Yale Sparse Matrix Package

I. The Symmetric Codes[1]

S. C. Eisenstat,[2] M. C. Gursky,[3]

M. H. Schultz,[2] and A. H. Sherman[4]

Research Report #112

[1]This research was supported in part by ONR Grant N00014-76-0277, NSF Grant MCS 76-11460, AFOSR Grant F49620-77-C-0037, and the Chevron Oil Field Research Company.

[2]Department of Computer Science, Yale University,

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

[4]Department of Computer Sciences, The University of Texas at Austin.

## 1. Introduction

Consider the NxN system of linear equations

(1) $M\, x = b,$

where the coefficient matrix M is large, sparse, symmetric, and positive definite. Such systems arise frequently in scientific computation, e.g., in finite difference and finite element approximations to elliptic boundary value problems. In this report, we present a package of efficient, reliable, well-documented, and portable FORTRAN subroutines for solving these systems.

Direct methods for solving (1) are generally variations of symmetric Gaussian elimination. We form the $U^t DU$ decomposition of A, where U is unit upper triangular and D positive diagonal, and then successively solve the triangular systems

(2) $U^t\, y = b,\quad D\, z = y,\quad U\, x = z.$

When M is large (N >> 1), (dense) Gaussian elimination is prohibitively expensive in terms of both the work ($\sim 1/3\ N^3$ multiplies) and storage ($N^2$ words) required. But, since M is sparse, most entries of M and U are zero and there are significant advantages to factoring M without storing or operating on the zeroes appearing in M and U. Recently, a number of implementations of sparse Gaussian elimination have appeared based on this idea, cf. [2,5,6,7].

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 [4,9]. In section 4, we illustrate the performance of the package on a typical model problem. Listings of the ordering subroutines and the subroutines for factoring and solving sparse symmetric positive definite systems appear as Appendices 1 and 2. Appendix 3 contains a test driver, a sample output of which appears as Appendix 4.

## 2. A Sparse Matrix Storage Scheme

Since the coefficient matrix M and the upper triangular factor U are large and sparse, it is inefficient to store them as dense matrices. Instead, we store matrices using a row-by-row storage scheme used in previous implementations of sparse symmetric Gaussian elimination, cf. [1,4].

This scheme 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 indices in JA and A where each row of M begins, 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), and the nonzero entries of the $I^{th}$ row are stored consecutively in

A(IA(I)), A(IA(I)+1), ..., A(IA(I+1)-1)

and the corresponding column indices are stored consecutively in

JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).

For example, the 5x5 matrix

$$M = \begin{bmatrix} 1 & 0 & 2 & 3 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 2 & 0 & 5 & 6 & 0 \\ 3 & 0 & 6 & 7 & 8 \\ 0 & 0 & 0 & 8 & 9 \end{bmatrix}$$

is stored as

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IA | 1 | 4 | 5 | 8 | 12 | 13 | | | | | | | |
| JA | 1 | 3 | 4 | 2 | 1 | 3 | 4 | 1 | 3 | 4 | 5 | 4 | 5 |
| A | 1 | 2 | 3 | 4 | 2 | 5 | 6 | 3 | 6 | 7 | 8 | 8 | 9 . |

Since the matrix M is symmetric, it suffices to store only the nonzero entries in the diagonal and strict upper triangle of M. The storage scheme used is the same as before (except that nonzero entries in the strict lower triangle of M are ignored), and our example matrix would be stored as

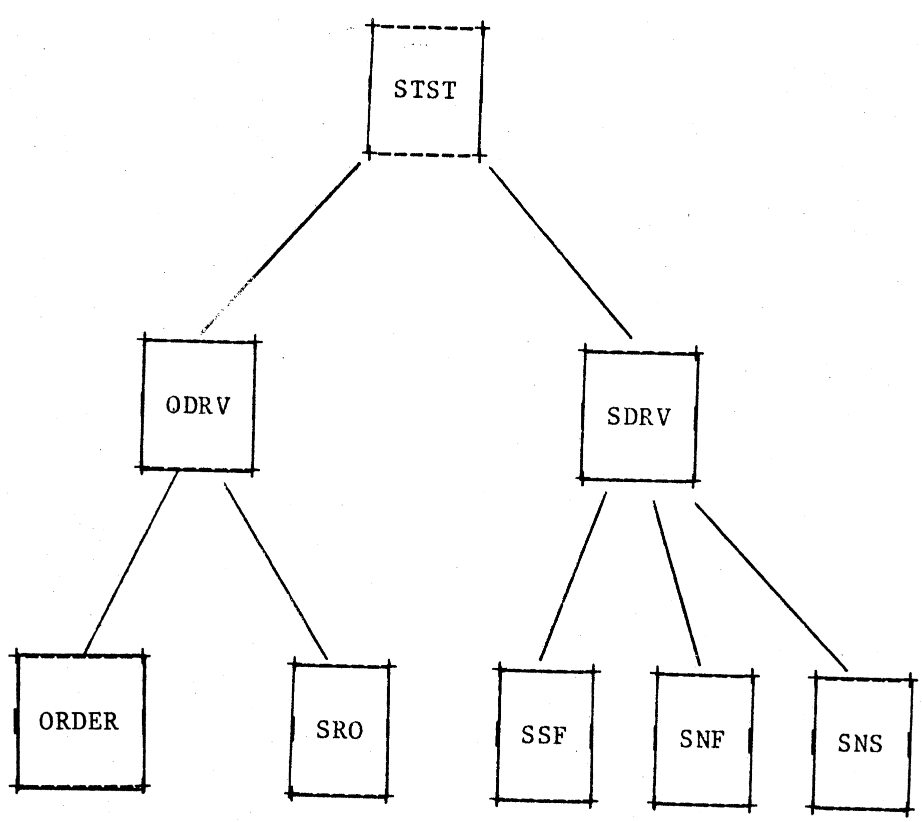
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| IA | 1 | 4 | 5 | 7 | 9 | 10 | | | |
| JA | 1 | 3 | 4 | 2 | 3 | 4 | 4 | 5 | 5 |
| A | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

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 (the diagonal and strict upper triangle of) M.

## 3. A Sparse Symmetric Matrix Package

The package consists of three drivers and five subroutines (see Figure 1). The ordering driver (subroutine ODRV) may be used to symmetrically reorder the variables and equations so as to reduce the total work (i.e. the number of multiplies) and storage required. The solution driver (subroutine SDRV) is used to solve the (permuted) system of linear equations. The test driver (program STST) sets up a model sparse symmetric positive definite system of linear equations, calls ODRV to reorder the variables and equations, and calls SDRV to solve the linear system. In the remainder of this section, we describe each of these routines in somewhat greater detail. The codes themselves are extensively documented; for further details about the algorithms employed, see [4,9].

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



A. <u>The Ordering Driver (ODRV)</u>

The work and storage required to solve a large sparse system of linear equations clearly depend upon the zero-nonzero structure of the coefficient matrix. But since this matrix is symmetric and positive definite, we could equally well solve the permuted system

$$(3) \qquad Q\,M\,Q^t\,y = Q\,b, \qquad Q\,x = y$$

given any permutation matrix Q. The permuted system corresponds to symmetrically reordering the variables and equations of the original system, and the net result can often be a significant reduction in the work and storage required to form the $U^tDU$ decomposition of A [3].

The ordering driver (subroutine ODRV) uses the important heuristic, the minimum degree algorithm (implemented in subroutine ORDER), to select Q. ORDER does a symbolic elimination on the nonzero structure of the system. At each step, it chooses a pivot element from among those uneliminated diagonal matrix entries which require the fewest arithmetic operations to eliminate (ties are broken arbitrarily). This has the effect of locally optimizing the elimination process with respect to the number of arithmetic operations performed. See [8,9] for more details.

ORDER returns two one-dimensional INTEGER arrays of length N: P contains the permutation of the row and column indices of M, i.e., the sequence of pivots; and IP contains the inverse permutation, i.e., IP(P(I)) = I for I = 1,2,...,N. If only the upper triangle of M is being stored, then the representation of M (i.e., the arrays IA, JA, and A) must be rearranged using the subroutine SRO (PATH=2 in ODRV).

The user may bypass ODRV entirely by setting P(I) = IP(I) = I for I = 1,2,3,...,N. Alternately, the user may substitute another ordering subroutine for ORDER, as long as it produces the two permutations P and IP. But again, if only the upper triangle of M is being stored, the representation of M must be rearranged using SRO.

B. The Solution Driver (SDRV)

The solution driver (subroutine SDRV) is used to solve the (permuted) linear system. Following Chang [1], SDRV breaks the solution process into three steps: symbolic factorization (subroutine SSF), numerical factorization (subroutine SNF), and back-solution (subroutine SNS). First, SSF determines the nonzero structure of the rows of U from the nonzero structure of the rows of M. Second, SNF uses the structure information generated by SSF to compute the $U^tDU$ factorization of M. Third, SNS computes the solution x from the factorization generated by SNF and the right-hand side b.

By splitting up the computation, we have gained flexibility. To solve a single system of equations, it suffices to use SSF, SNF, and SNS (PATH=1 in SDRV). To solve several systems in which the coefficient matrices have the same nonzero structure, it suffices to use SSF only once and then to use SNF and SNS for each system (PATH=2). To solve several systems with the same coefficient matrix but different right hand sides, it suffices to use SSF and SNF only once and then use SNS for each system (PATH=3).

C. <u>The Test Driver (STST)</u>

The test driver (program STST) 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 the standard five-point finite difference approximation on a 3x3 grid to the Poisson equation and chooses the right-hand side so that the solution vector x is (1,2,3,4,5,6,7,8,9). Since M is symmetric, we can specify either the entire matrix (CASE=1) or only the upper triangle (CASE=2). STST then calls ODRV to reorder the variables and equations, and SDRV to solve the linear system. At each stage, the values of all relevant variables are printed out. A sample output appears as Appendix 4.

## 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 I and II, we present the time and storage required to solve the familiar five-point finite difference equations on an n x n grid for several values of n. These computations were performed in single precision on an IBM 370/158 using the FORTRAN X optimizing compiler.

Table I

Time Required

Five-Point Operator on an n x n Grid

| Grid | STST | ORDER | SRO | SSF | SNF | sec/* | SNS | Total |
|---|---|---|---|---|---|---|---|---|
| 20 | 0.083 | 0.657 | 0.043 | 0.100 | 0.407 | 14.016 | 0.087 | 0.593 |
| 30 | 0.190 | 1.750 | 0.100 | 0.257 | 1.430 | 13.002 | 0.230 | 1.917 |
| 40 | 0.340 | 3.656 | 0.177 | 0.503 | 3.700 | 12.449 | 0.470 | 4.673 |

Table II

Storage Required

Five-Point Operator on an n x n Grid

| Grid | A, JA | U | JU | Total | Mults. |
|---|---|---|---|---|---|
| 20 | 1,160 | 3,368 | 1,889 | 7,259 | 35,195 |
| 30 | 2,640 | 9,456 | 4,538 | 18,496 | 127,666 |
| 40 | 4,720 | 19,926 | 8,423 | 36,351 | 334,937 |

## 5. References

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

[2] A. R. Curtis and J. K. Reid
Two FORTRAN Subroutines for Direct Solution of Linear Equations Whose Matrix is Sparse, Symmetric, and Positive Definite. Harwell Report AERE-R7119, 1972.

[3] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman.
Application of sparse matrix methods to partial differential equations. Proceedings of AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania 1975, pp. 40-45.

[4] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman.
Efficient implementation of sparse symmetric Gaussian elimination. Proceedings of AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania 1975, pp.33-39.

[5] 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.

[6] John E. Key

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

[7] 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.

[8] D. J. Rose.

A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. Read, editor, Graph Theory and Computing, Academic Press, 1972, pp. 183-217.

[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.

# Appendix 1

## Sparse Matrix Reordering Routines

```
C*** Subroutine ODRV
C*** Driver for sparse matrix reordering routines
C
        SUBROUTINE  ODRV
     *     (N, IA,JA,A, P,IP, NSP,ISP, PATH, FLAG)
C
C    PARAMETERS
C    Class abbreviations:
C       v - supplies a VALUE to the driver
C       r - contains a RESULT returned by the driver
C       i - is used INTERNALly by the driver
C       a - is an ARRAY
C       n - is an INTEGER variable
C       f - is a REAL variable.
C
C Class | Parameter
C ------+----------
C       |
C         The nonzero entries of the matrix  M  are stored row by row
C    in the array  A.  The array  JA  contains the corresponding column
C    indices;  i.e. if  A(K) = M(I,J),  then  JA(K) = J.  The array  IA
C    contains pointers to delimit the rows of  M;   IA(I)  is the index
C    in  JA and A  of the first entry stored in the Ith row of  M.  For
C    example, the symmetric 5 by 5 matrix
C       1  0  2  3  0
C       0  4  0  0  0
C       2  0  5  6  0
C       3  0  6  7  8
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  4  5  7  9 10
C        JA | 1  3  4  2  3  4  4  5  5
C         A | 1  2  3  4  5  6  7  8  9          .
C
C    IA(I+1) - IA(I)  is the number of nonzero entries in row I, so
C    IA(N+1), where N is the number of rows in  M, is needed to
C    determine the length of the Nth row in  A.
C         If  M  is a symmetric matrix only the elements of the upper
C    triangle (including the diagonal) need be stored in  A, although
C    the full matrix may be stored.
C
C vn    | N      - the number of rows/columns in matrix  M.
C vfa   | A      - coefficient matrix for the system of linear equations
C       |            Mx = b, stored in compressed form.
C       |            Size = number of nonzeros in  M  (or only in upper
C       |                   triangle of  M  for symmetric matrix).
C vna   | IA     - pointers to first elements of each row in  A.
C       |            Size = N+1.
C vna   | JA     - the column numbers corresponding to elements of  A.
C       |            Size = size of  A.
```

```
C
C        A minimum degree ordering of the matrix is done (ORDER).  If
C    M  is symmetric and only the upper triangle is being stored, the
C    array  A  needs to be reordered.
C
C vn     | PATH  - information on which subroutines are to be called.
C        |            Values and meanings of  PATH  are:
C        |                1  perform minimum degree ordering only.
C        |                2  perform minimum degree ordering and
C        |                   reordering of symmetric matrix.  This
C        |                   value should be passed only if  M  is
C        |                   symmetric and only the nonzeros of the
C        |                   upper triangle of  M  are being
C        |                   stored.  If the nonzeros of the entire
C        |                   matrix are being stored,  PATH  should
C        |                   equal 1.
C rn     | FLAG  - flag for error return from subroutines.  Error values
C        |            and their meanings are:
C        |                0       no error
C        |                N+I     null row in  A -- I
C        |               9N+I     ORDER  storage exceeded on row I
C        |              10N+1     ISP  too small to allocate space
C        |              11N+1     PATH  out of bounds
C
C        The result of the ordering algorithm is a permutation of the
C    row numbers of  M  and the inverse of the permutation.  The order
C    of the columns is the same as for the rows.
C
C rna    | IP    - inverse of the ordering of the rows/columns of  M.
C        |            Size = N.
C rna    | P     - ordering of the rows/columns of  M.
C        |            Size = N.
C
C        Workspace is needed to hold the temporary vectors used in the
C    ordering routine.
C
C rn     | NSP   - dimensioned size of  ISP.  NSP  must generally be at
C        |            least  2*(number of pairs (I,J) such that  M(I,J)
C        |            or M(J,I) is nonzero) + N  for the ordering routine
C        |            and at least  N + size of A  for the symmetric
C        |            reordering routine.
C fa     | ISP   - storage space divided up for various arrays of
C        |            the subroutines.
C
      INTEGER  IA(1), JA(1),  P(1), IP(1),  PATH,  FLAG,  VV, TMP, Q
      REAL  A(1),  ISP(1)
C
      IF (PATH.LT.1 .OR. PATH.GT.2)  GO TO 111
C******  Allocate space for ordering subroutine  *********************
      MAX = NSP/2
      VV = 1
      LV = VV + MAX
      IF (MAX.LT.N)  GO TO 110
C******  Call minimum degree ordering routine  **********************
      FLAG = 0
      CALL  ORDER
     *   (N,  IA, JA,  P, IP,  MAX, ISP(VV), ISP(LV),  FLAG)
      IF (FLAG.NE.0)  GO TO 100
```

```
C
C******  Allocate space, call symmetric reorder routine  **************
        IF (PATH.LT.2)  GO TO 1
          TMP = 1
          Q = TMP + N
          IF (NSP+1-Q .LT. IA(N+1)-1)  GO TO 110
          CALL SRO
     *       (N,  IP,  IA, JA, A,  ISP(TMP),  ISP(Q))
  1     RETURN
C
C ** ERROR:  Error Detected in ORDER
 100    RETURN
C ** ERROR:  Insufficient Storage
 110    FLAG = 10*N + 1
        RETURN
C ** ERROR:  Illegal PATH Specified
 111    FLAG = 11*N + 1
        RETURN
        END
C
C       ------------------------------------------------------------
C
C*** Subroutine ORDER
C*** Minimum degree ordering algorithm with threshhold search
C
        SUBROUTINE  ORDER
     *     (N, IA,JA, P,IP, MAX, VV,LV, FLAG)
C
C       Input variables:   N, IA,JA, MAX
C       Output variables:  P,IP, FLAG
C
C       Parameters used internally:
C nia   | VV  -  value field of a linked list describing adjacencies of
C       |          vertices.
C       |          Size .ge. number of pairs (I,J) such that  M(I,J)  or
C       |                    M(J,I)  is nonzero.
C nia   | LV  -  link field of the linked list.
C       |          Size = size of  VV.
C nv    | MAX -  dimensioned size of  VV  and  LV.
C
        INTEGER  IA(1), JA(1),  P(1), IP(1),  VV(1), LV(1),  FLAG,
     *    DMIN, DTHR,  S, SFS,  TMP,  VI, VJ, VK, VL
C
C * Initialize free storage  ****************************************
        VK = 1
        IF (MAX.LT.N)  GO TO 109
        DO 1 S=N,MAX
   1      LV(S) = S+1
        LV(MAX) = 0
        SFS = 1
C * Initialize ordering, degree, and adjacency ***********************
        DO 2 K=1,N
          VK = P(K)
          IP(VK) = K
          VV(K) = N+1
   2      LV(K) = K
        SFS = SFS + N
```

```
C
C * Initialize nonzero structure ***************************************
C * For every vertex VK ************************************************
C *** For every vertex VJ adjacent to VK *****************************
      DO 8 VK=1,N
        JMIN = IA(VK)
        JMAX = IA(VK+1) - 1
        IF (JMIN.GT.JMAX)  GO TO 101
        DO 7 J=JMIN,JMAX
          VJ = JA(J)
          IF (VJ.EQ.VK)  GO TO 7
C ***** Search for VJ in adjacency of VK *****************************
          LLK = VK
   3      LK = LLK
          LLK = LV(LK)
          IF (VV(LLK) - VJ)  3, 7, 4
C ***** Insert VJ in adjacency of VK *********************************
   4      VV(VK) = VV(VK) + 1
          IF (SFS.EQ.0)  GO TO 109
          LLK = SFS
          SFS = LV(SFS)
          VV(LLK) = VJ
          LV(LLK) = LV(LK)
          LV(LK) = LLK
C ***** Search for VK in adjacency of VJ *****************************
          LLJ = VJ
   5      LJ = LLJ
          LLJ = LV(LJ)
          IF (VV(LLJ) - VK)  5, 7, 6
C ***** Insert VK in adjacency of VJ *********************************
   6      VV(VJ) = VV(VJ) + 1
          IF (SFS.EQ.0)  GO TO 109
          LLJ = SFS
          SFS = LV(SFS)
          VV(LLJ) = VK
          LV(LLJ) = LV(LJ)
          LV(LJ) = LLJ
   7      CONTINUE
   8    CONTINUE
C
C * Minimum degree algorithm with threshhold search *******************
C * Initialize vertex count and threshholds for search ****************
      I = 0
      JMIN = 1
      DTHR = 0
      DMIN = N+N
C * While uneliminated vertices exist *********************************
C *** Search for vertex VI of minimum degree **************************
   9    JMIN = MAX0 (JMIN, I+1)
        DO 10 J=JMIN,N
          VI = P(J)
          IF (VV(VI).LE.DTHR)  GO TO 11
  10      DMIN = MIN0 (DMIN, VV(VI))
        JMIN = 1
        DTHR = DMIN
        DMIN = N+N
        GO TO 9
```

```
C *** Number vertex VI of minimum degree ******************************
  11      JMIN = J
          I = I+1
          VJ = P(I)
          P(J) = VJ
          IP(VJ) = J
          P(I) = VI
          IP(VI) = I
          NI = 1
C *** Delete eliminated vertices from adjacency of VI *****************
C *** For every vertex VK adjacent to VI ******************************
          LLI = VI
          KMAX = (VV(VI) - NI) - N
          IF (KMAX.LE.0)  GO TO 14
          DO 13 K=1,KMAX
  12        LI = LLI
            LLI = LV(LI)
            VK = VV(LLI)
C ***** If VK eliminated, then delete from adjacency of VI ************
            IF (IP(VK).GT.I)  GO TO 13
              LV(LI) = LV(LLI)
              LV(LLI) = SFS
              SFS = LLI
              LLI = LI
              GO TO 12
  13        CONTINUE
C *** Eliminate vertex VI *********************************************
C *** For every vertex VK adjacent to VI ******************************
  14      LLI = VI
          KMAX = (VV(VI) - NI) - N
          IF (KMAX.LE.0)  GO TO 21
          DO 20 K=1,KMAX
            LI = LLI
            LLI = LV(LI)
            VK = VV(LLI)
C ***** Merge adjacency of VI into adjacency of VK *******************
C ***** For every vertex VJ adjacent to VI ****************************
            LLK = VK
            LJ = VI
            JMAX = (VV(VI) - NI) - N
            IF (JMAX.LE.0)  GO TO 19
            DO 18 J=1,JMAX
              LJ = LV(LJ)
              VJ = VV(LJ)
              IF (VJ.EQ.VK)  GO TO 18
C ******* Search for VJ in adjacency of VK ... ************************
  15          LK = LLK
              LLK = LV(LK)
              VL = VV(LLK)
              IF (VJ.LE.VL)  GO TO 17
C ******* ... while deleting eliminated vertices *********************
              IF (IP(VL).GT.I)  GO TO 16
                LV(LK) = LV(LLK)
                LV(LLK) = SFS
                SFS = LLK
                LLK = LK
  16          GO TO 15
```

```
C ******* Insert VJ in adjacency of VK ********************************
 17          IF (VJ.EQ.VL)  GO TO 18
               VV(VK) = VV(VK) + 1
               IF (SFS.EQ.0)  GO TO 109
               LLK = SFS
               SFS = LV(SFS)
               VV(LLK) = VJ
               LV(LLK) = LV(LK)
               LV(LK) = LLK
 18          CONTINUE
C ***** If VK of minimal degree, then number vertex VK, ... ***********
 19        IF (VV(VK).GT.VV(VI))  GO TO 20
             I = I+1
             J = IP(VK)
             VJ = P(I)
             P(J) = VJ
             IP(VJ) = J
             P(I) = VK
             IP(VK) = I
             NI = NI + 1
C ***** ... recover storage for adjacency of VK, **********************
             TMP = LV(VK)
             LV(VK) = SFS
             SFS = TMP
C ***** ... and delete VK from adjacency of VI ************************
             LV(LI) = LV(LLI)
             LV(LLI) = SFS
             SFS = LLI
             LLI = LI
 20        CONTINUE
C *** Update degrees of uneliminated vertices adjacent to VI **********
C *** For every vertex VK in adjacency of VI **************************
 21      LI = VI
         KMAX = (VV(VI) - NI) - N
         IF (KMAX.LE.0)  GO TO 24
         DO 23 K=1,KMAX
           LI = LV(LI)
           VK = VV(LI)
C ***** Update degree of VK and threshholds for cyclic search *********
           VV(VK) = VV(VK) - NI
           IF (VV(VK).GE.DMIN)  GO TO 23
             IF (VV(VK).GT.DTHR)  GO TO 22
               DMIN = DTHR
               DTHR = VV(VK)
               JMIN = MIN0 (JMIN, IP(VK))
               GO TO 23
 22          DMIN = VV(VK)
 23        CONTINUE
C *** Recover storage for adjacency of VI  ****************************
 24      TMP = LV(VI)
         LV(VI) = SFS
         SFS = TMP
         IF (I.LT.N)  GO TO 9
C
       FLAG = 0
       RETURN
```

```
C
C ** ERROR:  Null row in A
 101     FLAG = N + VK
         RETURN

C ** ERROR:  Insufficient Storage
 109     FLAG = 9*N + VK
         RETURN
         END
C
C        ----------------------------------------------------------------
C
C*** Subroutine SRO
C*** Symmetric reordering of sparse symmetric matrix
C
         SUBROUTINE  SRO
     *      (N, IP, IA,JA,A, TMP, Q)
C
C        Input variables:   N, IP, IA,JA,A
C        Output variables:  IA,JA,A
C
C        Parameters used internally:
C nia    | TMP - Initially,  TMP(K)  is set to the number of elements
C        |           which will appear in the Kth row of  M  after
C        |           reordering.  Then  TMP  is initialized to  IA  and
C        |           USED to set  Q.
C        |           Size = N.
C nia    | Q   - Initially,  Q(J)  is set to the row in which  A(J)
C        |           (the element of the old A) will appear after
C        |           reordering.  Then it is set to the index of  A(J)  in
C        |           the reordered matrix.
C        |           Size = number of nonzeros in the upper triangle of M.
C
C          The subroutine does not rearrange the order of the rows, but
C    arranges each row so that the elements which will be above the
C    diagonal after reordering are filled in.  If  M(I,J)  is above the
C    diagonal but is below the diagonal after reordering, then  M(J,I)
C    must be filled in, so some elements will appear on different rows
C    after SRO is finished.
C
         INTEGER  IP(1),  IA(1), JA(1),  TMP(1), Q(1),  QK
         REAL  A(1)
C
C  ******  Initialize TMP  ********************************************
         DO 1 I=1,N
  1        TMP(I) = 0
C  ******  For each row of A  *****************************************
         DO 3 I=1,N
           JMIN = IA(I)
           JMAX = IA(I+1) - 1
           IF (JMIN.GT.JMAX)  GO TO 3
C  ******  For each element of the row  *******************************
           DO 2 J=JMIN,JMAX
             K = JA(J)
C  ******  Adjust TMP, Q, and adjust JA if necessary  *****************
             IF (IP(K).LT.IP(I))  JA(J) = I
             IF (IP(K).GE.IP(I))  K = I
             Q(J) = K
  2          TMP(K) = TMP(K) + 1
  3        CONTINUE
```

```
C
C  ******  Set new IA and copy it into TMP  ***************************
          DO 4 I=1,N
            IA(I+1) = IA(I) + TMP(I)
   4        TMP(I) = IA(I)
C  ******  Q(J) gets position of A(J) after reordering  ***************
          JMIN = IA(1)
          JMAX = IA(N+1) - 1
          DO 5 J=JMIN,JMAX
            K = Q(J)
            Q(J) = TMP(K)
   5        TMP(K) = TMP(K) + 1
C
C  ******  Reset JA and A  ********************************************
          DO 7 J=JMIN,JMAX
   6        IF (Q(J).EQ.J)  GO TO 7
              K = Q(J)
              Q(J) = Q(K)
              Q(K) = K
              JAK = JA(K)
              JA(K) = JA(J)
              JA(J) = JAK
              AK = A(K)
              A(K) = A(J)
              A(J) = AK
              GO TO 6
   7        CONTINUE
          RETURN
          END
```

## Appendix 2

## Subroutines for Solving Sparse Symmetric Positive Definite Systems of Linear Equations

```
C*** Subroutine SDRV
C*** Driver for subroutines for solving sparse symmetric positive
C       definite systems of linear equations
C
        SUBROUTINE SDRV
     *     (N, P,IP, IA,JA,A, B, Z, NSP,ISP,RSP, PATH, FLAG)
C
C    PARAMETERS
C    Class abbreviations are:
C       v - supplies a VALUE to the driver
C       r - contains a RESULT returned by the driver
C       i - is used INTERNALly by the driver
C       a - is an ARRAY
C       n - is an INTEGER variable
C       f - is a REAL variable.
C
C Class | Parameter
C ------+----------
C       |
C         The nonzero entries of the matrix  M  are stored row by row
C    in the array  A.  The array  JA  contains the corresponding column
C    indices; i.e. if  A(K) = M(I,J),  then  JA(K) = J.  The array  IA
C    contains pointers to delimit the rows of M --  IA(I)  is the index
C    in  JA and A  of the first entry stored in the Ith row of  M.
C    Only the nonzero entries on or above the diagonal need be stored.
C    However, the subroutines will work if all nonzeros are stored.
C    For example, the symmetric 5 by 5 matrix
C       1  0  2  3  0
C       0  4  0  0  0
C       2  0  5  6  0
C       3  0  6  7  8
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  4  5  7  9 10
C       JA | 1  3  4  2  3  4  4  5  5
C        A | 1  2  3  4  5  6  7  8  9
C
C    IA(I+1) - IA(I)  is the number of nonzero entries in row I, so
C    IA(N+1), where N is the number of rows in  M, is needed to
C    determine the length of the Nth row in  A.
C
C vn    | N     - the number of rows/columns in matrix  M.
C vfa   | A     - coefficient matrix for the system of linear equations
C       |           Mx = b, stored in compressed form.
C       |           Size = number of nonzeros in upper triangle of  M
C       |                  (or the number of nonzeros in all of  M).
C vna   | IA    - pointers to the first element of each row in  A.
C       |           Size = N+1.
C vna   | JA    - the column numbers corresponding to elements of  A.
C       |           Size = size of  A.
C vna   | B     - right-hand side for the equation  Mx = b.  B and  Z
C       |           cannot be the same vector.
C       |           Size = N.
```

```
C rna   | Z      - solution vector for the equation  Mx = b.  B and Z
C       |            cannot be the same vector.
C       |            Size = N.
C
C          The solution of the system is done in three stages:
C      SYMFAC - The matrix  M  is processed symbolically to determine
C               where fillin will occur during factorization.
C      NUMFAC - The matrix  M  is factored numerically into two
C               triangular matrices.
C      NUMSLV - The system resulting from NUMFAC is solved.
C          For several systems with identical nonzero structures, SYMFAC
C      need be done only once, then NUMFAC and NUMSLV are done for each
C      system.  For several system with identical matrices  M  and
C      different right-hand sides, SYMFAC and NUMFAC need be done only
C      once, then NUMSLV is done for each right-hand side.
C
C vn    | PATH  - information on which subroutines are to be called.
C       |            Values and meanings of  PATH  are:
C       |                1  perform SYMFAC, NUMFAC and NUMSLV.
C       |                2  perform NUMFAC and NUMSLV.  (SYMFAC is
C       |                     assumed to have been done in a manner
C       |                     compatible with the driver's storage
C       |                     allocation.)
C       |                3  perform NUMSLV only.  (SYMFAC and NUMFAC
C       |                     are assumed to have been done.)
C rn    | FLAG  - flag for error return from subroutines.  Error values
C       |            and their meanings are:
C       |                0        no error
C       |               N+I       row I of  A  is null
C       |              2N+I       duplicate entry on row I of  A
C       |              6N+I       storage exceeded on row I in SYMFAC
C       |              7N+1       storage exceeded in NUMFAC
C       |              8N+I       diagonal element=0 on row I in NUMFAC
C       |             10N+1       ISP/RSP  too small to allocate space
C       |             11N+1       PATH  out of bounds
C
C          The rows and columns of the original matrix  M  can be
C      arbitrarily reordered before calling the driver.  If no reordering
C      is done,  then  P(I) = IP(I) = I  for I=1,N.  The answer vector  Z
C      is returned in the original order.
C
C vna   | P      - the ordering of the rows (and columns) of  M.  P(I)
C       |            is the number of the row of  M  which becomes the
C       |            Ith row after reordering.
C       |            Size = N.
C vna   | IP     - the inverse of the ordering of the rows of  M.  That
C       |            is,  IP(P(I)) = I  for I=1,N.
C       |            Size = N.
C
C          Workspace is needed to hold the factored form of the matrix
C      M  plus various temporary vectors.
C
C na    | ISP    - integer storage space divided up for various arrays
C       |            of the subroutines.  ISP and RSP  should be the
C       |            same array.  This allows declaration of all real
C       |            storage to be double precision.
C n     | NSP    - dimensioned size of  ISP and RSP.  NSP  generally
C       |            must be at least 4N+1 + 2*K (where  K = (number of
C       |            nonzeros in the upper triangle of  M)), since  ISP
```

```
C        |            and RSP  must hold:
C        |                four vectors of fixed length;
C        |                JU  (with size = K + fillin - compression);
C        |                U  (with size = K + fillin).
C fa     | RSP   - real storage space divided up for various arrays of
C        |            the subroutines.  ISP and RSP  should be the same
C        |            array.  This allows declaration of all real storage
C        |            to be double precision.
C
        INTEGER  P(1), IP(1),  IA(1), JA(1),  ISP(1),  PATH,  FLAG,
     *     Q, D, U, ROW, TMP,  UMAX
        REAL  A(1),  B(1),  Z(1),  RSP(1)
C       EQUIVALENCE  (ISP(1), RSP(1))
C
        IF (PATH.LT.1 .OR. PATH.GT.3)  GO TO 111
C****** Initialize and divide up temporary storage  *****************
        IJU = 1
        IU  = IJU +  N
        JL  = IU  + N+1
        JU  = JL  +  N
        Q   = NSP -  N
        JUMAX = Q - JU
        IF (JUMAX.LT.0)  GO TO 110
C
C****** Call subroutines  ******************************************
        FLAG  = 0
        IF (PATH.GT.1)  GO TO 1
          CALL SSF
     *       (N,  P, IP,  IA, JA,  ISP(IJU), ISP(JU), ISP(IU), JUMAX,
     *        RSP(Q), ISP(JL), FLAG)
          IF(FLAG.NE.0)  GO TO 100
C
   1    D   = Q  - N
        U   = JU + ISP(IJU+(N-1))
        ROW = Q
        UMAX = D - U
        IF (PATH.GT.2)  GO TO 2
           CALL SNF
     *        (N,  P, IP,  IA, JA, A,
     *         RSP(D),  ISP(IJU), ISP(JU), ISP(IU), RSP(U), UMAX,
     *         RSP(ROW),  ISP(JL),  FLAG)
          IF (FLAG.NE.0)  GO TO 100
C
   2    TMP = Q
        CALL SNS
     *     (N,  P,  RSP(D), ISP(IJU), ISP(JU), ISP(IU), RSP(U),  Z,  B,
     *      RSP(TMP))
        RETURN
C
C ** ERROR:  Error Detected in SSF, SNF, or SNS
 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               YALE SPARSE MATRIX PACKAGE - SYMMETRIC CODES
C                  SOLVING THE SYSTEM OF EQUATIONS Mx = b
C
C    I.    SUBROUTINE NAMES
C          Subroutine names are of the form Sxx where:
C   (1)    the first letter is  S  for symmetric matrices;
C   (2)    the second letter is either  S  for symbolic or  N  for
C          numerical processing;
C   (3)    the third letter is either  F  for factorization or  S  for
C          solution.
C
C    II.   CALLING SEQUENCES
C          The input matrix can be processed with an ordering subroutine
C    before using the remaining subroutines.  If this is done and only
C    the upper triangle of  M  is being stored, SRO should be called to
C    reorder the matrix into symmetric form before using the other
C    subroutines.  If an ordering subroutine is not used, set  P(I) =
C    IP(I) = I  for I=1,N.  Then the calling sequence is
C          SSF   (symbolic factorization)
C          SNF   (numerical factorization)
C          SNS   (called once for each right-hand side).
C
C    III. STORAGE OF SPARSE MATRICES
C          The nonzero entries of the matrix  M  are stored row by row
C    in the array  A.  The array  JA  contains the corresponding column
C    indices; i.e. if  A(K) = M(I,J),  then  JA(K) = J.  The array  IA
C    contains pointers to delimit the rows of M --  IA(I)  is the index
C    in  JA and A  of the first entry stored in the Ith row of  M.
C    Only the nonzero entries on or above the diagonal need be stored.
C    However, the subroutines will work if all nonzeros are stored.
C    For example, the symmetric 5 by 5 matrix
C       1  0  2  3  0
C       0  4  0  0  0
C       2  0  5  6  0
C       3  0  6  7  8
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  4  5  7  9 10
C       JA | 1  3  4  2  3  4  4  5  5
C        A | 1  2  3  4  5  6  7  8  9
C
C    IA(I+1) - IA(I)  is the number of nonzero entries in row I, so
C    IA(N+1), where N is the number of rows in  M, is needed to
C    determine the length of the Nth row in  A.
C          The unit triangular matrix  U  is stored in a similar fashion
C    using the arrays  IU, JU, and U  except that an additional vector
C    IJU  is used to compress storage of  JU.  IJU(K)  points to the
C    starting location in  JU  of entries for the Kth row.  Compression
C    occurs in two ways.  First, if a row I was merged into the current
C    row K, and the number of elements merged in from row I (some tail
C    portion of row I) is the same as the final length of row K, then
C    the Kth row and the tail are identical and  IJU(K)  can point to
C    the start of the tail.  Second, if some tail portion of the K-1st
C    row equals the head of the Kth row, then  IJU(K)  can point to the
C    start of that tail section.  For example, the nonzero structure of
```

```
C    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    might be stored, ignoring the diagonal, 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
C         The diagonal entries of  U  are assumed to be equal to one
C    and are not stored.  The array  D  contains reciprocals of entries
C    of the diagonal matrix in the U DU decomposition.
C
C    IV.  ADDITIONAL STORAGE SAVINGS
C         In SSF and SNF,  P and IP  can be the same vector in the
C    calling sequences if no reordering of the matrix has been done
C    (i.e.  P(I) = IP(I) = I  for I=1,N).
C         In SNS,  B and Z  can be the same; however, the right-hand
C    side  B  will be destroyed.
C
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       v - supplies a VALUE to the subroutine
C       r - contains a RESULT returned by the subroutine
C       i - is used INTERNALly by the subroutine
C       a - is an ARRAY
C       n - is an INTEGER variable
C       f - is a REAL variable.
C
C Class | Parameter
C ------+----------
C fva   | A      - coefficient matrix for the system of linear equations
C       |            Mx = b, stored in compressed form.
C       |            Size = either the number of nonzeros in the upper
C       |                   triangle of  M, or the number of nonzeros in
C       |                   all of  M  (see section III).
C fva   | B      - right-hand side for the equation  Mx = b.
C       |            Size = N.
C fvra  | D      - inverse of diagonal matrix in UtDU factorization
C       |            (also used for temporary results in SNF).
C       |            Size = N.
C nvra  | IA     - pointers to first elements of each row in  A.
C       |            Size = N+1.
C nr    | FLAG   - flag for error return from subroutines.  Error values
C       |            and their meanings are:
C       |                0        no error
C       |               N+I       row I of  A  is null
C       |              2N+I       duplicate entry on row I of  A
C       |              6N+I       JU  storage exceeded on row I
C       |              7N+1       U  storage exceeded
C       |              8N+I       zero diagonal element on row I
C nvra  | IJU    - pointers to the first elements of each row in  JU,
C       |            used to compress storage of  JU.
C       |            Size = N.
C nva   | IP     - inverse of the ordering of the rows of  M.  For
C       |            example, if row 1 is the 5th row after reordering,
C       |            then  IP(1)=5.
C       |            Size = N.
```

```
C nvra  | IU     - pointers to the first elements of each row in  U.
C       |            Size = N+1.
C nvra  | JA     - column numbers corresponding to elements of  A.
C       |            Size = size of A.
C nvra  | JU     - column numbers corresponding to elements of  U.
C       |            Size = size of U - compression.
C nv    | JUMAX - declared dimension of  JU.
C nv    | N      - number of rows/columns in matrix  M.
C nva   | P      - ordering of rows (and columns) of  M.  P(I)  is
C       |            the number of the row of  M  which becomes the Ith
C       |            row after reordering.
C       |            Size = N.
C fvra  | U      - upper triangular matrix resulting from the
C       |            factorization of  M, stored in compressed form.
C       |            Size = number of nonzeros in upper triangle of  M
C       |                   plus fillin  (IU(N+1)-1 after SSF).
C nv    | UMAX  - declared dimension of  U.
C fra   | Z      - solution vector for the equation  Mx = b.
C       |            Size = N.
C
C       ------------------------------------------------------------
C
C*** Subroutine SSF
C*** Symbolic Ut-D-U factorization of sparse symmetric matrix
C
        SUBROUTINE  SSF
     *     (N, P,IP, IA,JA, IJU,JU,IU,JUMAX, Q, JL, FLAG)
C
C       Input variables:   N, P,IP, IA,JA, JUMAX
C       Output variables:  IJU,JU,IU, FLAG
C
C       Parameters used internally:
C nia   | JL - linked list of rows to be merged.  If the Kth row is
C       |         being processed,  JL(K)  contains the number of the
C       |         first row to be merged with the Kth row,  JL(JL(K))
C       |         is the number of the second row, etc.
C       |         Size = N.
C nia   | Q -  Suppose  M'  is the result of reordering  M.  If
C       |         processing of the Kth row of  M'  (hence the Kth row
C       |         of  U) is being done,  Q(J)  is initially nonzero if
C       |         M'(K,J)  is nonzero and above the diagonal.  Since
C       |         values need not be stored, each entry points to the
C       |         next nonzero and  Q(K)  points to the first.  N+1
C       |         indicates the last element.  For example, if N=9 and
C       |         the 5th row of  M'  is
C       |            0 x x 0 x 0 0 x 0
C       |         then  Q  will initially be
C       |            a a a a 8 a a 10 a           (a - arbitrary).
C       |         As the algorithm proceeds, other elements of  Q  are
C       |         inserted in the list because of fillin.
C       |         Size = N.
C
C  Internal variables:
C    JUMIN,JUPTR - are the indices in  JU  of the first and last
C                  elements in either the last or the current row.
C    LMAX - length of longest row merged into  Q.
C    LUI - number of elements in a row to be merged into  Q.
C    LUK - number of elements in the current row  (Q).
```

```
C
      INTEGER  P(1), IP(1),  IA(1), JA(1),  IJU(1), JU(1), IU(1),
     *   Q(1),  JL(1),  FLAG,  VJ, QM
C
C  ******  Initialize  **********************************************
      JUMIN = 1
      JUPTR = 0
      IU(1) = 1
      DO 1 K=1,N
  1     JL(K) = 0
C
C  ******  For each row  ********************************************
      DO 15 K=1,N
C  ******  Initialize Q to structure of Kth row above diagonal  *******
        LUK = 0
        Q(K) = N+1
        JMIN = IA(P(K))
        JMAX = IA(P(K)+1) - 1
        IF (JMIN.GT.JMAX)  GO TO 101
        DO 3 J=JMIN,JMAX
          VJ = IP(JA(J))
          IF (VJ.LE.K)  GO TO 3
            QM = K
  2         M = QM
            QM = Q(M)
            IF (QM.LT.VJ)  GO TO 2
            IF (QM.EQ.VJ)  GO TO 102
              LUK = LUK+1
              Q(M) = VJ
              Q(VJ) = QM
  3       CONTINUE
C
C  ******  Compute fillin for Q by  **********************************
        LMAX = 0
        IJU(K) = JUPTR
        I = K
C  ******  Linking through JL and  ***********************************
  4     I = JL(I)
        IF (I.EQ.0)  GO TO 8
          LUI = IU(I+1) - (IU(I)+1)
          JMIN = IJU(I) +  1
          JMAX = IJU(I) + LUI
          IF (LUI.LE.LMAX)  GO TO 5
            LMAX = LUI
            IJU(K) = JMIN
  5       QM = K
C  ******  Merging each row with Q  **********************************
          DO 7 J=JMIN,JMAX
            VJ = JU(J)
  6         M = QM
            QM = Q(M)
            IF (QM.LT.VJ)  GO TO 6
            IF (QM.EQ.VJ)  GO TO 7
              LUK = LUK+1
              Q(M) = VJ
              Q(VJ) = QM
              QM = VJ
  7         CONTINUE
          GO TO 4
```

```
C
C  ******  Check if row duplicates another.  If not  ******************
   8       IF (LUK.EQ.LMAX)  GO TO 14
C  ******  see if tail of K-1st row matches head of Kth  **************
             IF (JUMIN.GT.JUPTR)  GO TO 12
             I = Q(K)
             DO 9 JMIN=JUMIN,JUPTR
               IF (JU(JMIN)-I)  9, 10, 12
   9           CONTINUE
             GO TO 12
  10         IJU(K) = JMIN
             DO 11 J=JMIN,JUPTR
               IF (JU(J).NE.I)  GO TO 12
               I = Q(I)
               IF (I.GT.N)  GO TO 14
  11           CONTINUE
             JUPTR = JMIN - 1
C
C  ******  Set Kth row of U to Q  *************************************
  12         JUMIN = JUPTR +  1
             JUPTR = JUPTR + LUK
             IF (JUPTR.GT.JUMAX)  GO TO 106
             I = K
             DO 13 J=JUMIN,JUPTR
               I = Q(I)
  13           JU(J) = I
             IJU(K) = JUMIN
C
C  ******  If more than one element in row, adjust JL  ****************
  14       IF (LUK.LE.1)  GO TO 15
             I = JU(IJU(K))
             JL(K) = JL(I)
             JL(I) = K
  15       IU(K+1) = IU(K) + LUK
C
        FLAG = 0
        RETURN
C
C ** ERROR:  Null Row in A
 101    FLAG = N + P(K)
        RETURN
C ** ERROR:  Duplicate Entry in A
 102    FLAG = 2*N + P(K)
        RETURN
C ** ERROR:  Insufficient Storage for JU
 106    FLAG = 6*N + K
        RETURN
        END
```

```
C
C       ------------------------------------------------------------
C
C*** Subroutine SNF
C*** Numerical Ut-D-U factorization of sparse symmetric positive
C       definite matrix
C
        SUBROUTINE  SNF
     *     (N, P,IP, IA,JA,A, D, IJU,JU,IU,U,UMAX, IL, JL, FLAG)
C
C
C       Input variables:   N, P,IP, IA,JA,A, IJU,JU,IU
C       Output variables:  D,U, FLAG
C
C       Parameters used internally:
C niva  | D   - If the Kth row of  U  is being computed,  D(1) through
C       |         D(K-1)  contain reciprocals of the entries of the
C       |         diagonal matrix  D  from the decomposition.  The
C       |         remainder of  D  is initialized to the structure of
C       |         the Kth row of  M  (after reordering) and is adjusted
C       |         to become the Kth row of  U.
C nia   | IL - IL(I)  points to the first element of the Ith row to be
C       |         used in adjusting the current row.
C       |         Size = N.
C nia   | JL - linked list of rows to be used in adjusting the current
C       |         row.  If the Kth row is being processed,  JL(K)
C       |         contains the number of the first row to be used with
C       |         the Kth row,  JL(JL(K)) is the number of the second
C       |         row, etc.
C       |         Size = N.
C
        INTEGER  P(1), IP(1),  IA(1), JA(1),  IJU(1), JU(1), IU(1),
     *     UMAX,  IL(1),  JL(1),  FLAG,  VK, VJ
        DIMENSION  A(1),  D(1), U(1)
C
C  ******  Initialize JL, check storage  ****************************
        IF (IU(N+1)-1 .GT. UMAX)  GO TO 107
        DO 1 K=1,N
   1      JL(K) = 0
C
C  ******  For each row  *********************************************
        DO 10 K=1,N
C  ******  Initialize D on and above the diagonal  *******************
          JMIN = IU(K)
          JMAX = IU(K+1) - 1
          IF (JMIN.GT.JMAX)  GO TO 3
          MU = IJU(K) - IU(K)
          DO 2 J=JMIN,JMAX
   2        D(JU(MU+J)) = 0
   3      D(K) = 0
          VK = P(K)
          JMIN = IA(VK)
          JMAX = IA(VK+1) - 1
          DO 4 J=JMIN,JMAX
            VJ = IP(JA(J))
            IF (K.LE.VJ)  D(VJ) = A(J)
   4        CONTINUE
```

```
C
C  ******  For each element in lower triangle to be eliminated  *******
          DK = D(K)
          NXTI = JL(K)
    5     I = NXTI
          IF (I.EQ.0)  GO TO 8
C  ******  Change D and adjust IL and JL  ****************************
            NXTI = JL(I)
            UKIDI = - U(IL(I)) * D(I)
            DK = DK + UKIDI * U(IL(I))
            U(IL(I)) = UKIDI
            JMIN = IL( I ) + 1
            JMAX = IU(I+1) - 1
            IF (JMIN.GT.JMAX)  GO TO 7
              MU = IJU(I) - IU(I)
              DO 6 J=JMIN,JMAX
    6           D(JU(MU+J)) = D(JU(MU+J)) + UKIDI * U(J)
              IL(I) = JMIN
              J = JU(MU+JMIN)
              JL(I) = JL(J)
              JL(J) = I
    7       GO TO 5
C
C  ******  Set D(K) and copy rest of D into Kth row of U  *************
    8     IF (DK.EQ.0)  GO TO 108
          D(K) = 1 / DK
          JMIN = IU(K)
          JMAX = IU(K+1) - 1
          IF (JMIN.GT.JMAX)  GO TO 10
            MU = IJU(K) - JMIN
            DO 9 J=JMIN,JMAX
    9         U(J) = D(JU(MU+J))
            IL(K) = JMIN
            I = JU(MU+JMIN)
            JL(K) = JL(I)
            JL(I) = K
   10     CONTINUE
C
        FLAG = 0
        RETURN
C
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 SNS
C*** Numerical solution of sparse symmetric positive definite system of
C       linear equations given Ut-D-U factorization
C
       SUBROUTINE  SNS
     *    (N, P, D, IJU,JU,IU,U, Z, B, TMP)
C
C      Input variables:   N, P, D, IJU,JU,U, B
C      Output variables:  Z
C
C      Parameters used internally:
C fia  | TMP - vector which gets result of solving  Ut Dy = b.
C      |         Size = N.
C
       INTEGER  P(1),  IJU(1), JU(1), IU(1)
       REAL  D(1), U(1),  Z(1), B(1),  TMP(1)
C
C  ******  Initialize TMP to the reordered B  ************************
       DO 1 K=1,N
  1      TMP(K) = B(P(K))
C  ******  Solve  Ut Dy = b  by forward substitution  ****************
       DO 3 K=1,N
         TMPK = TMP(K)
         JMIN = IU(K)
         JMAX = IU(K+1) - 1
         IF (JMIN.GT.JMAX)  GO TO 3
         MU = IJU(K) - JMIN
         DO 2 J=JMIN,JMAX
  2        TMP(JU(MU+J)) = TMP(JU(MU+J)) + U(J) * TMPK
  3      TMP(K) = TMPK * D(K)
C
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(P(K)) = SUM
  6      K = K-1
       RETURN
       END
```

## Appendix 3

## Test Driver for Sparse Symmetric Matrix Package

```
C*** Program STST
C*** Test Driver for Symmetric 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 (upper triangle of) M;
C             DIMENSION = number of nonzero entries in (upper triangle
C             of) M.
C
C     A    -  REAL one-dimensional array used to store nonzero elements
C             of (upper triangle of) M;  DIMENSION = number of nonzero
C             entries in (upper triangle of) 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
      INTEGER  IA(101), JA(500),  P(100), IP(100),  ISP(1500),
     *   CASE, PATH, FLAG,  APTR,VP,VQ,  X,XMIN,XMAX,  Y,YMIN,YMAX
      REAL  A(500), Z(100), B(100),  RSP(1500)
      EQUIVALENCE  (ISP(1), RSP(1))
      DATA  NSP/1500/, EPS/1E-5/
C
      INDEX(I,J) = NG*I + J - NG
C
      NG = 3
      N = NG*NG
```

```
C******  For CASE=1 we store the entire matrix, for CASE=2 we store
C******  only the upper triangular part
      DO 5 CASE=1,2
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 = MAX0 ( 1, I-1)
          XMAX = MIN0 (NG, I+1)
          YMIN = MAX0 ( 1, J-1)
          YMAX = MIN0 (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) = 4
                IF (VP .NE. VQ)  A(APTR) = -1
                SUM = SUM + A(APTR) * VQ
C******  If CASE=2, do not store elements below diagonal  **************
                IF(VP.GT.VQ .AND. CASE.EQ.2)  GO TO 1
                  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 '
     *        /    '     (ALL ENTRIES OF MATRIX STORED) ')
      IF (CASE.EQ.2)  PRINT 1002, NG,NG
1002  FORMAT (/' *** FIVE-POINT OPERATOR ON ', I1, ' BY ' I1, ' GRID '
     *        /    '     (ONLY ENTRIES OF UPPER TRIANGLE STORED) ')
      PRINT 1003, (IA(I),I=1,N), IA(N+1)
1003  FORMAT (/' COEFFICIENT MATRIX: '/
     *        /'    IA (INDICES OF FIRST ELEMENTS IN ROWS)'
     *        /(10I5))
      PRINT 1004, (I,JA(I),A(I), I=1,NZA)
1004  FORMAT (/'             JA             A  '
     *        /'  I   COLUMN INDICES    MATRIX'
     *        /(I3, I10, F16.5))
      PRINT 1005, (B(I), I=1,N)
1005  FORMAT (/' RIGHT HAND SIDE B: '
     *        /(5F10.5))
C
C *****  Call ODRV ***************************************************
      FLAG = 0
      PATH = CASE
      CALL  ODRV
     *   (N, IA,JA,A, P,IP, NSP,RSP, PATH, FLAG)
      IF (FLAG.NE.0)  GO TO 101
C
C****** Output reordered array A *************************************
      PRINT 1006, (I,P(I),IP(I), I=1,N)
```

```
 1006    FORMAT (/' ROW/COLUMN ORDERING FROM ODRV: '/
     *          /'                P                    IP        '
     *          /'  I  ROW/COL ORDERING    INVERSE ORDERING '
     *          /(I3, I10, I20))
         IF (CASE.EQ.2)  PRINT 1007, (IA(I), I=1,N), IA(N+1)
 1007    FORMAT (/' REORDERED COEFFICIENT MATRIX: '/
     *          /' IA (INDICES OF FIRST ELEMENTS IN ROWS) '
     *          /(10I5))
         IF (CASE.EQ.2)  PRINT 1008, (I,JA(I),A(I), I=1,NZA)
 1008    FORMAT (/'              JA              A    '
     *          /'  I   COLUMN INDICES    MATRIX '
     *        (I3, I10, F16.5))
C
C *****  Call SDRV ******************************************************
         PATH = 1
         CALL  SDRV
     *      (N, P,IP, IA,JA,A, B, Z, NSP,ISP,RSP, PATH, FLAG)
         IF (FLAG.NE.0)  GO TO 102
C
C *****  Calculate error  **********************************************
         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 1009, (Z(I),I=1,N)
 1009    FORMAT (/' SOLUTION FROM SDRV: '
     *          /(5F10.5))
         IF (RMS.LE.EPS)  PRINT 1010, RMS
 1010    FORMAT (/' SOLUTION CORRECT:  RMS ERROR = ', 1PE8.2)
         IF (RMS.GT.EPS)  PRINT 1011, RMS
 1011    FORMAT (/' SOLUTION INCORRECT:  RMS ERROR = ', 1PE8.2)
C
   5     CONTINUE
         STOP
C
C******  Error messages  **********************************************
 101     PRINT 1012, FLAG
 1012    FORMAT (/' ERROR IN ODRV:  FLAG = ', I5)
         STOP
C
 102     PRINT 1013, FLAG
 1013    FORMAT (/' ERROR IN SDRV:  FLAG = ', I5)
         STOP
         END
```

## Appendix 4

### Sample Output From Test Driver

```
*** FIVE-POINT OPERATOR ON 3 BY 3 GRID
   (ALL ENTRIES OF MATRIX STORED)

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 | 4.00000 |
| 2 | 2 | -1.00000 |
| 3 | 4 | -1.00000 |
| 4 | 1 | -1.00000 |
| 5 | 2 | 4.00000 |
| 6 | 3 | -1.00000 |
| 7 | 5 | -1.00000 |
| 8 | 2 | -1.00000 |
| 9 | 3 | 4.00000 |
| 10 | 6 | -1.00000 |
| 11 | 1 | -1.00000 |
| 12 | 4 | 4.00000 |
| 13 | 5 | -1.00000 |
| 14 | 7 | -1.00000 |
| 15 | 2 | -1.00000 |
| 16 | 4 | -1.00000 |
| 17 | 5 | 4.00000 |
| 18 | 6 | -1.00000 |
| 19 | 8 | -1.00000 |
| 20 | 3 | -1.00000 |
| 21 | 5 | -1.00000 |
| 22 | 6 | 4.00000 |
| 23 | 9 | -1.00000 |
| 24 | 4 | -1.00000 |
| 25 | 7 | 4.00000 |
| 26 | 8 | -1.00000 |
| 27 | 5 | -1.00000 |
| 28 | 7 | -1.00000 |
| 29 | 8 | 4.00000 |
| 30 | 9 | -1.00000 |
| 31 | 6 | -1.00000 |
| 32 | 8 | -1.00000 |
| 33 | 9 | 4.00000 |

```
RIGHT HAND SIDE B:
 -2.00000  -1.00000   4.00000   3.00000   0.00000
  7.00000  16.00000  11.00000  22.00000
```

ROW/COLUMN ORDERING FROM ODRV:

| I | P ROW/COL ORDERING | IP 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 SDRV:

| | | | | |
|---|---|---|---|---|
| 1.00000 | 2.00000 | 3.00000 | 4.00000 | 5.00000 |
| 6.00000 | 7.00000 | 8.00000 | 9.00000 | |

SOLUTION CORRECT: RMS ERROR = 1.39E-08

*** FIVE-POINT OPERATOR ON 3 BY 3 GRID
(ONLY ENTRIES OF UPPER TRIANGLE STORED)

COEFFICIENT MATRIX:

IA (INDICES OF FIRST ELEMENTS IN ROWS)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 9 | 12 | 15 | 17 | 19 | 21 | 22 |

| I | JA COLUMN INDICES | A MATRIX |
|---|---|---|
| 1 | 1 | 4.00000 |
| 2 | 2 | -1.00000 |
| 3 | 4 | -1.00000 |
| 4 | 2 | 4.00000 |
| 5 | 3 | -1.00000 |
| 6 | 5 | -1.00000 |
| 7 | 3 | 4.00000 |
| 8 | 6 | -1.00000 |
| 9 | 4 | 4.00000 |
| 10 | 5 | -1.00000 |
| 11 | 7 | -1.00000 |
| 12 | 5 | 4.00000 |
| 13 | 6 | -1.00000 |
| 14 | 8 | -1.00000 |
| 15 | 6 | 4.00000 |
| 16 | 9 | -1.00000 |
| 17 | 7 | 4.00000 |
| 18 | 8 | -1.00000 |
| 19 | 8 | 4.00000 |
| 20 | 9 | -1.00000 |
| 21 | 9 | 4.00000 |

RIGHT HAND SIDE B:

| | | | | |
|---|---|---|---|---|
| -2.00000 | -1.00000 | 4.00000 | 3.00000 | 0.00000 |
| 7.00000 | 16.00000 | 11.00000 | 22.00000 | |

ROW/COLUMN ORDERING FROM ODRV:

| I | P<br>ROW/COL ORDERING | IP<br>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 |

REORDERED COEFFICIENT MATRIX:

IA (INDICES OF FIRST ELEMENTS IN ROWS)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 5 | 8 | 9 | 13 | 15 | 18 | 19 | 22 |

| I | JA<br>COLUMN INDICES | A<br>MATRIX |
|---|---|---|
| 1 | 1 | 4.00000 |
| 2 | 2 | -1.00000 |
| 3 | 4 | -1.00000 |
| 4 | 2 | 4.00000 |
| 5 | 2 | -1.00000 |
| 6 | 3 | 4.00000 |
| 7 | 6 | -1.00000 |
| 8 | 4 | 4.00000 |
| 9 | 2 | -1.00000 |
| 10 | 4 | -1.00000 |
| 11 | 5 | 4.00000 |
| 12 | 8 | -1.00000 |
| 13 | 5 | -1.00000 |
| 14 | 6 | 4.00000 |
| 15 | 4 | -1.00000 |
| 16 | 7 | 4.00000 |
| 17 | 8 | -1.00000 |
| 18 | 8 | 4.00000 |
| 19 | 6 | -1.00000 |
| 20 | 8 | -1.00000 |
| 21 | 9 | 4.00000 |

SOLUTION FROM SDRV:

| | | | | |
|---|---|---|---|---|
| 1.00000 | 2.00000 | 3.00000 | 4.00000 | 5.00000 |
| 6.00000 | 7.00000 | 8.00000 | 9.00000 | |

SOLUTION CORRECT:   RMS ERROR = 1.39E-08