB = IRL(I) - IL(I) + IJL(I)
J = JL(IJLB)
15 IF (I .GT. JRL(J)) GO TO 16
J = JRL(J)
GO TO 15
16 JRL(I) = JRL(J)
JRL(J) = I
17 I = I1
IF (I .NE. 0) GO TO 14
18 IF (IRL(K) .GE. IL(K+1)) GO TO 19
J = JL(IJL(K))
JRL(K) = JRL(J)
JRL(J) = K
19 CONTINUE

```

```

C
C ***** Solve Ux = TMP by back substitution *****
      K = N
      DO 22 I=1,N
        SUM = TMP(K)
        JMIN = IU(K)
        JMAX = IU(K+1) - 1
        IF (JMIN .GT. JMAX) GO TO 21
        MU = IJU(K) - JMIN
        DO 20 J=JMIN,JMAX
          SUM = SUM - U(J) * TMP(IJU(MU+J))
        20
        21 TMP(K) = SUM
        Z(C(K)) = SUM
        22 K = K-1
      FLAG = 0
      RETURN
C
C ** ERROR: Insufficient Storage for L
104 FLAG = 4*N + 1
      RETURN
C ** ERROR: Insufficient Storage for U
107 FLAG = 7*N + 1
      RETURN
C ** ERROR: Zero Pivot
108 FLAG = 8*N + K
      RETURN
      END
C
C -----
C
C *** Subroutine NNSC
C *** Numerical solution of sparse nonsymmetric system of linear
C equations given LDU-factorization (compressed pointer storage)
C
      SUBROUTINE NNSC
        * (N, R, C, IL, JL, IJL, L, D, IU, JU, IJU, U, Z, B, TMP)
C
C Input variables: N, R, C, IL, JL, IJL, L, D, IU, JU, IJU, U, B
C Output variables: Z
C
C Parameters used internally:
C fia | TMP - temporary vector which gets result of solving Ly = b.
C | Size = N.
C
C Internal variables:
C JMIN, JMAX - indices of the first and last positions in a row of
C U or L to be used.
C
      INTEGER R(1), C(1), IL(1), JL(1), IJL(1), IU(1), JU(1), IJU(1)
      REAL L(1), D(1), U(1), B(1), Z(1), TMP(1)

```

```

C
C ***** Set TMP to reordered B *****
DO 1 K=1,N
1   TMP(K) = B(R(K))
C ***** Solve Ly = b by forward substitution *****
DO 3 K=1,N
   JMIN = IL(K)
   JMAX = IL(K+1) - 1
   TMPK = -D(K) * TMP(K)
   TMP(K) = -TMPK
   IF (JMIN .GT. JMAX) GO TO 3
   ML = IJL(K) - JMIN
   DO 2 J=JMIN,JMAX
2      TMP(JL(ML+J)) = TMP(JL(ML+J)) + TMPK * L(J)
3   CONTINUE
C ***** Solve Ux = y by back substitution *****
K = N
DO 6 I=1,N
   SUM = -TMP(K)
   JMIN = IU(K)
   JMAX = IU(K+1) - 1
   IF (JMIN .GT. JMAX) GO TO 5
   MU = IJU(K) - JMIN
   DO 4 J=JMIN,JMAX
4      SUM = SUM + U(J) * TMP(JU(MU+J))
5   TMP(K) = -SUM
   Z(C(K)) = -SUM
   K = K - 1
6   CONTINUE
RETURN
END

```

```

C
C
C      Test Driver for Sparse Nonsymmetric Matrix Package
C
C
C*** Program NTST
C*** Test Driver for Nonsymmetric Codes in Yale Sparse Matrix Package
C
C  Variables:
C
C      NG  - size of grid used to generate test problem.
C
C      N   - number of variables and equations (= NG x NG).
C
C      IA  - INTEGER one-dimensional array used to store row pointers
C            to JA and A; DIMENSION = N+1.
C
C      JA  - INTEGER one-dimensional array used to store column
C            indices of nonzero elements of M; DIMENSION = number of
C            nonzero entries in M.
C
C      A   - REAL one-dimensional array used to store nonzero elements
C            of M; DIMENSION = number of nonzero entries in M.
C
C      X   - REAL one-dimensional array used to store solution x;
C            DIMENSION = N.
C
C      B   - REAL one-dimensional array used to store right-hand-side b
C            DIMENSION = N.
C
C      P   - INTEGER one-dimensional array used to store permutation of
C            rows and columns for reordering linear system;
C            DIMENSION = N.
C
C      IP  - INTEGER one-dimensional array used to store inverse of
C            permutation stored in P; DIMENSION = N.
C
C      NSP - declared dimension of one-dimensional arrays ISP and RSP.
C
C      ISP - INTEGER one-dimensional array used as working storage
C            (equivalenced to RSP); DIMENSION = NSP.
C
C      RSP - REAL one-dimensional array used as working storage
C            (equivalenced to ISP); DIMENSION = NSP.
C
C      ESP - INTEGER amount of excess storage available
C
C
C      INTEGER IA(101), JA(500), P(100), IP(100), ISP(1500), ESP,
*      CASE, PATH, FLAG, APTR, VP, VQ, X, XMIN, XMAX, Y, YMIN, YMAX
      REAL A(500), Z(100), B(100), RSP(1500), NAME(3)
      EQUIVALENCE (ISP(1), RSP(1))
      DATA NSP/1500/, EPS/1E-5/,
*      NAME(1)/'N'/, NAME(2)/'T'/, NAME(3)/'C'/
C
C      INDEX(I,J) = NG*I + J - NG
C
C      NG = 3
C      N = NG*NG
C

```

```

C ***** CASE=1 => NDRV, CASE=2 => TDRV, CASE=3 => CDRV *****
DO 5 CASE=1,3
C
C ***** Set up matrix for five-point finite difference operator *****
APTR = 1
DO 2 I=1,NG
DO 2 J=1,NG
VP = INDEX (I, J)
P(VP) = VP
IP(VP) = VP
IA(VP) = APTR
SUM = 0
XMIN = MAXO (I, I-1)
XMAX = MINO (NG, I+1)
YMIN = MAXO (J, J-1)
YMAX = MINO (NG, J+1)
DO 1 X=XMIN,XMAX
DO 1 Y=YMIN,YMAX
IF ((X-I) * (Y-J) .NE. 0) GO TO 1
VQ = INDEX(X, Y)
JA(APTR) = VQ
A(APTR) = 8
IF (VP .LT. VQ) A(APTR) = -1
IF (VP .GT. VQ) A(APTR) = -2
SUM = SUM + A(APTR) * VQ
APTR = APTR + 1
1 CONTINUE
B(VP) = SUM
2 CONTINUE
IA(N+1) = APTR
NZA = IA(N+1) - 1
C
C ***** Output original array A *****
IF (CASE.EQ.1) PRINT 1001, NG,NG
1001 FORMAT (/ ' *** FIVE-POINT OPERATOR ON ',
* I1, ' BY ' I1, ' GRID ')
IF (CASE.EQ.1) PRINT 1002, (IA(I),I=1,N), IA(N+1)
1002 FORMAT (/ ' COEFFICIENT MATRIX: ' /
* IA (INDICES OF FIRST ELEMENTS IN ROWS) '
* /(10I5))
IF (CASE.EQ.1) PRINT 1003, (I,JA(I),A(I), I=1,NZA)
1003 FORMAT (/ ' JA A
* I COLUMN INDICES MATRIX '
* /(I3, I10, F16.5))
IF (CASE.EQ.1) PRINT 1004, (B(I), I=1,N)
1004 FORMAT (/ ' RIGHT HAND SIDE B: '
* /(5F10.5))
C
C ***** Call ODRV *****
FLAG = 0
PATH = 1
CALL ODRV
* (N, IA,JA,A, P,IP, NSP,RSP, PATH, FLAG)
IF (FLAG.NE.0) GO TO 101

```

```

C
C ***** Output ordering of variables/equations *****
      IF (CASE.EQ.1) PRINT 1005, (I,P(I),IP(I), I=1,N)
1005  FORMAT (' ROW/COLUMN ORDERING FROM ODRV: ' /
*      /'          P          IP          '
*      /' 1 ROW/COL ORDERING    INVERSE ORDERING '
*      /(I3, I10, I20))
C
C ***** Call NDRV / TDRV / CDRV *****
      PATH = 1
      IF (CASE.EQ.1) CALL NDRV
*      (N, P,P,IP, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, PATH, FLAG)
      IF (CASE.EQ.2) CALL TDRV
*      (N, P,IP, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, FLAG)
      IF (CASE.EQ.3) CALL CDRV
*      (N, P,P,IP, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, PATH, FLAG)
      IF (FLAG.EQ.0) GO TO 3
      PRINT 1006, NAME(CASE), FLAG
1006  FORMAT (' ERROR IN ', A1, 'DRV: FLAG = ', I5)
      GO TO 5
C
C ***** Calculate error *****
      3  SUM = 0
      DO 4 I=1,N
      4  SUM = SUM + ((Z(I)-I)/I)**2
      RMS = SQRT(SUM/N)
C
C ***** Output solution and error measure *****
      PRINT 1007, NAME(CASE), (Z(I), I=1,N)
1007  FORMAT (' SOLUTION FROM ', A1, 'DRV: '
*      /(5F10.5))
C
      IF (RMS.LE.EPS) PRINT 1008, RMS
1008  FORMAT (' SOLUTION CORRECT: RMS ERROR = ', 1PE8.2)
      IF (RMS.GT.EPS) PRINT 1009, RMS
1009  FORMAT (' SOLUTION INCORRECT: RMS ERROR = ', 1PE8.2)
C
      PRINT 1010, ESP
1010  FORMAT (' EXTRA STORAGE AVAILABLE = ', I4)
C
      5  CONTINUE
      STOP
C
C ***** Error messages *****
101  PRINT 1013, FLAG
1013  FORMAT (' ERROR IN ODRV: FLAG = ', I5)
      STOP
      END

```

Appendix 5

Sample Output From Test Driver

*** FIVE-POINT OPERATOR ON 3 BY 3 GRID

COEFFICIENT MATRIX:

IA (INDICES OF FIRST ELEMENTS IN ROWS)

1 4 8 11 15 20 24 27 31 34

I	JA COLUMN INDICES	A MATRIX
1	1	8.00000
2	2	-1.00000
3	4	-1.00000
4	1	-2.00000
5	2	8.00000
6	3	-1.00000
7	5	-1.00000
8	2	-2.00000
9	3	8.00000
10	6	-1.00000
11	1	-2.00000
12	4	8.00000
13	5	-1.00000
14	7	-1.00000
15	2	-2.00000
16	4	-2.00000
17	5	8.00000
18	6	-1.00000
19	8	-1.00000
20	3	-2.00000
21	5	-2.00000
22	6	8.00000
23	9	-1.00000
24	4	-2.00000
25	7	8.00000
26	8	-1.00000
27	5	-2.00000
28	7	-2.00000
29	8	8.00000
30	9	-1.00000
31	6	-2.00000
32	8	-2.00000
33	9	8.00000

RIGHT HAND SIDE B:

2.00000	6.00000	14.00000	18.00000	14.00000
23.00000	40.00000	31.00000	44.00000	

ROW/COLUMN ORDERING FROM ODRV:

	P	IP
I	ROW/COL ORDERING	INVERSE ORDERING
1	1	1
2	3	7
3	7	2
4	9	8
5	6	6
6	5	5
7	2	3
8	4	9
9	8	4

SOLUTION FROM NDRV:

1.00000	2.00000	3.00000	4.00000	5.00000
6.00000	7.00000	8.00000	9.00000	

SOLUTION CORRECT: RMS ERROR = 6.36E-09

EXTRA STORAGE AVAILABLE = 1384

SOLUTION FROM TDRV:

1.00000	2.00000	3.00000	4.00000	5.00000
6.00000	7.00000	8.00000	9.00000	

SOLUTION CORRECT: RMS ERROR = 6.36E-09

EXTRA STORAGE AVAILABLE = 1412

SOLUTION FROM CDRV:

1.00000	2.00000	3.00000	4.00000	5.00000
6.00000	7.00000	8.00000	9.00000	

SOLUTION CORRECT: RMS ERROR = 6.36E-09

EXTRA STORAGE AVAILABLE = 1364