内部资料不对外

数学软件资料汇编

（3）

稀疏矩阵程序包　　YSMP

Y S M P

Yale Sparse Matrix Package

Source

Content

```
C                                                                    1/16/81
C*********************************************************************
C  ODRV -- DRIVER FOR SPARSE MATRIX REORDERING ROUTINES
C*********************************************************************
      SUBROUTINE  ODRV
     *     (N, IA,JA,A, P,IP, NSP,ISP, PATH, FLAG)
C
C  DESCRIPTION
C
C    ODRV FINDS A MINIMUM DEGREE ORDERING OF THE ROWS AND COLUMNS OF A
C    SYMMETRIC MATRIX M STORED IN (IA,JA,A) FORMAT (SEE BELOW).  FOR THE
C    REORDERED MATRIX, THE WORK AND STORAGE REQUIRED TO PERFORM GAUSSIAN
C    ELIMINATION IS (USUALLY) SIGNIFICANTLY LESS.
C
C    IF ONLY THE NONZERO ENTRIES IN THE UPPER TRIANGLE OF M ARE BEING
C    STORED, THEN ODRV SYMMETRICALLY REORDERS (IA,JA,A), (OPTIONALLY)
C    WITH THE DIAGONAL ENTRIES PLACED FIRST IN EACH ROW.  THIS IS TO
C    ENSURE THAT IF M(I,J) WILL BE IN THE UPPER TRIANGLE OF M WITH
C    RESPECT TO THE NEW ORDERING, THEN M(I,J) IS STORED IN ROW I (AND
C    THUS M(J,I) IS NOT STORED);  WHEREAS IF M(I,J) WILL BE IN THE
C    STRICT LOWER TRIANGLE OF M, THEN M(J,I) IS STORED IN ROW J (AND
C    THUS M(I,J) IS NOT STORED).
C
C
C  STORAGE OF SPARSE MATRICES
C
C    THE NONZERO ENTRIES OF THE MATRIX M ARE STORED ROW-BY-ROW IN THE
C    ARRAY A.  TO IDENTIFY THE INDIVIDUAL NONZERO ENTRIES IN EACH ROW,
C    WE NEED TO KNOW IN WHICH COLUMN EACH ENTRY LIES.  THESE COLUMN
C    INDICES ARE STORED IN THE ARRAY JA;  I.E., IF  A(K) = M(I,J),  THEN
C    JA(K) = J.  TO IDENTIFY THE INDIVIDUAL ROWS, WE NEED TO KNOW WHERE
C    EACH ROW STARTS.  THESE ROW POINTERS ARE STORED IN THE ARRAY IA;
C    I.E., IF M(I,J) IS THE FIRST NONZERO ENTRY (STORED) IN THE I-TH ROW
C    AND  A(K) = M(I,J),  THEN  IA(I) = K.  MOREOVER, IA(N+1) POINTS TO
C    THE FIRST LOCATION FOLLOWING THE LAST ELEMENT IN THE LAST ROW.
C    THUS, THE NUMBER OF ENTRIES IN THE I-TH ROW IS  IA(I+1) - IA(I),
C    THE NONZERO ENTRIES IN THE I-TH ROW ARE STORED CONSECUTIVELY IN
C
C            A(IA(I)),  A(IA(I)+1),  ..., A(IA(I+1)-1),
C
C    AND THE CORRESPONDING COLUMN INDICES ARE STORED CONSECUTIVELY IN
C
C            JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).
C
C    SINCE THE COEFFICIENT MATRIX IS SYMMETRIC, ONLY THE NONZERO ENTRIES
C    IN THE UPPER TRIANGLE NEED BE STORED.  FOR EXAMPLE, THE MATRIX
C
C             ( 1  0  2  3  0 )
C             ( 0  4  0  0  0 )
C         M = ( 2  0  5  6  0 )
C             ( 3  0  6  7  8 )
C             ( 0  0  0  8  9 )
C
C    COULD BE STORED AS
C
C           | 1  2  3  4  5  6  7  8  9 10 11 12 13
C         ---+--------------------------------------
```

```
C         IA | 1  4  5  8 12 14
C         JA | 1  3  4  2  1  3  4  1  3  4  5  4  5
C          A | 1  2  3  4  2  5  6  3  6  7  8  8  9
C
C    OR (SYMMETRICALLY) AS
C
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
C  PARAMETERS
C
C    N    - ORDER OF THE MATRIX
C
C    IA   - INTEGER ONE-DIMENSIONAL ARRAY CONTAINING POINTERS TO DELIMIT
C           ROWS IN JA AND A;  DIMENSION = N+1
C
C    JA   - INTEGER ONE-DIMENSIONAL ARRAY CONTAINING THE COLUMN INDICES
C           CORRESPONDING TO THE ELEMENTS OF A;  DIMENSION = NUMBER OF
C           NONZERO ENTRIES IN (THE UPPER TRIANGLE OF) M
C
C    A    - REAL ONE-DIMENSIONAL ARRAY CONTAINING THE NONZERO ENTRIES IN
C           (THE UPPER TRIANGLE OF) M, STORED BY ROWS;  DIMENSION =
C           NUMBER OF NONZERO ENTRIES IN (THE UPPER TRIANGLE OF) M
C
C    P    - INTEGER ONE-DIMENSIONAL ARRAY USED TO RETURN THE PERMUTATION
C           OF THE ROWS AND COLUMNS OF M CORRESPONDING TO THE MINIMUM
C           DEGREE ORDERING;  DIMENSION = N
C
C    IP   - INTEGER ONE-DIMENSIONAL ARRAY USED TO RETURN THE INVERSE OF
C           THE PERMUTATION RETURNED IN P;  DIMENSION = N
C
C    NSP  - DECLARED DIMENSION OF THE ONE-DIMENSIONAL ARRAY ISP;  NSP
C           MUST BE AT LEAST  3N+4K,  WHERE K IS THE NUMBER OF NONZEROES
C           IN THE STRICT UPPER TRIANGLE OF M
C
C    ISP  - INTEGER ONE-DIMENSIONAL ARRAY USED FOR WORKING STORAGE;
C           DIMENSION = NSP
C
C    PATH - INTEGER PATH SPECIFICATION;  VALUES AND THEIR MEANINGS ARE -
C             1  FIND MINIMUM DEGREE ORDERING ONLY
C             2  FIND MINIMUM DEGREE ORDERING AND REORDER SYMMETRICALLY
C                  STORED MATRIX (USED WHEN ONLY THE NONZERO ENTRIES IN
C                  THE UPPER TRIANGLE OF M ARE BEING STORED)
C             3  REORDER SYMMETRICALLY STORED MATRIX AS SPECIFIED BY
C                  INPUT PERMUTATION (USED WHEN AN ORDERING HAS ALREADY
C                  BEEN DETERMINED AND ONLY THE NONZERO ENTRIES IN THE
C                  UPPER TRIANGLE OF M ARE BEING STORED)
C             4  SAME AS 2 BUT PUT DIAGONAL ENTRIES AT START OF EACH ROW
C             5  SAME AS 3 BUT PUT DIAGONAL ENTRIES AT START OF EACH ROW
C
C    FLAG - INTEGER ERROR FLAG;  VALUES AND THEIR MEANINGS ARE -
C               0    NO ERRORS DETECTED
C              9N+K  INSUFFICIENT STORAGE IN MD
```

```
C             10N+1  INSUFFICIENT STORAGE IN ODRV
C             11N+1  ILLEGAL PATH SPECIFICATION
C
C
C  CONVERSION FROM REAL TO DOUBLE PRECISION
C
C    CHANGE THE REAL DECLARATIONS IN ODRV AND SRO TO DOUBLE PRECISION
C    DECLARATIONS.
C
C-----------------------------------------------------------------------
C
        INTEGER  IA(1), JA(1),  P(1), IP(1),  ISP(1),  PATH,  FLAG,
     *     V, L, HEAD,  TMP, Q
        REAL  A(1)
C...    DOUBLE PRECISION  A(1)
        LOGICAL  DFLAG
C
C----INITIALIZE ERROR FLAG AND VALIDATE PATH SPECIFICATION
        FLAG = 0
        IF (PATH.LT.1 .OR. 6.LT.PATH)  GO TO 111
C
C----ALLOCATE STORAGE AND FIND MINIMUM DEGREE ORDERING
        IF ((PATH-1) * (PATH-2) * (PATH-4) .NE. 0)  GO TO 1
          MAX = (NSP-N)/2
          V    = 1
          L    = V     +  MAX
          HEAD = L     +  MAX
          NEXT = HEAD  +  N
          IF (MAX.LT.N)  GO TO 110
C
          CALL  MD
     *       (N, IA,JA, MAX,ISP(V),ISP(L), ISP(HEAD),P,IP, ISP(V), FLAG)
          IF (FLAG.NE.0)  GO TO 100
C
C----ALLOCATE STORAGE AND SYMMETRICALLY REORDER MATRIX
   1    IF ((PATH-2) * (PATH-3) * (PATH-4) * (PATH-5) .NE. 0)  GO TO 2
          TMP = (NSP+1) -      N
          Q   = TMP     - (IA(N+1)-1)
          IF (Q.LT.1)  GO TO 110
C
          DFLAG = PATH.EQ.4 .OR. PATH.EQ.5
          CALL SRO
     *       (N,  IP,  IA, JA, A,  ISP(TMP),  ISP(Q),  DFLAG)
C
   2    RETURN
C
C ** ERROR -- ERROR DETECTED IN MD
 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  MD -- MINIMUM DEGREE ALGORITHM (BASED ON ELEMENT MODEL)
C*********************************************************************
      SUBROUTINE  MD
     *     (N, IA,JA, MAX, V,L, HEAD,LAST,NEXT, MARK, FLAG)
C
C  DESCRIPTION
C
C    MD FINDS A MINIMUM DEGREE ORDERING OF THE ROWS AND COLUMNS OF A
C    SYMMETRIC MATRIX M STORED IN (IA,JA,A) FORMAT.
C
C
C  ADDITIONAL PARAMETERS
C
C    MAX  - DECLARED DIMENSION OF THE ONE-DIMENSIONAL ARRAYS V AND L;
C           MAX MUST BE AT LEAST  N+2K,  WHERE K IS THE NUMBER OF
C           NONZEROES IN THE STRICT UPPER TRIANGLE OF M
C
C    V    - INTEGER ONE-DIMENSIONAL WORK ARRAY;  DIMENSION = MAX
C
C    L    - INTEGER ONE-DIMENSIONAL WORK ARRAY;  DIMENSION = MAX
C
C    HEAD - INTEGER ONE-DIMENSIONAL WORK ARRAY;  DIMENSION = N
C
C    LAST - INTEGER ONE-DIMENSIONAL ARRAY USED TO RETURN THE PERMUTATION
C           OF THE ROWS AND COLUMNS OF M CORRESPONDING TO THE MINIMUM
C           DEGREE ORDERING;  DIMENSION = N
C
C    NEXT - INTEGER ONE-DIMENSIONAL ARRAY USED TO RETURN THE INVERSE OF
C           THE PERMUTATION RETURNED IN LAST;  DIMENSION = N
C
C    MARK - INTEGER ONE-DIMENSIONAL WORK ARRAY (MAY BE THE SAME AS V);
C           DIMENSION = N
C
C    FLAG - INTEGER ERROR FLAG;  VALUES AND THEIR MEANINGS ARE -
C             0      NO ERRORS DETECTED
C             11N+1  INSUFFICIENT STORAGE IN MD
C
C
C  DEFINITIONS OF INTERNAL PARAMETERS
C
C    ---------+-----------------------------------------------------
C    V(S)     | VALUE FIELD OF LIST ENTRY
C    ---------+-----------------------------------------------------
C    L(S)     | LINK FIELD OF LIST ENTRY  (0 => END OF LIST)
C    ---------+-----------------------------------------------------
C    L(VI)    | POINTER TO ELEMENT LIST OF UNELIMINATED VERTEX VI
C    ---------+-----------------------------------------------------
C    L(EJ)    | POINTER TO BOUNDARY LIST OF ACTIVE ELEMENT EJ
C    ---------+-----------------------------------------------------
C    HEAD(D)  | VJ => VJ HEAD OF D-LIST D
C             |  0 => NO VERTEX IN D-LIST D
C
C
C             |                  VI UNELIMINATED VERTEX
C             |          VI IN EK           |       VI NOT IN EK
C    ---------+-----------------------------+-----------------------
C    NEXT(VI) | UNDEFINED BUT NONNEGATIVE   | VJ => VJ NEXT IN D-LIST
```

```
C             |                              |  0 => VI TAIL OF D-LIST
C    ---------+------------------------------+-------------------------
C    LAST(VI) | (NOT SET UNTIL MDP)          | -D => VI HEAD OF D-LIST D
C             |-VK => COMPUTE DEGREE         | VJ => VJ LAST IN D-LIST
C             | EJ => VI PROTOTYPE OF EJ     |  0 => VI NOT IN ANY D-LIST
C             |  0 => DO NOT COMPUTE DEGREE  |
C    ---------+------------------------------+-------------------------
C    MARK(VI) | MARK(VK)                     | NONNEGATIVE TAG < MARK(VK)
C
C
C             |                    VI ELIMINATED VERTEX
C             |      EI ACTIVE ELEMENT       |           OTHERWISE
C    ---------+------------------------------+-------------------------
C    NEXT(VI) | -J => VI WAS J-TH VERTEX     | -J => VI WAS J-TH VERTEX
C             |       TO BE ELIMINATED       |       TO BE ELIMINATED
C    ---------+------------------------------+-------------------------
C    LAST(VI) |  M => SIZE OF EI = M         | UNDEFINED
C    ---------+------------------------------+-------------------------
C    MARK(VI) | -M => OVERLAP COUNT OF EI    | UNDEFINED
C             |       WITH EK = M            |
C             | OTHERWISE NONNEGATIVE TAG    |
C             |       < MARK(VK)             |
C
C-----------------------------------------------------------------------
C
      INTEGER  IA(1), JA(1),  V(1), L(1),  HEAD(1), LAST(1), NEXT(1),
     *    MARK(1),  FLAG,  TAG, DMIN, VK,EK, TAIL
      EQUIVALENCE  (VK,EK)
C
C----INITIALIZATION
      TAG = 0
      CALL  MDI
     *    (N, IA,JA, MAX,V,L, HEAD,LAST,NEXT, MARK,TAG, FLAG)
      IF (FLAG.NE.0)  RETURN
C
      K = 0
      DMIN = 1
C
C----WHILE  K < N  DO
   1  IF (K.GE.N)  GO TO 4
C
C------SEARCH FOR VERTEX OF MINIMUM DEGREE
   2    IF (HEAD(DMIN).GT.0)  GO TO 3
          DMIN = DMIN + 1
          GO TO 2
C
C------REMOVE VERTEX VK OF MINIMUM DEGREE FROM DEGREE LIST
   3    VK = HEAD(DMIN)
        HEAD(DMIN) = NEXT(VK)
        IF (HEAD(DMIN).GT.0)  LAST(HEAD(DMIN)) = -DMIN
C
C------NUMBER VERTEX VK, ADJUST TAG, AND TAG VK
        K = K+1
        NEXT(VK) = -K
        LAST(EK) = DMIN - 1
        TAG = TAG + LAST(EK)
        MARK(VK) = TAG
```

```
C
C------FORM ELEMENT EK FROM UNELIMINATED NEIGHBORS OF VK
          CALL  MDM
     *       (VK,TAIL, V,L, LAST,NEXT, MARK)
C
C------PURGE INACTIVE ELEMENTS AND DO MASS ELIMINATION
          CALL  MDP
     *       (K,EK,TAIL, V,L, HEAD,LAST,NEXT, MARK)
C
C------UPDATE DEGREES OF UNELIMINATED VERTICES IN EK
          CALL  MDU
     *       (EK,DMIN, V,L, HEAD,LAST,NEXT, MARK)
C
          GO TO 1
C
C----GENERATE INVERSE PERMUTATION FROM PERMUTATION
   4      DO 6 K=1,N
            NEXT(K) = -NEXT(K)
   6        LAST(NEXT(K)) = K
C
          RETURN
          END
C
C*********************************************************************
C  MDI -- INITIALIZATION
C*********************************************************************
          SUBROUTINE  MDI
     *       (N, IA,JA, MAX,V,L, HEAD,LAST,NEXT, MARK,TAG, FLAG)
          INTEGER  IA(1), JA(1),  V(1), L(1),  HEAD(1), LAST(1), NEXT(1),
     *       MARK(1), TAG,  FLAG,  SFS, VI,DVI, VJ
C
C----INITIALIZE DEGREES, ELEMENT LISTS, AND DEGREE LISTS
          DO 1 VI=1,N
            MARK(VI) = 1
            L(VI) = 0
   1        HEAD(VI) = 0
          SFS = N+1
C
C----CREATE NONZERO STRUCTURE
C----FOR EACH NONZERO ENTRY A(VI,VJ) IN STRICT UPPER TRIANGLE
          DO 3 VI=1,N
            JMIN = IA(VI)
            JMAX = IA(VI+1) - 1
            IF (JMIN.GT.JMAX)  GO TO 3
            DO 2 J=JMIN,JMAX
              VJ = JA(J)
              IF (VI.GE.VJ)  GO TO 2
                IF (SFS.GE.MAX)  GO TO 101
C
C------ENTER VJ IN ELEMENT LIST FOR VI
                MARK(VI) = MARK(VI) + 1
                V(SFS) = VJ
                L(SFS) = L(VI)
                L(VI) = SFS
                SFS = SFS+1
C
C------ENTER VI IN ELEMENT LIST FOR VJ
```

```
              MARK(VJ) = MARK(VJ) + 1
              V(SFS) = VI
              L(SFS) = L(VJ)
              L(VJ) = SFS
              SFS = SFS+1
    2       CONTINUE
    3     CONTINUE
C
C----CREATE DEGREE LISTS AND INITIALIZE MARK VECTOR
        DO 4 VI=1,N
          DVI = MARK(VI)
          NEXT(VI) = HEAD(DVI)
          HEAD(DVI) = VI
          LAST(VI) = -DVI
          IF (NEXT(VI).GT.0)  LAST(NEXT(VI)) = VI
    4     MARK(VI) = TAG
C
        RETURN
C
C ** ERROR -- INSUFFICIENT STORAGE
  101   FLAG = 9*N + VI
        RETURN
        END
C
C*======================================================================*
C  MDM -- FORM ELEMENT FROM UNELIMINATED NEIGHBORS OF VK
C*======================================================================*
        SUBROUTINE  MDM
     *     (VK,TAIL, V,L, LAST,NEXT, MARK)
        INTEGER  VK, TAIL,  V(1), L(1),   LAST(1), NEXT(1),   MARK(1),
     *     TAG, S,LS,VS,ES, B,LB,VB, BLP,BLPMAX
        EQUIVALENCE  (VS, ES)
C
C----INITIALIZE TAG AND LIST OF UNELIMINATED NEIGHBORS
        TAG = MARK(VK)
        TAIL = VK
C
C----FOR EACH VERTEX/ELEMENT VS/ES IN ELEMENT LIST OF VK
        LS = L(VK)
    1   S = LS
        IF (S.EQ.0)  GO TO 5
          LS = L(S)
          VS = V(S)
          IF (NEXT(VS).LT.0)  GO TO 2
C
C------IF VS IS UNELIMINATED VERTEX, THEN TAG AND APPEND TO LIST OF
C------UNELIMINATED NEIGHBORS
           MARK(VS) = TAG
           L(TAIL) = S
           TAIL = S
           GO TO 4
C
C------IF ES IS ACTIVE ELEMENT, THEN ...
C--------FOR EACH VERTEX VB IN BOUNDARY LIST OF ELEMENT ES
    2      LB = L(ES)
           BLPMAX = LAST(ES)
           DO 3 BLP=1,BLPMAX
```

```
              B = L8
              LB = L(B)
              VB = V(B)
C
C----------IF VB IS UNTAGGED VERTEX, THEN TAG AND APPEND TO LIST OF
C----------UNELIMINATED NEIGHBORS
              IF (MARK(VB).GE.TAG)  GO TO 3
                MARK(VB) = TAG
                L(TAIL) = B
                TAIL = B
   3          CONTINUE
C
C--------MARK ES INACTIVE
             MARK(ES) = TAG
C
   4       GO TO 1
C
C----TERMINATE LIST OF UNELIMINATED NEIGHBORS
   5     L(TAIL) = 0
C
         RETURN
         END
C
C*************************************************************************
C  MDP -- PURGE INACTIVE ELEMENTS AND DO MASS ELIMINATION
C*************************************************************************
         SUBROUTINE  MDP
     *      (K,EK,TAIL, V,L, HEAD,LAST,NEXT, MARK)
         INTEGER  EK, TAIL,  V(1), L(1),  HEAD(1), LAST(1), NEXT(1),
     *      MARK(1),  TAG, FREE, LI,VI,LVI,EVI, S,LS,ES, ILP,ILPMAX
C
C----INITIALIZE TAG
         TAG = MARK(EK)
C
C----FOR EACH VERTEX VI IN EK
         LI = EK
         ILPMAX = LAST(EK)
         IF (ILPMAX.LE.0)  GO TO 12
         DO 11 ILP=1,ILPMAX
           I = LI
           LI = L(I)
           VI = V(LI)
C
C------REMOVE VI FROM DEGREE LIST
           IF (LAST(VI).EQ.0)  GO TO 3
             IF (LAST(VI).GT.0)  GO TO 1
               HEAD(-LAST(VI)) = NEXT(VI)
               GO TO 2
   1           NEXT(LAST(VI)) = NEXT(VI)
   2         IF (NEXT(VI).GT.0)  LAST(NEXT(VI)) = LAST(VI)
C
C------REMOVE INACTIVE ITEMS FROM ELEMENT LIST OF VI
   3       LS = VI
   4       S = LS
           LS = L(S)
           IF (LS.EQ.0)  GO TO 6
             ES = V(LS)
```

```
            IF (MARK(ES).LT.TAG)  GO TO 5
              FREE = LS
              L(S) = L(LS)
              LS = S
   5        GO TO 4
C
C------IF VI IS INTERIOR VERTEX, THEN REMOVE FROM LIST AND ELIMINATE
   6      LVI = L(VI)
          IF (LVI.NE.0)  GO TO 7
            L(I) = L(LI)
            LI = I
C
            K = K+1
            NEXT(VI) = -K
            LAST(EK) = LAST(EK) - 1
            GO TO 11
C
C------ELSE ...
C--------CLASSIFY VERTEX VI
   7        IF (L(LVI).NE.0)  GO TO 9
              EVI = V(LVI)
              IF (NEXT(EVI).GE.0)  GO TO 9
                IF (MARK(EVI).LT.0)  GO TO 8
C
C----------IF VI IS PROTOTYPE VERTEX, THEN MARK AS SUCH, INITIALIZE
C----------OVERLAP COUNT FOR CORRESPONDING ELEMENT, AND MOVE VI TO END
C----------OF BOUNDARY LIST
                  LAST(VI) = EVI
                  MARK(EVI) = -1
                  L(TAIL) = LI
                  TAIL = LI
                  L(I) = L(LI)
                  LI = I
                  GO TO 10
C
C----------ELSE IF VI IS DUPLICATE VERTEX, THEN MARK AS SUCH AND ADJUST
C----------OVERLAP COUNT FOR CORRESPONDING ELEMENT
   8              LAST(VI) = 0
                  MARK(EVI) = MARK(EVI) - 1
                  GO TO 10
C
C----------ELSE MARK VI TO COMPUTE DEGREE
   9              LAST(VI) = -EK
C
C--------INSERT EK IN ELEMENT LIST OF VI
  10        V(FREE) = EK
            L(FREE) = L(VI)
            L(VI) = FREE
  11      CONTINUE
C
C----TERMINATE BOUNDARY LIST
  12    L(TAIL) = 0
C
        RETURN
        END
C
C*******************************************************************************
```

```
C  MDU -- UPDATE DEGREES OF UNELIMINATED VERTICES IN EK
C*********************************************************************
        SUBROUTINE  MDU
     *     (EK,DMIN, V,L, HEAD,LAST,NEXT, MARK)
        INTEGER  EK, DMIN,  V(1), L(1),  HEAD(1), LAST(1), NEXT(1),
     *     MARK(1),  TAG, VI,EVI,DVI, S,VS,ES, B,VB, ILP,ILPMAX,
     *     BLP,BLPMAX
        EQUIVALENCE  (VS, ES)
C
C----INITIALIZE TAG
        TAG = MARK(EK) - LAST(EK)
C
C----FOR EACH VERTEX VI IN EK
        I = EK
        ILPMAX = LAST(EK)
        IF (ILPMAX.LE.0)  GO TO 11
        DO 10 ILP=1,ILPMAX
          I = L(I)
          VI = V(I)
          IF (LAST(VI))  1, 10, 8
C
C------IF VI NEITHER PROTOTYPE NOR DUPLICATE VERTEX, THEN MERGE ELEMENTS
C------TO COMPUTE DEGREE
   1        TAG = TAG + 1
            DVI = LAST(EK)
C
C--------FOR EACH VERTEX/ELEMENT VS/ES IN ELEMENT LIST OF VI
            S = L(VI)
   2        S = L(S)
            IF (S.EQ.0)  GO TO 9
              VS = V(S)
              IF (NEXT(VS).LT.0)  GO TO 3
C
C----------IF VS IS UNELIMINATED VERTEX, THEN TAG AND ADJUST DEGREE
                MARK(VS) = TAG
                DVI = DVI + 1
                GO TO 5
C
C----------IF ES IS ACTIVE ELEMENT, THEN EXPAND
C------------CHECK FOR OUTMATCHED VERTEX
   3            IF (MARK(ES).LT.0)  GO TO 6
C
C------------FOR EACH VERTEX VB IN ES
                B = ES
                BLPMAX = LAST(ES)
                DO 4 BLP=1,BLPMAX
                  B = L(B)
                  VB = V(B)
C
C--------------IF VB IS UNTAGGED, THEN TAG AND ADJUST DEGREE
                  IF (MARK(VB).GE.TAG)  GO TO 4
                    MARK(VB) = TAG
                    DVI = DVI + 1
   4              CONTINUE
C
   5          GO TO 2
C
```

```
C------ELSE IF VI IS OUTMATCHED VERTEX, THEN ADJUST OVERLAPS BUT DO NOT
C------COMPUTE DEGREE
   6        LAST(VI) = 0
            MARK(ES) = MARK(ES) - 1
   7        S = L(S)
            IF (S.EQ.0)  GO TO 10
              ES = V(S)
              IF (MARK(ES).LT.0)  MARK(ES) = MARK(ES) - 1
              GO TO 7
C
C------ELSE IF VI IS PROTOTYPE VERTEX, THEN CALCULATE DEGREE BY
C------INCLUSION/EXCLUSION AND RESET OVERLAP COUNT
   8        EVI = LAST(VI)
            DVI = LAST(EK) + LAST(EVI) + MARK(EVI)
            MARK(EVI) = 0
C
C------INSERT VI IN APPROPRIATE DEGREE LIST
   9      NEXT(VI) = HEAD(DVI)
          HEAD(DVI) = VI
          LAST(VI) = -DVI
          IF (NEXT(VI).GT.0)  LAST(NEXT(VI)) = VI
          IF (DVI.LT.DMIN)  DMIN = DVI
C
  10      CONTINUE
C
  11    RETURN
        END
C*********************************************************************
C*********************************************************************
C  SRO -- SYMMETRIC REORDERING OF SPARSE SYMMETRIC MATRIX
C*********************************************************************
        SUBROUTINE  SRO
     *     (N, IP, IA,JA,A, Q, R, DFLAG)
C
C  DESCRIPTION
C
C    THE NONZERO ENTRIES OF THE MATRIX M ARE ASSUMED TO BE STORED
C    SYMMETRICALLY IN (IA,JA,A) FORMAT (I.E., NOT BOTH M(I,J) AND M(J,I)
C    ARE STORED IF I NE J).
C
C    SRO DOES NOT REARRANGE THE ORDER OF THE ROWS, BUT DOES MOVE
C    NONZEROES FROM ONE ROW TO ANOTHER TO ENSURE THAT IF M(I,J) WILL BE
C    IN THE UPPER TRIANGLE OF M WITH RESPECT TO THE NEW ORDERING, THEN
C    M(I,J) IS STORED IN ROW I (AND THUS M(J,I) IS NOT STORED);  WHEREAS
C    IF M(I,J) WILL BE IN THE STRICT LOWER TRIANGLE OF M, THEN M(J,I) IS
C    STORED IN ROW J (AND THUS M(I,J) IS NOT STORED).
C
C
C  ADDITIONAL PARAMETERS
C
C    Q      - INTEGER ONE-DIMENSIONAL WORK ARRAY;  DIMENSION = N
C
C    R      - INTEGER ONE-DIMENSIONAL WORK ARRAY;  DIMENSION = NUMBER OF
C             NONZERO ENTRIES IN THE UPPER TRIANGLE OF M
C
C    DFLAG - LOGICAL VARIABLE;  IF DFLAG = .TRUE., THEN STORE NONZERO
C             DIAGONAL ELEMENTS AT THE BEGINNING OF THE ROW
```

```
C
C-----------------------------------------------------------------------
C
      INTEGER  IP(1),  IA(1), JA(1),  Q(1), R(1)
      REAL  A(1),  AK
C...  DOUBLE PRECISION  A(1),  AK
      LOGICAL  DFLAG
C
C
C--PHASE 1 -- FIND ROW IN WHICH TO STORE EACH NONZERO
C----INITIALIZE COUNT OF NONZEROES TO BE STORED IN EACH ROW
      DO 1 I=1,N
 1      Q(I) = 0
C
C----FOR EACH NONZERO ELEMENT A(J)
      DO 3 I=1,N
        JMIN = IA(I)
        JMAX = IA(I+1) - 1
        IF (JMIN.GT.JMAX)  GO TO 3
        DO 2 J=JMIN,JMAX
C
C--------FIND ROW (=R(J)) AND COLUMN (=JA(J)) IN WHICH TO STORE A(J) ...
          K = JA(J)
          IF (IP(K).LT.IP(I))  JA(J) = I
          IF (IP(K).GE.IP(I))  K = I
          R(J) = K
C
C--------... AND INCREMENT COUNT OF NONZEROES (=Q(R(J)) IN THAT ROW
 2        Q(K) = Q(K) + 1
 3      CONTINUE
C
C
C--PHASE 2 -- FIND NEW IA AND PERMUTATION TO APPLY TO (JA,A)
C----DETERMINE POINTERS TO DELIMIT ROWS IN PERMUTED (JA,A)
      DO 4 I=1,N
        IA(I+1) = IA(I) + Q(I)
 4      Q(I) = IA(I+1)
C
C----DETERMINE WHERE EACH (JA(J),A(J)) IS STORED IN PERMUTED (JA,A)
C----FOR EACH NONZERO ELEMENT (IN REVERSE ORDER)
      ILAST = 0
      JMIN = IA(1)
      JMAX = IA(N+1) - 1
      J = JMAX
      DO 6 JDUMMY=JMIN,JMAX
        I = R(J)
        IF (.NOT.DFLAG .OR. JA(J).NE.I .OR. I.EQ.ILAST)  GO TO 5
C
C------IF DFLAG, THEN PUT DIAGONAL NONZERO AT BEGINNING OF ROW
          R(J) = IA(I)
          ILAST = I
          GO TO 6
C
C------PUT (OFF-DIAGONAL) NONZERO IN LAST UNUSED LOCATION IN ROW
 5        Q(I) = Q(I) - 1
          R(J) = Q(I)
C
```

```
  6       J = J-1
C
C
C--PHASE 3 -- PERMUTE (JA,A) TO UPPER TRIANGULAR FORM (WRT NEW ORDERING)
        DO 8 J=JMIN,JMAX
  7       IF (R(J).EQ.J)  GO TO 8
            K = R(J)
            R(J) = R(K)
            R(K) = K
            JAK = JA(K)
            JA(K) = JA(J)
            JA(J) = JAK
            AK = A(K)
            A(K) = A(J)
            A(J) = AK
            GO TO 7
  8       CONTINUE
C
        RETURN
        END
C                                                                    1/16/81
C***********************************************************************
C  SDRV -- DRIVER FOR SPARSE SYMMETRIC POSITIVE DEFINITE MATRIX ROUTINES
C***********************************************************************
        SUBROUTINE SDRV
     *     (N, P,IP, IA,JA,A, B, Z, NSP,ISP,RSP,ESP, PATH, FLAG)
C
C  DESCRIPTION
C
C    SDRV SOLVES SPARSE SYMMETRIC POSITIVE DEFINITE SYSTEMS OF LINEAR
C    EQUATIONS.  THE SOLUTION PROCESS IS DIVIDED INTO THREE STAGES --
C
C      SSF - THE COEFFICIENT MATRIX M IS FACTORED SYMBOLICALLY TO
C            DETERMINE WHERE FILLIN WILL OCCUR DURING THE NUMERIC
C            FACTORIZATION.
C
C      SNF - M IS FACTORED NUMERICALLY INTO THE PRODUCT UT-D-U, WHERE
C            D IS DIAGONAL AND U IS UNIT UPPER TRIANGULAR.
C
C      SNS - THE LINEAR SYSTEM  MX = B  IS SOLVED USING THE UT-D-U
C            FACTORIZATION FROM SNF.
C
C    FOR SEVERAL SYSTEMS WITH THE SAME COEFFICIENT MATRIX, SSF AND SNF
C    NEED BE DONE ONLY ONCE (FOR THE FIRST SYSTEM);  THEN SNS IS DONE
C    ONCE FOR EACH ADDITIONAL RIGHT-HAND SIDE.  FOR SEVERAL SYSTEMS
C    WHOSE COEFFICIENT MATRICES HAVE THE SAME NONZERO STRUCTURE, SSF
C    NEED BE DONE ONLY ONCE (FOR THE FIRST SYSTEM);  THEN SNF AND SNS
C    ARE DONE ONCE FOR EACH ADDITIONAL SYSTEM.
C
C
C  STORAGE OF SPARSE MATRICES
C
C    THE NONZERO ENTRIES OF THE MATRIX M ARE STORED ROW-BY-ROW IN THE
C    ARRAY A.  TO IDENTIFY THE INDIVIDUAL NONZERO ENTRIES IN EACH ROW,
C    WE NEED TO KNOW IN WHICH COLUMN EACH ENTRY LIES.  THESE COLUMN
C    INDICES ARE STORED IN THE ARRAY JA;  I.E., IF  A(K) = M(I,J),  THEN
C    JA(K) = J.  TO IDENTIFY THE INDIVIDUAL ROWS, WE NEED TO KNOW WHERE
```

```
C    EACH ROW STARTS.  THESE ROW POINTERS ARE STORED IN THE ARRAY IA;
C    I.E., IF M(I,J) IS THE FIRST NONZERO ENTRY (STORED) IN THE I-TH ROW
C    AND  A(K) = M(I,J),  THEN  IA(I) = K.  MOREOVER, IA(N+1) POINTS TO
C    THE FIRST LOCATION FOLLOWING THE LAST ELEMENT IN THE LAST ROW.
C    THUS, THE NUMBER OF ENTRIES IN THE I-TH ROW IS  IA(I+1) - IA(I),
C    THE NONZERO ENTRIES IN THE I-TH ROW ARE STORED CONSECUTIVELY IN
C
C            A(IA(I)),  A(IA(I)+1),  ..., A(IA(I+1)-1),
C
C    AND THE CORRESPONDING COLUMN INDICES ARE STORED CONSECUTIVELY IN
C
C            JA(IA(I)), JA(IA(I)+1), ..., JA(IA(I+1)-1).
C
C    SINCE THE COEFFICIENT MATRIX IS SYMMETRIC, ONLY THE NONZERO ENTRIES
C    IN THE UPPER TRIANGLE NEED BE STORED, FOR EXAMPLE, THE MATRIX
C
C               ( 1  0  2  3  0 )
C               ( 0  4  0  0  0 )
C          M = ( 2  0  5  6  0 )
C               ( 3  0  6  7  8 )
C               ( 0  0  0  8  9 )
C
C    COULD BE STORED AS
C
C            | 1  2  3  4  5  6  7  8  9 10 11 12 13
C       -----+-------------------------------------
C       IA   | 1  4  5  8 12 14
C       JA   | 1  3  4  2  1  3  4  1  3  4  5  4  5
C        A   | 1  2  3  4  2  5  6  3  6  7  8  8  9
C
C    OR (SYMMETRICALLY) AS
C
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
C  REORDERING THE ROWS AND COLUMNS OF M
C
C    A SYMMETRIC PERMUTATION OF THE ROWS AND COLUMNS OF THE COEFFICIENT
C    MATRIX M (E.G., WHICH REDUCES FILLIN OR ENHANCES NUMERICAL
C    STABILITY) MUST BE SPECIFIED.  THE SOLUTION Z IS RETURNED IN THE
C    ORIGINAL ORDER.
C
C    TO SPECIFY THE TRIVIAL ORDERING (I.E., THE IDENTITY PERMUTATION),
C    SET  P(I) = IP(I) = I,  I=1,...,N.  IN THIS CASE, P AND IP CAN BE
C    THE SAME ARRAY.
C
C    IF A NONTRIVIAL ORDERING (I.E., NOT THE IDENTITY PERMUTATION) IS
C    SPECIFIED AND M IS STORED SYMMETRICALLY (I.E., NOT BOTH M(I,J) AND
C    M(J,I) ARE STORED FOR I NE J), THEN ODRV SHOULD BE CALLED (WITH
C    PATH = 3 OR 5) TO SYMMETRICALLY REORDER (IA,JA,A) BEFORE CALLING
C    SDRV.  THIS IS TO ENSURE THAT IF M(I,J) WILL BE IN THE UPPER
C    TRIANGLE OF M WITH RESPECT TO THE NEW ORDERING, THEN M(I,J) IS
C    STORED IN ROW I (AND THUS M(J,I) IS NOT STORED);  WHEREAS IF M(I,J)
```