Further Programs for the Solution of
Large Sparse Systems of Linear Equations

by Werner
K. Mesztenyi and C. Rheinboldt

Abstract

A package of FORTRAN subroutines is presented for the solution of nonsymmetric or symmetric sparse linear systems by triangular decomposition. Two principal aims are (1) to handle matrices which originally fit into primary core storage but do so no longer after decomposition, and (2) to solve a sequence of linear systems all of which have the same sparsity structure by generating—in secondary storage—a record of the decomposition process in the form of an integer array. Some experimental results using the package are included.
FURTHER PROGRAMS FOR THE SOLUTION OF
LARGE SPARSE SYSTEMS OF LINEAR EQUATIONS\textsuperscript{1)}

by

Charles K. Mesztenyi\textsuperscript{2)} and Werner C. Rheinboldt\textsuperscript{2)}

1. Introduction

In a previous report \cite{1} a package of FORTRAN subroutines was presented for the solution of a linear system

\begin{equation}
Ax = b
\end{equation}

based on triangular decomposition of the (symmetric or nonsymmetric) matrix $A$. The underlying data structure was motivated by a more general arc-graph structure discussed in \cite{2}.

The programs given here have the same purpose but pursue the following two different aims:

a. To handle matrices which originally fit into primary core storage but do so no longer after decomposition.

b. To solve a sequence of systems (1) all of which have the same sparsity structure. This case arises, for example, in the solution of nonlinear systems by Newton's method.

The first aim is accomplished during decomposition by writing the decomposed part of the matrix into secondary storage and using its place for newly introduced nonzero elements. In order to meet the second aim

\textsuperscript{1)}This work was supported in part by the National Science Foundation under Grant GJ-35568 and the Office of Naval Research under Grant N00014-67-A-0239-0021 (NR-044-431).

\textsuperscript{2)}Computer Science Center, University of Maryland, College Park, Md. 20742.
we follow an idea in [3] and generate--in secondary storage--a record of the decomposition process in the form of an integer array. This record can be used to decompose any matrix with the same sparsity structure provided there are no round-off problems.

2. Some Background

The desired triangular decomposition of the \( n \times n \), nonsingular matrix \( A \) has the form

\[
PAQ = LU, \quad L = I + L^0, \tag{2}
\]

where \( L^0 \) is strictly lower triangular, \( U \) upper triangular, and the permutation matrices \( P, Q \) define the pivoting sequence. The decomposition is accomplished in \( n \) steps, such that

\[
P_iA_iQ_i = (I + L_i^0)U_i + A_i, \quad i = 0, 1, \ldots, n \tag{3}
\]

where the first \( i \) rows and \( i \) columns of \( A_i \), the last \( n-i \) columns of \( L_i^0 \), and the last \( n-i \) rows of \( U_i \) are zero, respectively. Moreover, \( PP_i^TL_i^0 \) has the same first \( i \) columns as \( L^0 \) and \( U_iQ_i^T \) the same first \( i \) rows as \( U \). This latter fact allows us to keep \( L_i^0 \) and \( U_i \) in secondary storage.

Let \( \eta(B) \) denote the number of nonzero elements of any matrix \( B \). Then the storage required before and after decomposition is of the order of \( m_0 = \eta(A) \) and \( m_2 = \eta(U) + \eta(L^0) \), respectively. Furthermore, \( m_1 = \max_i \eta(A_i) \) is the maximal storage needed for the matrices \( A_i \). Clearly, we have
m_0 \leq m_1 \leq m_2 \text{ and, in practice, it turns out that } m_2 - m_0 \text{ is very much larger than } m_1 - m_0. \text{ In fact, sometimes we found } m_1 \text{ to be equal to } m_0 \text{ (see Section 5). Hence by retaining only the } A_i \text{ in primary storage we require, in general, only little more storage than for } A \text{ itself.}

The basic storage structure allows for easy modification of the pivoting strategy. In fact, in the nonsymmetric case the pivot selection is handled by an easily replaceable subroutine. We use here the well-known minimal degree algorithm. If } S_i \text{ is the set of nonzero elements of } A_i, \text{ then for any } x \in S_i \text{ we denote by } R_i(x) \text{ and } C_i(x) \text{ the subsets of } S_i \text{ consisting of the elements in the same row and column as } x, \text{ respectively. Now, with } E_i(x) = R_i(x) \text{ if } |R_i(x)| \leq |C_i(x)| \text{ and otherwise } E_i(x) = C_i(x), \text{ the set of potential pivots is given by}

\begin{equation}
S_i^0 = \{x \in S_i; |a(x)| \geq \mu \max_{z \in E_i(x)} |a(z)|\}.
\end{equation}

Here } a(z) \text{ is the value of the matrix element corresponding to } z \text{ and } \mu \in [0,1], \text{ a user-defined parameter. The } i \text{th pivot is then the element } x \in S_i^0 \text{ for which } (|R_i(x)|-1)(|C_i(x)|-1) \text{ is minimal. Generally, with decreasing } \mu \text{ the fill-in decreases while the round-off influence increases.}

For symmetric } A \text{ it is assumed that the pivots remain on the main diagonal and hence that } Q = P^T. \text{ In that case each matrix } A_i \text{ is again symmetric. If } D_i \text{ is the set of nonzero diagonal elements of } A_i, \text{ then the } i \text{th pivot is the element } x \text{ of the set}

\begin{equation}
D_i^0 = \{z \in D_i; |a(z)| \geq \mu \max_{y \in D_i} |a(y)|\}
\end{equation}

for which the number of nonzero elements in its row is minimal.
It is theoretically possible to use the value $\mu = 0$. In that case, (4) and (5) show that any nonzero element of $S_i$ or $D_i$, respectively, is a potential pivot. Then the pivot selection depends only on the sparsity structure and not on the elements of the matrix—but, of course, the round-off influence may be considerable.

3. **Basic Storage Arrangements**

3.1 **Matrices in Primary Storage:** As mentioned before, the basic storage structure used here is the same as that in [1]. We summarize it briefly.

Set $N = \{1, 2, \ldots, n\}$ and let $S \subseteq N \times N$ be the set of locations corresponding to the nonzero elements of a given $n \times n$ matrix $A$. We number the elements of $S$ from $n+1$ to $n+m$, $m = |S|$, that is, we introduce a bijective mapping

$$v: S \rightarrow \{n+1, \ldots, n+m\}.$$  

Now define two integer arrays $R_Y$ and $O_Y$ each of length $n+m$ in which the relative addresses $n+1, \ldots, n+m$ correspond to the elements of $S$ in the order provided by $v$. The images $v(R_i)$ of the row sets

$$R_i = \{(i, k) \in S; \text{some } k \in N\}, \quad i \in N$$

form a partition of $\{n+1, \ldots, n+m\}$. For any set $v(R_i) = \{i_1, \ldots, i_k\}$ we link the locations $i, i_1, i_2, \ldots, i_k$ into a circular list

$$i_1 = R_Y(i), \quad i_{j+1} = R_Y(i_j), \quad j = 1, \ldots, k-1, \quad i = R(i_k)$$

where for practical reasons
(8) \[ i_1 > i_2 > \cdots > i_k > i. \]

Analogously, we proceed with the images \( v(C_i) \) of the column sets

\[ C_j = \{(k,j) \in S; \text{some } k \in \mathbb{N}, j \in \mathbb{N}\} \]

in the array \( CY \).

In order to store the associated matrix elements a third array \( A \) is, of course, needed. Thus, for example, the matrix

\[
\begin{pmatrix}
0 & 1 & 0 & 3 \\
0 & 0 & 5 & 0 \\
-1 & 0 & 2 & 0 \\
0 & -2 & 0 & 0
\end{pmatrix}
\]

may be stored as follows:

<table>
<thead>
<tr>
<th>loc</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RY</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>CY</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-1</td>
<td>1</td>
<td>-2</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

We shall refer to \( RY \) and \( CY \) as the sparsity structure arrays and to \( A \) as the coefficient array.

For symmetric \( A \) the set \( S \) only needs to be the set of locations of the nonzero elements in the upper (or lower) triangle (including the diagonal) of \( A \). Moreover, we assume always that in the symmetric case all diagonal elements are nonzero. Then the first \( n \) cells of the sparsity structure arrays \( RY \) and \( CY \) are no longer needed if (6) is changed to
\( v : S \to \{1, 2, \ldots, m\}, \quad v(i, i) = i, i=1, \ldots, n \). \\
There is no need to repeat the details; the resulting data arrangement should be self-evident from the following example:

\[
\begin{pmatrix}
1 & 0 & -1 & 0 \\
0 & 2 & 0 & -2 \\
-1 & 0 & 3 & 0 \\
0 & -2 & 0 & 4
\end{pmatrix}
\]

\[
\begin{array}{c|ccccccc}
\text{loc} & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\text{RY} & 5 & 6 & 3 & 4 & 1 & 2 \\
\hline
\text{CY} & 1 & 2 & 5 & 6 & 3 & 4 \\
\hline
\text{A} & 1 & 2 & 3 & 4 & -1 & -2
\end{array}
\]

During decomposition, this storage arrangement is used for the matrices \( A_i, i = 0, \ldots, n \). When a nonzero element of \( A_i \) remains in \( A_{i+1} \) its position in the RY, CY, AY arrays is maintained. After each pivoting step the pivot row and column are written on secondary storage and the corresponding cells in the storage arrays are freed, that is, the circular linkages containing these elements are modified appropriately. The resulting free locations are reassigned when fill-in occurs.

3.2 Triangular Matrices in Secondary Storage: The programs here are written for use with a random-access secondary storage device. Some information about the necessary I/O routines is provided in Section 4.1 below.

In the nonsymmetric case the triangular matrices \( L \) and \( U \) are written as two arrays of pairs of numbers. The L-array contains the columns of \( L \) in consecutive order and each column has the form

\[
\begin{align*}
&-i_c, -i_r \\
j_1, j_1, i_c \\
&\vdots \\
j_k, j_k, i_c
\end{align*}
\]
where $i_c$ and $i_r$ are the column- and row-index, respectively, of the pivot (stored negatively) and $j_i$ represents the row index and $\ell_{j_i,i_c}$ the value of each nonzero element in the particular column. Similarly the U-array contains the rows of $U$ in consecutive order, each of them in the form

$$
\begin{align*}
  j_1, u_{i_r}, j_1 \\
  \vdots \\
  j_k, u_{i_r}, j_k \\
  -i_c, p_{i_c}
\end{align*}
$$

Here $i_c$ is the column index of the pivot and $p_i$ its value, while $j_i, u_{i_r}, j_i$ denote the column index and value, respectively, of each nonzero element in the row. The entire U-array is initialized by a dummy pair $-1,-1$.

For the backsubstitution programs it is assumed that the L-array is read forward and the U-array backward.

In the symmetric case, there is, of course, no need for both the L- and U-array. Accordingly, only the L-array is set up containing the columns of $L$ in consecutive order, each one in the form

$$
\begin{align*}
  -i_d, d_{i_d} \\
  j_1, \ell_{j_1}, i_d \\
  \vdots \\
  j_k, \ell_{j_k}, i_d \\
  -i_d, d_{i_d}
\end{align*}
$$
Here $i_d$ is the index of the pivot (on the diagonal) and $d_i$ its value and $j_i, l_i, i_d$ have the previous meaning. The header at the beginning and end of each column is needed, since during backsubstitution the array is read once forward and once backward.

3.3 Representation of the Decomposition Record: As mentioned in the introduction, the programs can generate a record of the decomposition process for later use with any other matrix of the same sparsity type. This record is in the form of an array of positive integers in secondary storage. For each pivoting step the following information is recorded:

**Non-symmetric Case:**

\[
i_x, i_c, i_r, k_c, x_1, j_1, \ldots , x_{k_c}, j_{k_c} \\
k_r, y_1, m_1, \ldots , y_{k_r}, m_{k_r} \\
l_1, l_2, \ldots , l_t
\]

- $i_x$: relative location of the pivot in the RY,CY arrays
- $i_c, i_r$: the column- and row-index of the pivot, respectively
- $k_c, k_r$: the number of nonzero elements in the pivot-column and pivot-row, respectively
- $x_i, j_i$: relative location (in CY) and row-index, respectively, of the nonzero elements in the pivot column
- $y_i, m_i$: relative location (in RY) and column-index, respectively, of the nonzero elements in the pivot row
- $l_i$: relative locations of the elements in $A_i$ which must be modified at the step, $t = k_c \cdot k_r$.
Symmetric Case:

\[ i, k, y_1, m_1, \ldots, y_k, m_k \]
\[ \ell_1, \ldots, \ell_t \]

- \( i \) index of the pivot and hence also its relative location in the \( RY, CY \) arrays
- \( k \) the number of nonzero elements in the pivot row. Each of these elements is identified by a pair \( (y_j, m_j) \) as in the nonsymmetric case
- \( \ell_j \) the relative locations of the elements in \( A_i \) to be modified, \( t = k(k-1) \)

4. Description of the Programs

The package consists of four groups of subroutines with names INT, BLD, DEC, and SLV; in addition, there is a pivot selection routine PVT01 for the nonsymmetric case and a set of I/O routines for interface with the random storage device.

The INT programs initialize the storage area and have to be called first. The BLD routines establish the data structure described in Section 3.1 for the given matrix \( A \). Then the DEC routines are called to perform the decomposition of the matrix and/or to generate a record of the decomposition process. Finally, if applicable, the SLV routines are used to obtain the solution of the given system (1) by backsubstitution.

In general, any efficient routine for building up the basic data structure from given data about the matrix depends strongly on the details
of the files used. The BLI programs presented here avoid all assumptions about file formats, etc., by establishing the data structure one matrix element at a time. In other words, the chosen BLI routine has to be called once for each nonzero matrix element. For many practical purposes this may be inefficient. The routines were included principally for the sake of completeness; it should be easy to rewrite them for any specific application.

The names of all subroutines in the four principal groups are preceded by the letters S or N for the case of symmetric or nonsymmetric matrices, respectively. The names of the subroutines in the INT, BLI, DEC group are ending with one of the numerals 0, 1 or 01. This indicates the following alternatives:

0 - Initialize or build only the sparsity structure arrays of the matrix or generate a record of the decomposition based solely on the sparsity structure. These routines are only available for symmetric matrices; for nonsymmetric matrices it is not an advisable approach since the resulting round-off error could be severe.

1 - Initialize or build only the coefficient array for the matrix elements, or decompose the matrix using a previously generated record of a decomposition for matrices with the same sparsity structure.

01 - Initialize or build both the sparsity structure arrays and the coefficient array, or decompose the given matrix and, optionally, generate a record of the decomposition.

The pivot selection for the nonsymmetric case is performed by the routine PVT01. For the symmetric case, pivot selection is incorporated within the routines SDECO and SDECO1.
4.1 **Catalog of Subroutines:** In this subsection we list the various subroutines of the package together with their calling sequences and brief descriptions of their purposes. The arguments in the calling sequences are discussed in the next subsection.

**INT - Routines**

- **SINT0(MD,RY,CY,ND)**
  - Initialize the sparsity structure arrays of a symmetric matrix.

- **SINT01(MD,FD,RY,CY,A,AN,ND)**
  - Initialize the sparsity structure arrays and the coefficient array of a symmetric matrix.

- **SINT1(MD,FD,AN)**
  - Initialize the coefficient array of a symmetric matrix.

- **NINT01(MD,FD,RY,CY,AN,NDR,NDC)**
  - Initialize the sparsity structure arrays and the coefficient array of a nonsymmetric matrix.

- **NINT1(MD,FD,AN)**
  - Initialize the coefficient array of a nonsymmetric matrix.

**BLD - Routines**

All routines add a matrix element with value \( V \), row index \( I \), and column index \( J \) to the structure. Note that in the symmetric case only the nonzero elements in the upper (or lower) triangle and the diagonal should be given.
SBLD0(I,J,MD,RY,CY,ND)
Insert element (I,J) into the sparsity structure arrays of a symmetric matrix.

SBLD01(I,J,V,MD,FD,RY,CY,A,AN,ND)
Insert element (I,J) into the sparsity structure arrays of a symmetric matrix and a corresponding value V into the coefficient array.

SBLD1(I,J,V,MD,FD,A,AN)
Associate a value V to element (I,J) of a symmetric matrix. The V-values must be in the same order in which the (I,J)-values were read-in during prior construction of the corresponding sparsity structure arrays by SBLD0 or SBLD01.

NBLD01(I,J,V,MD,FD,RY,CY,A,AN,NDR,NDC)
Insert element (I,J) into the sparsity structure arrays of a nonsymmetric matrix and a corresponding value V into the coefficient array.

NBLD1(I,J,V,MD,FD,A,AN)
Associate a value V to element (I,J) of a nonsymmetric matrix. The V-values must be in the same order in which the (I,J)-values were read-in during prior construction of the corresponding sparsity structure arrays by NBLD01.
DEC - Routines

SDEC0 (MD, RY, CY, ND, IP, IE, IH)
Generate a record of the decomposition of a symmetric matrix on the basis of the given sparsity structure.

SDEC01 (MD, FD, RY, CY, A, AN, ND, IP, IE, IH)
Decompose a given symmetric matrix and optionally (MD(3)≠0) generate a record of the decomposition.

SDEC1 (MD, FD, A, AN, IE)
Decompose a given symmetric matrix using a previously generated decomposition record.

NDEC01 (MD, FD, RY, CY, A, AN, NDR, NDC, IPR, IPC, IE, IH, NG1, NG2)
Decompose a given nonsymmetric matrix and optionally (MD(3)≠0) generate a decomposition record.

NDEC1 (MD, FD, A, AN, IE, IH)
Decompose a given nonsymmetric matrix using a previously generated decomposition record.

Pivot Routine

PVT01 (I, N, IX, KR, KC, F, RY, CY, A, IPR, IPC, NDR, NDC, IE, IH)
Select the next pivot by the minimal degree algorithm during the decomposition of a nonsymmetric matrix. The routine is used by NDEC01.
SLV - Routines

These routines use the decomposed matrix in secondary storage. The right side of the system is given in the form of the input array X which in turn may be the same as the output array Y of the solution. The routines may be called repeatedly for different right sides.

SSLV(MD,X,Y)
Return the solution Y of the symmetric system with right side in X.

NSLV(MD,X,Y,AN)
Return the solution Y of the nonsymmetric system with right side in X.

I/O - Routines

The I/O routines for communication with the random access storage device are not in basic FORTRAN. They should be modified to suit a user's machine configuration. The routines have the following entries:

For I/O of decomposition record (array of positive integers)

DWI    - Initialize for writing.
DW(K)   - Write K as next entry of the array.
DWE    - Terminate writing.
DRI    - Initialize for reading.
DR(K)   - Return the next entry of the array in K.

For I/O of symmetric decomposed matrix (array of pairs)

SVWI   - Initialize for writing.
SVW(l,s) - Write (l,s) as next entry of the array. l-signed integer, s-real.
SVWE - Terminate writing.
SVRI - Initialize for reading (forward).
SVRF(ℓ,s) - Return the next entry of the array in ℓ and s.
SVRB(ℓ,s) - Return the previous entry of the array in ℓ and s.

For I/O of a nonsymmetric decomposed matrix (two arrays of pairs)

NVWI - Initialize both files for writing.
NVWF(ℓ,s) - Write (ℓ,s) as next entry of the L-array (ℓ-signed integer, s-real).
NVWB(ℓ,s) - Write (ℓ,s) as next entry of the U-array.
NVWE - Terminate writing of both files.
NVRI - Initialize for reading, file L forward, file U backward.

NVRF(ℓ,s) - Return next entry of L-array in (ℓ,s).
NVRB(ℓ,s) - Return previous entry of U-array in (ℓ,s).

The following Table 1 shows the usage of the I/O routines by the various main routines:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SD8C0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD8C01</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SD8C1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SSDLV</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SD8C01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD8C1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SSDLV</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

X - routine used
# - use is optional, depending on user's request
4.2 **Arguments**: The arguments in the various calling sequences either reference single values or data arrays. For simplicity the single variables are collected in two arrays, an integer array

$$\text{MD}(I), I = 1,2,\ldots,8$$

and a real array

$$\text{FD}(I), I = 1,2,\ldots,7.$$  

The first three values of MD and the first two of FD are to be supplied by the user; the others represent output of various other routines. Care should be taken that these latter values are not modified whenever they are still to be used as input by other routines.

The use of the various arguments by the routines in the package is summarized in Tables 2 and 3 below.

**MD - Array**

- **MD(1) = N**  
  The dimension of the matrix; to be supplied by the user.

- **MD(2) = MX**  
  The lengths of the arrays RY, CY and A. If the decomposition of the matrix requires more internal storage, that is, if $m_1 > MX$, then the error indicator MD(4) is set to one and the process is terminated with a return to the user's main program.

- **MD(3)**  
  If this indicator is zero, the DECO1 program does not produce a decomposition record; for any nonzero value of MD(3) such a record is generated.
Error indicator set as follows:

= 0 : no error.
= 1 : storage overflow; MX is too small for decomposition.
= 2 : on the basis of the sparsity structure (independent of the values of the elements) the matrix is singular.
= 3 : the matrix is declared numerically singular.

In the nonsymmetric case equal to \( M_0 + N \) where \( M_0 \) is the number of nonzero elements in the matrix; in the symmetric case equal to \( M_0 \), the number of nonzero elements on the diagonal and in the upper (or lower) triangle of the matrix.

The length of the actually utilized portion of the arrays \( R_Y, C_Y \) or \( A \), equal to \( M_1 + N \) or \( M_1 \) in the nonsymmetric or symmetric case, respectively.

In the nonsymmetric case equal to \( M_2 + N \) where \( M_2 \) is the number of nonzero elements in the decomposed matrix; in the symmetric case equal to the nonzero elements on the diagonal and in the lower triangle after the decomposition.

Length of the decomposition record.

A tolerance EPS supplied by the user. If a pivot value in magnitude is less than EPS the matrix is considered to be numerically singular and the decomposition is terminated.

Initial input by the user containing the pivot selection parameter \( \mu \).

Largest coefficient value in magnitude in the original matrix. Initialized by the INT routines and updated by the BLD routines.
FD(4)  Largest coefficient value in magnitude encountered during
decomposition, calculated by the DEC routines.

FD(5)  Natural logarithm of the absolute value of the determinant
calculated by the DEC routines.

FD(6)  The sign of the determinant as +1.0 or -1.0 calculated by
the DEC routines.

FD(7)  The natural logarithm of the product of the $L_2$ norms
of the row vectors of the original matrix calculated by
the DEC routines.

**Data Arrays**

RY(MX), CY(MX)  Integer arrays for the sparsity structure. Their
length MX is specified by MD(2).

A(MX)  Real array for the values of the nonzero matrix
elements.

AN(N)  Real array of length N (see MD(1)) used to collect
row-vector norms of the matrix.

ND(N)  For symmetric A.

NDR(N), NDCC(N)  For nonsymmetric A. Integer arrays of length N
containing the number of nonzero elements by row
(or column).

IE(N), IH(N)  For any matrix.

IP(N)  For symmetric matrices.

IPR(N), IPC(N)  For nonsymmetric matrices. Temporary arrays of

NG1(N), NG2(N)  length N.
Except for the last seven temporary arrays all data arrays are initialized in the appropriate INT routines. All integer arrays are used only for storing nonnegative integers. Thus for particular computers these arrays could be packed together.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Length</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINT0</td>
<td>-</td>
<td>-</td>
<td>US - S S - S - - -</td>
</tr>
<tr>
<td>SINT01</td>
<td>-</td>
<td>-</td>
<td>US US S S S S - - - -</td>
</tr>
<tr>
<td>SINT1</td>
<td>-</td>
<td>-</td>
<td>UPS US - - - S - - - -</td>
</tr>
<tr>
<td>SBLD0</td>
<td>U</td>
<td>U</td>
<td>PS - PS PS - - PS - - - -</td>
</tr>
<tr>
<td>SBLD01</td>
<td>U</td>
<td>U</td>
<td>U PS PS PS PS PS PS PS - - - -</td>
</tr>
<tr>
<td>SBLD1</td>
<td>U</td>
<td>U</td>
<td>U PS PS - - PS PS - - - -</td>
</tr>
<tr>
<td>SDEC0</td>
<td>-</td>
<td>-</td>
<td>PS - PT PT - - PT T T T -</td>
</tr>
<tr>
<td>SDEC01</td>
<td>-</td>
<td>-</td>
<td>PS PS PT PT PT PT PT T T T -</td>
</tr>
<tr>
<td>SDEC1</td>
<td>-</td>
<td>-</td>
<td>PS PS - - PT PT - - T - -</td>
</tr>
<tr>
<td>SSLV</td>
<td>-</td>
<td>-</td>
<td>PS - - - - - - - - U</td>
</tr>
</tbody>
</table>

Table 2
Usage of Arguments in the Symmetric Case
(for legend see Table 3)
### Table 3

**Usage of Arguments in the Nonsymmetric Case**

| Program | Length | Type | Name | I | J | V | M | D | F | D | R | Y | C | Y | A | AN | NDR | NDC | IPR | IPC | IE | IH | NG1 | NG2 | X | Y |
|---------|--------|------|------|---|---|---|---|---|---|---|---|---|---|---|---|---|-----|-----|-----|-----|-----|-----|---|---|
| NBLD01  | U U U | PS   | PS   | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS   | PS   | PS   | PS   | PS   | PS   | PS | PS |
| NBLD1   | U U U | PS   | PS   | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS | PS   | PS   | PS   | PS   | PS   | PS   | PS | PS |
| NDECO1  | -      | PS   | PS   | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT   | PT   | PT   | PT   | PT   | PT   | PT | PT |
| (PVT01) |       |      |      |    |    |    |    |    |    |    |    |    |    |    |    |    |      |      |      |      |      |      |    |    |
| NDEC1   | -      | PS   | PS   | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT | PT   | PT   | PT   | PT   | PT   | PT   | PT | PT |
| NSLV    | -      | PS   | -    | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -    | -    | -    | -    | -    | -    | -  | -  |

**Legend:**

- Argument type: I - integer
- R - real
- Entries:
  - U - user-supplied data
  - S - upon exit, the argument contains data to be saved for other reasons
  - P - contains data generated by previously called program
  - T - temporary storage
  - U - output result
5. Some Experimental Results

Two groups of computational experiments were conducted on the Univac 1108 of the University of Maryland, Computer Science Center. They correspond to the computational experiments reported in [1]. Since the basic decomposition procedure is the same as in [1], the overall elapsed time for execution of the decomposition programs \( T_{LU} \) and the number of elements after decomposition \( M_2 \) are essentially the same as reported there.

The new results presented below concern the maximal in-core storage requirement \( M_1 \), the elapsed time for execution of the decomposition routines \( T_1 \) using a previously generated decomposition record, and the elapsed time for execution of the backsstitution routines \( T_{SO} \) when the decomposed matrices are residing on auxiliary storage. These times are given below relative to the elapsed time \( T_{LU} \) for the execution of the decomposition programs. It should be pointed out that for \( N \) larger than 100 there is less than a three percent difference between the elapsed time for the generation of a decomposition record (SDECO) and that for a regular decomposition with or without retaining the record (SDECO1, NDECO1).

The first group of experiments involved nonsymmetric matrix decompositions. As in [1], a special program was used to generate permuted diagonally dominant random sparse matrices \( B = (b_{ij}) \). Given the dimension \( N \) of \( B \) and a number \( M_0 \) of nonzero elements, the program randomly generates \( M_0 - N \) distinct index pairs \((i,j), i \neq j, 1 \leq i,j \leq N\), and the corresponding matrix elements \( b_{ij} \). Then each diagonal element is obtained by adding a random positive number to the sum of the moduli of the off-diagonal elements in its row. Finally, the rows and columns are independently and randomly
permuted. Table 4 contains the results obtained when the decomposition programs are applied to these random matrices. The pivot selection parameter $\mu = 0.125$ was used.

Table 4

<table>
<thead>
<tr>
<th>N</th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$T_1/T_{LU}$</th>
<th>$T_{SO}/T_{LU}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>200</td>
<td>200</td>
<td>375</td>
<td>.255</td>
<td>.068</td>
</tr>
<tr>
<td>50</td>
<td>300</td>
<td>300</td>
<td>632</td>
<td>.218</td>
<td>.041</td>
</tr>
<tr>
<td>100</td>
<td>400</td>
<td>400</td>
<td>811</td>
<td>.198</td>
<td>.044</td>
</tr>
<tr>
<td>100</td>
<td>600</td>
<td>1,114</td>
<td>2,026</td>
<td>.140</td>
<td>.016</td>
</tr>
<tr>
<td>200</td>
<td>800</td>
<td>913</td>
<td>2,312</td>
<td>.138</td>
<td>.020</td>
</tr>
<tr>
<td>200</td>
<td>1,200</td>
<td>3,991</td>
<td>6,439</td>
<td>.074</td>
<td>.005</td>
</tr>
<tr>
<td>300</td>
<td>1,200</td>
<td>1,924</td>
<td>4,224</td>
<td>.121</td>
<td>.012</td>
</tr>
</tbody>
</table>

The second group of experiments involved the decomposition of symmetric matrices obtained by the discretizing of Dirichlet's problem for Laplace's equation on the unit square with the five-point and nine-point formulas. In all cases a uniform mesh was used. The coefficients of the resulting matrices are well known and are not repeated here. Table 5 contains the results obtained with the symmetric decomposition programs.
Table 5
Symmetric Matrix Decomposition

<table>
<thead>
<tr>
<th>5-point Formula</th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$T_1/T_{LU}$</th>
<th>$T_{SO}/T_{LU}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 81$</td>
<td>225</td>
<td>225</td>
<td>469</td>
<td>.47</td>
<td>.18</td>
</tr>
<tr>
<td>$N = 289$</td>
<td>833</td>
<td>883</td>
<td>2,413</td>
<td>.22</td>
<td>.06</td>
</tr>
<tr>
<td>$N = 484$</td>
<td>1,408</td>
<td>1,699</td>
<td>4,733</td>
<td>.17</td>
<td>.04</td>
</tr>
<tr>
<td>$N = 625$</td>
<td>1,825</td>
<td>2,328</td>
<td>6,566</td>
<td>.16</td>
<td>.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9-point Formula</th>
<th>$M_0$</th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$T_1/T_{LU}$</th>
<th>$T_{SO}/T_{LU}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 81$</td>
<td>353</td>
<td>353</td>
<td>715</td>
<td>.35</td>
<td>.10</td>
</tr>
<tr>
<td>$N = 289$</td>
<td>1,345</td>
<td>1,673</td>
<td>4,296</td>
<td>.15</td>
<td>.03</td>
</tr>
<tr>
<td>$N = 484$</td>
<td>2,290</td>
<td>2,928</td>
<td>8,054</td>
<td>.13</td>
<td>.02</td>
</tr>
<tr>
<td>$N = 625$</td>
<td>2,977</td>
<td>4,047</td>
<td>11,800</td>
<td>.10</td>
<td>.01</td>
</tr>
</tbody>
</table>
6. Program Listings

6.1 Main Package:

SUBROUTINE SINT0(MD,RY,CY,ND)
DIMENSION MD(1),RY(1),CY(1),ND(1)
INTEGER RY,CY
C *********************************************************************************************
C * INITIALIZE SYMMETRIC STRUCTURE ARRAYS *
C *********************************************************************************************
N = MD(1)
MD(5) = N
DO 10 I=1,N
RY(I) = I
CY(I) = I
ND(I) = 1
RETURN
10
END

SUBROUTINE SINT1(MD,FD,RY,CY,A,AN,ND)
DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),AN(1),ND(1)
INTEGER RY,CY
C *********************************************************************************************
C * INITIALIZE SYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C *********************************************************************************************
N = MD(1)
MD(5) = N
FD(3) = 0.
DO 10 I=1,N
AN(I) = 0.
RY(I) = I
CY(I) = I
ND(I) = 1
RETURN
10
END

SUBROUTINE SINT11(MD,FD,AN)
DIMENSION MD(1),FD(1),AN(1)
C *********************************************************************************************
C * INITIALIZE SYMMETRIC COEFFICIENT ARRAY *
C *********************************************************************************************
N = MD(1)
MD(5) = N
FD(3) = 0.
DO 10 I=1,N
AN(I) = 0.
RETURN
10
END
SUBROUTINE NINTU1(MD,FD,RY,CY,AN,NDR,NUC)
DIMENSION MD(1),FD(1),RY(1),CY(1),NDR(1),NUC(1),AN(1)
INTEGER RY,CY
C ***********************************************************************
C * INITIALIZE NONSYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ***********************************************************************
C
N = MD(1)
MU(5) = N
FD(3) = 0.
DO 10 I=1,N
AN(I) = 0*
RY(1) = 1
CY(1) = 1
NDR(1) = 0
NU10 NDC(1) = 0
RETURN
END

SUBROUTINE NINT1(MD,FD,AN)
DIMENSION MD(1),FD(1),AN(1)
INTEGER RY,CY
C ***********************************************************************
C * INITIALIZE NONSYMMETRIC COEFFICIENT ARRAY *
C ***********************************************************************
C
N = MD(1)
MU(5) = N
FD(3) = 0.
DO 10 I=1,N
AN(I) = 0*
RETURN
END

SUBROUTINE SBLDU(I,J,MD,RY,CY,ND)
DIMENSION MD(1),RY(1),CY(1),ND(1)
INTEGER RY,CY
C ***********************************************************************
C * BUILDU SYMMETRIC STRUCTURE ARRAY *
C ***********************************************************************
C
IF (I.EQ.J) RETURN
MD(5) = MD(5)+1
MU = MD(5)
ND(I) = ND(I)+1
ND(J) = ND(J)+1
IF (I.GT.J) GO TO 1U
RY(MU) = RY(I)
CY(MU) = CY(J)
RY(I) = MU
CY(J) = MU
RETURN
1U RY(MU) = RY(J)
CY(MU) = CY(I)
RY(J) = MU
CY(I) = MU
RETURN
END
SUBROUTINE SBLD01(I,J,V,MD,FD,RY,CY,A,AN,ND)
DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),AN(1),ND(1)
INTEGER RY,CY
C ***********************************************************************
C * BUILD SYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ***********************************************************************
C
S = V**2
FD(3) = AMAX1(FD(3),ABS(V))
AN(I) = AN(I)+S
IF (I.EQ.J) GO TO 20
MD(5) = MD(5)+1
MU = MD(5)
AN(J) = AN(J)+S
ND(I) = ND(I)+1
IF (I.GT.J) GO TO 10
HY(MU) = HY(I)
CY(MU) = CY(J)
KY(I) = MU
CY(J) = MU
RETURN
10 HY(MU) = HY(J)
CY(MU) = CY(I)
KY(J) = MU
CY(I) = MU
RETURN
20 A(I) = V
RETURN
END
SUBROUTINE NBLD01(I,J,V,MD,FD,RY,CY,A,AN,NDR,NDC)
DIMENSION MD(1),FD(1),RY(1),CY(1),A(1),NDR(1),NDC(1),AN(1)
INTEGER RY,CY
C ***************************************************
C * BUILD NONSYMMETRIC STRUCTURE AND COEFFICIENT ARRAYS *
C ***************************************************
MD(5) = MD(5)+1
MD = MD(5)
RY(J) = RY(I)
CY(J) = CY(I)
MU = MD(J)
ND(J) = NDR(J)+1
NDC(J) = NDC(J)+1
A(MU) = V
AN(I) = AN(I)+V**2
FD(3) = AMAX1(FD(3),ABS(V))
RETURN
END

SUBROUTINE SDECO(MD,RY,CY,NR,IP,IE,IH)
DIMENSION MD(1),RY(1),CY(1),NR(1),IP(1),IE(1),IH(1)
INTEGER RY,CY
C ***************************************************
C * GENERATE SYMMETRIC DECOMPOSITION RECORD *
C ***************************************************
MM = 0
N = MD(1)
MD(4) = 0
MD(6) = MD(5)
MD(7) = MD(5)
MD(8) = 0
C INITIALIZE AUXILIARY FILE FOR WRITING
CALL UWI
DO 10 J=1,N
10 IP(J) = 1
RETURN
END
C LOOP ON PIVOTING
DO 250 I=1,N
K = 0
IF (I.EQ.N) GO TO 30
C SELECT PIVOT BY MINIMAL DEGREE
NDX = N+1
DO 20 J=1,N
IX = IP(I)
IF (ND(IX).GE.NDX) GO TO 20
NDX = ND(IX)
IY = J
CONTINUE
J = IP(IY)
IP(IY) = IP(I)
IP(I) = J
C COLLECT THE ROW AND COLUMN OF THE PIVOT
ALSO DELETE THEM FROM THE STORAGE
IX = IP(I)
IF (I.EQ.IY) GO TO 110
IY1 = 0
IY = IX
IY = RY(IY)
IF (IY.EQ.IX) GO TO 70
I2 = IY
I2 = CY(I2)
IF (I2.GT.N) GO TO 60
K = K+1
IE(K) = IY
IH(K) = I2
ND(IZ) = ND(IZ)-1
IF (CY(IZ).NE.IY) GO TO 50
CY(I2) = CY(IY)
CY(IY) = MM
MM = IY
GO TO 40
IY = CY(IY)
IF (IY.EQ.IX) GO TO 100
I2 = IY
IY1 = IY
I2 = RY(I2)
IF (I2.GT.N) GO TO 90
K = K+1
IE(K) = IY
IH(K) = I2
ND(IZ) = ND(IZ)-1
IF (RY(I2).NE.IY) GO TO 80
RY(I2) = RY(IY)
GO TO 70
IF (IY1.EQ.0) GO TO 110
CY(IY1) = MM
MM = CY(IX)
C MODIFICATION OF THE ROW ELEMENTS
WRITE OUT PIVOT
CALL Dw(IX)
CALL Dw(K)
MD(I) = MD(I)+1+K**2
IF (K.EQ.0) GO TO 250
DO 115 J=1,K
CALL Dw(IE(J))
115 CALL Dw(IH(J))
IF (K.EQ.1) GO TO 250
K1 = K-1
C LOOP FOR THE CROSS-POINT ELEMENTS

C 240 J=1,K1
J1 = J+1
J2 = IH(J)
DO 230 JU=J1,K
J2 = IH(JU)
11 = MIN(I1,J2)
I2 = MAXU11,J2)
L1 = RY(U1)
L2 = CY(I2)
120 IF ((L1,EG,11),OR,(L2,EQ,12)) GO TO 140
105 L1 = CY(L1)
120 IF (L1,GT,L2) GO TO 220
105 L2 = CY(L2)
109 GO TO 120
110 L1 = CY(L1)
120 IF (L1,LT,L2) GO TO 210
105 L2 = CY(L2)
110 IF ((L1,LE,L2)) GO TO 170
113 USE AVAILABLE STORAGE
114 MD(6) = MD(6)+1
115 IF (MD(6),LE,MD(2)) GO TO 160
116 MD(4) = 0
117 RETURN
118 L1 = MD(6)
119 RY(L1) = RY(11)
120 CY(L1) = CY(12)
121 L1 = CY(L1)
122 GO TO 220
124 USE AVAILABLE STORAGE
125 L1 = MM
126 MM = CY(MM)
127 L1 = L1
128 L2 = L2
129 IF (RY(L3),LT,L1) GO TO 190
130 L3 = RY(L3)
131 GO TO 180
132 RY(L1) = RY(L3)
133 RY(L3) = L1
134 IF (L2,GT,L1) GO TO 200
135 L2 = CY(L2)
136 GO TO 200
137 CY(L1) = CY(L2)
138 CY(L2) = L1
139 WRITE OUT CROSS-POINT ELEMENT
140 CALL Dw(L1)
141 CONTINUE
142 CONTINUE
143 END OF MODIFICATION LOOP
144 CONTINUE
145 CONTINUE
146 END OF PIVOTING LOOP
147 CALL DwE
148 RETURN
150 C ENU
* DECOMPOSE SYMMETRIC MATRIX AND OPTIONALLY *

* GENERATE DECOMPOSITION RECORD *

** SUBROUTINE SDECU(MD,FD,RY,CY,A,AN,ND,IP,IE,IH) **
** DIMENSION MD(1),FD(1),RY(1),CY(1),ND(1),IP(1),IE(1),IH(1) **

** INTEGER RY,CY **

** ****************************************************************** **
** ******** DECOMPOSE SYMMETRIC MATRIX AND OPTIONALLY ******** **
** ******** GENERATE DECOMPOSITION RECORD ******** **
** ****************************************************************** **

N = MD(1)
MD(4) = 0
FD(5) = 0.
FD(6) = 1.0
FD(7) = 0.0
MM = 0
MD(6) = MD(5)
MD(7) = MD(5)
MD(8) = 0

** INITIALIZE AUXILIARY FILE FOR WRITING **
C IF (MD(3),NE,0) CALL DWL

DO 10 I=1,N
FD(7) = FD(7)+ALOG(AN(I))
IP(I) = I
FD(7) = 0.5*FD(7)
CALL SW1

** LOOP ON PIVOTING **

DO 20 I=1,N
K = 0
IF (I.EQ,N) GO TO 30

SELECT PIVOT BY MINIMAL DEGREE

NX = N+1
AX = 0.
DO 15 J=1,N
IX = IP(J)
15
AX = AMAX1(AX,ABS(A(IX)))
DO 20 J=1,N
IX = IP(J)
20
IF (ABS(A(IX)),LT,AX) GO TO 20
IF (MD(IX),GE,NDX) GO TO 20
NX = MD(IX)

CONTINUE

IF (I.EQ.IY) GO TO 30
J = IP(IY)
IP(IY) = IP(I)
IP(I) = J

COLLECT THE ROW AND COLUMN OF THE PIVOT

ALSO DELETE THEM FROM THE STORAGE

IX = IP(I)
S = A(IX)
IF (ABS(S),LT,FD(1)) GO TO 300
IF (I.EQ,N) GO TO 110
LY = 0
LY = IX
UB3 4U  IY = RY(IY)
064 5U  IF (IY.EQ.IX) GO TO 70
065 5U  IZ = IY
066 5U  IZ = CY(IZ)
067 5U  IF (IZ.GT.N) GO TO 60
068 6U  K = K+1
069 6U  IE(K) = IY
070 6U  IH(K) = IZ
071 6U  AN(K) = A(IY)
072 6U  A(IY) = 0.
073 6U  ND(IZ) = ND(IZ)-1
074 6U  IF (CY(IZ),NE,IY) GO TO 50
075 6U  CY(IZ) = CY(IY)
076 6U  CY(IY) = MM
077 6U  MM = IY
078 7U  GO TO 4U
079 7U  IY = CY(IY)
080 8U  IF (IY.EQ.IX) GO TO 100
081 8U  IZ = IY
082 8U  IY = IY
083 8U  IZ = RY(IZ)
084 8U  IF (IZ.GT.N) GO TO 90
085 9U  K = K+1
086 9U  IE(K) = IY
087 9U  IH(K) = IZ
088 9U  AN(K) = A(IY)
089 9U  A(IY) = 0.
090 9U  ND(IZ) = ND(IZ)-1
091 9U  IF (HY(IZ),NE,IY) GO TO 80
092 9U  HY(IZ) = HY(IY)
093 9U  GO TO 7U
094 10U  IF (IY1.EQ.0) GO TO 110
095 10U  CY(IY1) = MM
096 10U  MM = CY(IY)
097 C	C	MODIFICATION OF THE ROW ELEMENTS
098 C	C	C	WHILE OUT PIVOT
099 C	C	110 IF (MD(3).NE.0) CALL DW(IX)
100 C	C	111 IF (MD(3).NE.0) CALL DW(K)
101 C	C	112 FD(U) = AMAX1(FD(U),ABS(A))
102 C	C	113 FD(U) = FD(U)+ALUG(ABS(A))
103 C	C	114 IF (S.LT.0.) FD(U) = -FD(U)
104 C	C	115 CALL SVW(-IX,5)
105 C	C	116 MD(U) = MD(U)+1+K**2
106 C	C	117 IF (K.EQ.0) GO TO 250
107 C	C	118 GO 115 J=1,K
108 C	C	119 AN(U) = AN(U)/S
110 C	C	120 CALL SVW(IM,AN(U)),AN(U))
111 C	C	121 FD(U) = AMAX1(FD(U),ABS(A))
112 C	C	122 IF (MD(3).NE.0) CALL DW(IE(J))
113 C	C	123 IF (MD(3).NE.0) CALL DW(IH(J))
114 C	C	124 K1 = K-1
115 C	C	125 IF (J.EQ.K) GO TO 240
116 C	C	C	C	C	C	LOOP FOR THE CROSS-POINT ELEMENTS
117 C	C	U0 120 J=1,K
118 C	C	J1 = J+1
119 C	C	I2 = I1H(J)
120 C	C	2 = AN(U)
121 C	C	A(I2) = A(I2)-S*Z**2
122 C	C	FD(U) = AMAX1(FD(U),ABS(A(I2))
123 C	C	IF (J.EQ.K) GO TO 240
DO 230 JJ=J1,K
J2 = IM(JJ)
I1 = MIN0(I2,J2)
I2 = MAX0(I2,J2)
L1 = RY(I1)
L2 = CY(I2)
IF (L1.EQ.I1) OR (L2.EQ.I2) GO TO 140
IF (L1.EQ.L2) GO TO 220
IF (L1.GT.L2) GO TO 130
L2 = CY(L2)
L1 = RY(L1)
GO TO 120
130 L1 = RY(L1)
GO TO 120
C INSERTION OF A NEW NON-ZERO ELEMENT
140 ND(11) = ND(11)+1
ND(12) = ND(12)+1
MD(7) = MD(7)+1
IF (MD(7).NE.0) GO TO 170
C USE NEW STORAGE
144 MD(6) = MD(6)+1
IF (MD(6).GT.MD(2)) GO TO 310
160 L1 = MD(6)
A(L1) = 0,
RY(L1) = RY(I1)
CY(L1) = CY(I2)
RY(L1) = LI
CY(L1) = LI
GO TO 220
C USE AVAILABLE STORAGE
154 L1 = MM
MM = CY(MM)
L2 = I2
180 IF (RY(L3).LT.L1) GO TO 190
L4 = RY(L3)
GO TO 180
190 RY(L1) = RY(L3)
RY(L3) = L1
IF (CY(L2).LT.L1) GO TO 210
L2 = CY(L2)
GO TO 200
200 CY(L1) = CY(L2)
CY(L2) = L1
C WRITE OUT CROSS-POINT ELEMENT
220 IF (MD(3).NE.0) CALL DWE(L1)
A(L1) = A(L1)+AN(J)*AN(JT)*S
230 FD(4) = AMAX1(FD(4),ABS(A(L1)))
240 CONTINUE
C END OF MODIFICATION LOOP
250 CALL SVW(-IX,S)
C END OF PIVOTING LOOP
270 CALL SVWE
IF (MD(3).NE.0) CALL DWE
RETURN
C SINGULAR MATRIX
300 MD(4) = 1
RETURN
310 MD(4) = 3
RETURN
C END
SUBROUTINE SUEC1(MD, FD, A, AN, IE)

C ***********************************************************************
C * DECOMPOSE SYMMETRIC MATRIX USING GENERATED RECORD *
C ***********************************************************************

C
N = MD(1)
MD(4) = 0
FD(5) = 0.
FD(6) = 1.
FD(7) = 0.
FD(4) = 0.
DO 10 I=1,N
10   FD(7) = FD(7)+ALOG(AN(I))
FD(7) = 0.5*FD(7)
CLEAN STORAGE TO BE FILLED
IF (MD(6).LE.MD(5)) GO TO 30
MM = MD(5)+1
M1 = MD(6)
DO 20 I=MM,M1
20   A(I) = 0.
C INITIALIZE FOR READ-IN AND WRITE-OUT
CALL DR1
CALL SW1
C LOOP ON THE PIVOTS
C
DO 90 I=1,N
C
CALL DR1(I)
CALL DR1(K)
S = A(I)
IF (ABS(S).LT.FD(1)) GO TO 100
FD(4) = AMAX1(FD(4),ABS(S))
FD(5) = FD(5)+ALOG(ABS(S))
IF (S.LT.0.) FD(6) = -FD(6)
IF (K.EQ.0) GO TO 70
C
COLLECT THE ROW OF THE PIVOT
C
DO 40 J=1,K
CALL DR1(L1)
CALL DR1(IE(J))
AN(J) = A(L1)
A(L1) = 0.
FD(4) = AMAX1(FD(4),ABS(AN(J)))
C
MODIFICATION OF THE ROW ELEMENTS
C
DO 60 J=1,K
I2 = IE(J)
Z = AN(J)
AN(J) = AN(J)/S
A(I2) = A(I2)-Z*AN(J)
FD(4) = AMAX1(FD(4),ABS(AN(J)))
FD(4) = AMAX1(FD(4),ABS(A(I2)))
IF (J.EQ.K) GO TO 60
J1 = J+1
40    CONTINUE
C
MODIFICATION OF THE CROSS-POINT ELEMENTS
C
DO 50 JJ=J+1,K
C
CALL DR1(L1)
A(L1) = A(L1)-AN(J)*AN(JJ)
FD(4) = AMAX1(FD(4),ABS(A(L1)))
50    CONTINUE
C WRITE OUT PIVOT AND ITS ROW
70    CALL SVW(-IX,S)
      IF (K.EQ.0) GO TO 90
50    DO 80 J=1,K
     80    CALL SVW(IE(J), AN(J))
90    CALL SVW(-IX,S)
C END OF PIVOTING LOOP
C
C SINGULAR MATRIX
C
100    MD(4) = 0
C
C END

SUBROUTINE NDUE01(MD, FD, CY, A, AN, NDR, NDC, IPR, IPC, IE, IH, NG1, NG2)
DIMENSION MD(1), FD(1), CY(1), A(1), AN(1), NDR(1), NDC(1)
INTEGER H, CY

*** * DECOMPOSE NONSYMMETRIC MATRIX AND OPTIONALLY *
*** * GENERATE DECOMPOSITION RECORD *
*** * ***************************************************************************

MM = 0
N = MD(1)

MD(4) = 0
MU(6) = MD(5)
MD(7) = MD(5)
MD(8) = 0

IF (MD(3).NE.0) CALL DW1
FD(5) = 0.
FD(6) = 1.
FD(7) = 0.
FD(4) = 0.
DO 10 I=1,N
     10 FD(7) = FD(7)+ALOG(AN(I))
IPR(I) = I
IPC(I) = I

DO 125 I=1,N

125 FD(7) = 0.5*FD(7)
CALL NVW1
CALL NVW2(-1,-1)

LOOP ON PIVOTING
DO 290 I=1,N

PIVOT SELECTION BY SEPARATE PIVOTING ROUTINE
CALL PVTO1(1,N, IX, KR, KC, FD(2), CY, A, IPR, IPC, NDR, NDC, IE, IH)
IF (KR.EQ.0) GO TO 310

CHECK PIVOT VALUE AND UPDATE DETERMINANT
S = A(IX)

IF (ABS(S).LE.FD(1)) GO TO 300
FD(5) = FD(5)+ALOG(ABS(S))
K = (KR+KC)/2
K = KR+KC/2*K
IF (K.NE.0) FD(6) = -FD(6)
FD(4) = AMAX1(FD(4), ABS(S))
C COLLECT THE ROW AND COLUMN OF THE PIVOT
C ALSO FREE THEIR STORAGE LOCATIONS
C THE ROW
K1 = 0
J = KR
20 IF (J, LE, N) GO TO 40
IF (J, EQ, IX) GO TO 20
K1 = K1 + 1
IE(K1) = J
40 AN(K1) = A(J)
J1 = J
30 IF (J1, LE, N) NG1(K1) = J1
IF (J1, LE, N) NDC(J1) = NDC(J1) - 1
IF (CY(J1), NE, J) GO TO 30
CY(J1) = CY(J)
50 CY(J) = MM
GO TO 20
C THE COLUMN
K2 = 0
J = KC
K3 = 0
J = CY(J)
50 IF (J, LE, N) GO TO 70
K3 = J
70 IF (J, EQ, IX) GO TO 50
K2 = K2 + 1
IF(K2) = J
A(K2) = A(J)/S
FD(4) = MAX1(FD(4), ABS(A(K2)))
70 J1 = J
60 IF (J1, LE, N) NG2(K2) = J1
IF (J1, LE, N) NDR(J1) = NDR(J1) - 1
IF (RY(J1), NE, J) GO TO 60
RY(J1) = RY(J)
60 GO TO 50
70 CY(K3) = MM
MM = CY(KC)
80 C WRITE OUT THE PIVOT, ITS ROW AND COLUMN
C AS PART OF THE DECOMPOSITION RECORD
90 C
91 IF (M, EQ, 0) GO TO 105
92 CALL Dw(IX)
93 CALL Dw(KC)
95 CALL Dw(KR)
96 CALL Dw(K2)
97 MW(j) = MW(j) + (K1 + 1) * (K2 + 1)
98 IF (K2, EQ, 0) GO TO 90
99 DO 80 J = 1, K2
90 CALL Dw(jh(j))
100 DO 80 J = 1, K2
101 CALL Dw(NG2(j))
102 CALL Dw(K1)
103 IF (K1, EQ, 0) GO TO 105
104 DO 100 J = 1, K1
105 CALL Dw(IE(j))
106 CALL Dw(NG1(j))
107 CALL Dw(NG1(j))
108 IF ((K1, EQ, 0) OR, (K2, EQ, 0)) GO TO 230
109 C LOOP TO MODIFY THE INTERSECTING ELEMENTS BETWEEN THE ROW
110 C AND COLUMN
DO 220 J=1,K2

   IY = NG2(J)
   DO 220 K=1,K1

   K3 = HY(IY)
   JY = NG1(K)
   K4 = CY(JY)

C SEARCH FOR ELEMENT IY,JY
110  IF (K3.EQ.K4) GO TO 210

120  IF (K3.LT.K4) GO TO 128

   K3 = HY(K3)
   IF (K3.LE.N) GO TO 130
   GO TO 110

128  K4 = CY(K4)

125  IF (K4.GT.N) GO TO 110

130  C IT DOES NOT EXIST

135  NDR(IY) = NDR(IY)+1

138  NDC(JY) = NDC(JY)+1

140  MD(7) = MD(7)+1

145  IF (MM.NE.0) GO TO 160

131  C GET NEW LOCATION FOR THE NEW ELEMENT

132  MD(6) = MD(6)+1

135  IF (MD(6).LE.MD(2)) GO TO 150

140  MD(4) = 1

145  RETURN

150  K3 = MD(6)

153  RY(K3) = RY(IY)

156  CY(K3) = CY(JY)

159  RY(IY) = K3

162  CY(JY) = K3

165  A(K3) = 0.

GO TO 210

143  C OLD LOCATION AVAILABLE FOR THE NEW ELEMENT

146  K3 = MM

149  A(K3) = 0.

152  MM = CY(MM)

155  K4 = IY

160  IF (HY(K4).LT.K3) GO TO 180

163  K4 = HY(K4)

166  GO TO 170

160  RY(K3) = RY(K4)

169  GO TO 180

163  RY(K4) = K3

176  IF (CY(JY).LT.K3) GO TO 200

179  CY(JY) = K3

182  CY(JY) = K3

185  C MODIFY ELEMENT

186  IF (MD(3).NE.0) CALL DW(K3)

190  A(K3) = A(K3)-AN(K)*A(J)

193  FD(4) = AMAX1(FD(4),ABS(A(K3)))

196  CONTINUE

199  C END OF MODIFICATION LOOP

202  C WRITE OUT PIVOT ROW AND COLUMN

205  CALL NVWF(-KC,-KR)

209  CALL NVWF(NG2(J),A(J))

212  CALL NVWB(NG1(J),AN(J))

216  CALL NVWB(-KC,S)

220  CONTINUE

223  C END OF PIVOTING LOOP

226  C END FILES
**C** IF (MD(3).NE.0) CALL DWE
181        CALL NVWE
182        RETURN
183          **C** SINGULAR MATRIX
184          **C** 300 MD(4) = 3
185          RETURN
186          310 MD(4) = 2
187          RETURN
188          **C** END

**C** SUBROUTINE PVT01(I,N,IX,KR,KC,F,Ry,CY,A,IPR,IPC,NDR,NDC,IE,IH)
002        DIMENSION Ry(1),Cy(1),IPR(1),IPC(1),NDR(1),NDC(1),IE(1),IH(1)
003        INTEGER Ry,CY
005          **C** ********************************************************************
006          **C** * MINIMAL DEGREE PIVOTING FOR NON-SYMMETRIC MATRIX *
007          **C** ********************************************************************
008          **C** THE ROUTINE SELCTS PIVOT BY MINIMAL DEGREE, IT IS
009          **C** USED BY THE ROUTINE DECO1.
010          **C**
012          IF (I.NE.N) GO TO 30
014          KR = IPR(I)
015          IX = Ry(KR)
016          IF (IX.NE.KR) RETURN
017          30    N1 = N+1-I
018          RETURN
019          30    N1 = N+1-I
020          30    N1 = N+1-I
021          **C** SORT AVAILABLE ROWS BY DEGREE
022            **C**
023            DO 40 J=1,N1
024                40    IE(J) = 0
025            DO 50 J=1,N1
026                50    K1 = IPR(J)
027                    50    IH(J) = K1
028                50    K2 = NDR(K1)
029                    50    IF (K2.LE.0) GO TO 10
030                50    IE(K2) = IE(K2)+1
031            DO 60 J=2,N1
032                60    IE(J) = IE(J)+IE(J-1)
033            DO 70 J=1,N1
034                70    K1 = IH(J)
035                70    K2 = NDR(K1)
036                70    IE(K2) = IE(K2)+1
037                70    IE(K2) = IE(K2)+1
038                70    IE(K2) = IE(K2)+1
039                **C** SORT AVAILABLE COLUMNS BY DEGREE
040            **C**
042            DO 80 J=1,N1
043                80    IE(J) = 0
044            DO 90 J=1,N1
045                90    K1 = IPC(J)
046                    90    IH(J) = K1
047                90    K2 = NDC(K1)
048                90    IF (K2.LE.0) GO TO 10
049                90    IE(K2) = IE(K2)+1
DO 100 J=2,N1

IE(J) = IE(J) + IE(J-1)
DO 110 J=1,N
K1 = IH(J)
K2 = NDJ(K1)
K3 = IE(K2)+I-1
IE(K2) = IE(K2)-1

110 IPC(K3) = K1

C INITIALIZE FOR MINIMAL DEGREE SEARCH

IX = 0
IDX = N**2
JC = 1
JRP = IPC(JC)
JCP = IPC(JC)

C TEST FOR TERMINATION OF SEARCH

120 NDX = (NDJ(JRP)-1)*(NDJ(JCP)-1)

1.1 IF (NDX .LT. IX) GO TO 240
1.2 IF (NDJ(JCP), GT, NDR(JRP)) GO TO 180

C SEARCH IN THE COLUMN

J = JCP
AM = 0.

130 J = CY(J)
140 IF (J, EQ, JCP) GO TO 140
150 AM = AMAX1(AM, ABS(A(J)))
160 GO TO 130
170 AM = F*AM
180 IF (J, EQ, JCP) GO TO 170
190 IF (ABS(A(J)) , LT, AM) GO TO 150
200 K1 = J
210 K1 = CY(K1)
220 IF (K1, GT, N) GO TO 220
230 IF (K2, EQ, 1) GO TO 150
240 IX = J
250 KC = JCP
260 IDX = K2
270 GO TO 150
280 JC = IX + 1
290 IF (JC, GT, N) GO TO 240
300 JCP = IPC(JC)
310 GO TO 130

C SEARCH IN THE ROW

320 J = JRP
330 AM = 0.
340 IF (J, EQ, JRP) GO TO 200
350 AM = AMAX1(AM, ABS(A(J)))
360 GO TO 200
370 AM = F*AM
380 IF (J, EQ, JRP) GO TO 200
390 IF (ABS(A(J)) , LT, AM) GO TO 210
400 K1 = J
410 K1 = CY(K1)
420 IF (K1, GT, N) GO TO 220
430 IF (K1, GT, N) GO TO 220
SUBROUTINE NUEC1(MD,FD,A,AN,IE,IH)
DIMENSION MD(1),FD(1),A(1),AN(1),IE(1),IH(1)

C *********************************************************
C * DECOMPOSE NONSYMMETRIC MATRIX USING GENERATED RECORD *
C *********************************************************

N = MD(1)
MD(4) = 0
FD(5) = 0.
FD(6) = 1.
FD(7) = 0.
FD(4) = 0.
DO 10 I=1,N
FD(7) = FD(7)+ALOG(AN(I))
10 FD(7) = 0.5*FD(7)

C CLEAR EXTRA STORAGE
DO 20 I=MD(5),MD(5)+1
MM = MD(5)+1
M1 = MD(6)
DO 20 I=MM,M1
A(I) = 0.
20 CONTINUE

C SEARCH FINISHED, REMOVE KR,KC FROM AVAILABLE
C PIVOT ROWS AND COLUMNS
DO 250 J=1,N
IF (IPR(I),NE,KR) GO TO 250
IPR(I) = KR
250 CONTINUE

C CONTINUE
C INITIALIZE FILES
50 CALL DRI
51 CALL NVW1
52 CALL NVWB(-1,-1)
C
57 C LOOP ON THE PIVOTS
58 DO 130 J=1,N
59 C GET PIVOT ADDRESS AND CHECK PIVOT MAGNITUDE
60 CALL DR(IX)
61 S = A(I)
62 IF (ABS(S),LE.FD(1)) GO TO 150
63 FD(5) = FD(5) + ALOG(ABS(S))
64 IF (S,LT.0.) FD(6) = -FD(6)
65 FD(4) = AMAX1(FD(4),ABS(S))
66 A(I) = 0.
67 CALL DR(KC)
68 CALL DR(KK)
69 C GET THE COLUMN ELEMENTS
70 CALL DR(K2)
71 IF (K2,LE.0) GO TO 50
72 DO 40 J=1,K2
73 CALL DR(K3)
74 CALL DR(IJ(J))
75 A(J) = A(K)/S
76 FD(4) = AMAX1(FD(4),ABS(A(J)))
77 A(K) = 0.
80 C GET THE ROW ELEMENTS
81 DO 90 J=1,K1
82 CALL DR(K3)
83 CALL DR(IJ(J))
84 A(J) = A(K3)*AN(J)
85 FD(4) = AMAX1(FD(4),ABS(A(J)))
86 A(K3) = 0.
87 C MODIFY CROSS-POINT ELEMENTS
88 IF (K2,LE.0) GO TO 80
89 DO 70 J=1,K2
90 DO 70 JJ=1,K1
91 CALL DR(K3)
92 CALL DR(IJ(J))
93 A(JJ) = A(K3)-A(J)*AN(JJ)
94 FD(4) = AMAX1(FD(4),ABS(A(JK)))
95 C WRITE OUT PIVOT ROW AND COLUMN
96 CALL NVWF(-KC,-KH)
97 IF (K2,LE.0) GO TO 100
98 DO 90 J=1,K2
99 CALL NVWF(IJ(J),A(J))
100 IF (K1,LE.0) GO TO 120
101 DO 110 J=1,K1
110 CALL NVWB(IJ(J),A(J))
111 CALL NVWB(-KC,S)
112 CONTINUE
113 C
114 CALL NVWE
115 RETURN
116 C MD(4) = 3
117 RETURN
C
150 END
SUBROUTINE SSLV(MD,X,Y)
DIMENSION MD(1),X(1),Y(1)
C ***********************************************************************
C * BACKSUBSTITUTION FOR SYMMETRIC DECOMPOSED MATRIX *
C ***********************************************************************
C
N = MD(1)
DO 10 I=1,N
10 Y(I) = X(I)
CALL SVRI
I = 0
C FORWARD BACKSUBSTITUTION
20 CALL SVRF(L2,Z)
   I = I+1
   L2 = -L2
   S = Y(L2)
   30 CALL SVRF(L2,Z)
   IF (L2.LT.0) GO TO 40
   Y(L2) = Y(L2)-Z*S
   GO TO 30
40 IF (I.LT.N) GO TO 20
C BACKWARD BACKSUBSTITUTION
50 CALL SVRB(L2,Z)
   I = I-1
   J = -L2
   60 Y(J) = Y(J)/Z
   70 IF (L2.LT.0) GO TO 70
   Y(J) = Y(J)-Z*Y(L2)
   GO TO 70
80 IF (I.GT.0) GO TO 50
RETURN
C END

SUBROUTINE NSLV(MD,X,Y,AN)
DIMENSION MD(1),X(1),Y(1),AN(1)
C ***********************************************************************
C * BACKSUBSTITUTION FOR NONSYMMETRIC DECOMPOSED MATRIX *
C ***********************************************************************
C
EQUIVALENCE (KS,S)
N = MD(1)
C SAVE RIGHT SIDE
DO 10 I=1,N
10 AN(I) = X(I)
C INITIALIZE FILES
CALL NVRI
I = 0
J = 0
C SOLVE LOWER TRIANGULAR SYSTEM
C        CALL NVRF(K4,S)
024      IF (K4,GT,0) GO TO 30
025      I = I+1
026      K4 = -K4
027      K3 = -K5
028      Y(K4) = AN(K3)
029      IF (I,GE,N) GO TO 40
030      D1 = AN(K3)
031      GO TO 20
032      AN(K4) = AN(K4)-D1*S
033      GO TO 20
034      C
035      C        SOLVE UPPER TRIANGULAR SYSTEM
036      C
037      40     CALL NVRU(K4,S)
038      IF (K4,GT,0) GO TO 50
039      J = J+1
040      IF (J,NE,1) Y(I) = Y(I)/D1
041      IF (J,GT,N) GO TO 60
042      IX = -K4
043      D1 = S
044      GO TO 40
045      50     Y(IX) = Y(IX)-S*K4
046      GO TO 40
047      C
048      60     RETURN
049      C
050      END
6.2 I/O Programs:

```fortran
C ********************************************
C * I/O ROUTINE FOR DECOMPOSITION PROCESSES *
C ********************************************
SUBROUTINE Dw1
C THE DECOMPOSITION PROCESS GENERATES AN ARRAY OF
C POSITIVE INTEGERS. THIS PROGRAM PROVIDES A BUFFERED
C INPUT/OUTPUT USING FILE 10. THE ENTRIES ARE AS FOLLOWS:
C
C IX IS THE BUFFER SIZE
C PARAMETER IX = 250
C DIMENSION IB(IX)

C ********************************************************
C 020 C * INITIALIZE FOR WRITING *
C 021 C ********************************************************
C 022 J = 1
C 023 REWIND 10
C 024 RETURN
C 025 C
C 026 ENTRY Dw(K)
C 027 ********************************************************
C 028 IB(J) = K
C 029 J = J+1
C 030 IF (J.LE.IX) RETURN
C 031 WRITE (10) IB
C 032 J = 1
C 033 RETURN
C 034 C
C 035 ENTRY Dwe
C 036 ********************************************************
C 037 IF (J.NE.1) WRITE (10) IB
C 038 RETURN
C 039 C
C 040 ENTRY Dri
C 041 ********************************************************
C 042 REWIND 10
C 043 J = IX
C 044 RETURN
C 045 C
C 046 ENTRY Dr(K)
C 047 ********************************************************
C 048 J = J+1
C 049 IF (J.LE.IX) GO TO 10
C 050 READ (10) IB
C 051 J = 1
C 052 RETURN
C 053 C
C 054 END
```
SUBROUTINE SYW1
C THE DECOMPOSED MATRIX IS PLACED IN FILE 11 AS A RANDOM
C ACCESS FILE. IT CONSISTS OF A DOUBLE ARRAY WHICH IS
C BUFFERED. ALTHOUGH THE FIRST PART OF THE ARRAY
C IS AN INTEGER (SIGNED) ARRAY, THIS ROUTINE DOES NOT
C PACK IT. THE ROUTINE HAS THE FOLLOWING ENTRIES:
C
SYW1 - INITIALIZE FOR WRITE
SYW(A1,A2) - WRITE A1,A2 AS NEXT ENTRY
SYWE - TERMINATE WRITE
SVRI - INITIALIZE FOR READ
SVRF(A1,A2) - READ NEXT ENTRY A1,A2
SVRB(A1,A2) - READ PREVIOUS ENTRY A1,A2

C THE ROUTINE ASSUMES THAT THE WRITTEN ARRAY IS READ ONCE
C FORWARD THEN READ BACKWARD.
PARAMETER NX = 100
PARAMETER MXY = 2*MX
NX IS THE MAXIMUM NUMBER OF RECORDS
MXY IS THE LENGTH OF THE RECORDS
DIMENSION B(2,MX)

ENTRY SVW(A1,A2)

ENTRY SVWE

ENTRY SVRI

PARAMETER MX = 200
PARAMETER MXX = 2*MX

ENTRY SVRF(A1,A2)

ENTRY SVRB(A1,A2)

ENTRY SVRb(Al,A2)

ENTRY SVW(Al,A2)

ENTRY SVRF(Al,A2)

ENTRY SVRB(Al,A2)

ENTRY SVRb(Al,A2)

ENTRY SVW(Al,A2)

ENTRY SVRF(Al,A2)

ENTRY SVRB(Al,A2)

ENTRY SVRb(Al,A2)

ENTRY SVW(Al,A2)

ENTRY SVRF(Al,A2)

ENTRY SVRB(Al,A2)

ENTRY SVRb(Al,A2)

ENTRY SVW(Al,A2)

ENTRY SVRF(Al,A2)

ENTRY SVRB(Al,A2)

ENTRY SVRb(Al,A2)
C ENTRY SVRF(A1,A2)
C *****************************************
C * READ NEXT ENTRY A1,A2 *
C *****************************************
C
J = J+1
IF (J.LE.MX) GO TO 10
M = M+1
READ (11,N) B
J = 1
10 A1 = H(1,J)
A2 = B(2,J)
RETURN
C ENTRY SVRB(A1,A2)
C *****************************************
C * READ PREVIOUS ENTRY A1,A2 *
C *****************************************
C
IF (J.GT.U) GO TO 20
M = M-1
READ (11,N) B
J = MX
20 A1 = B(1,J)
A2 = B(2,J)
J = J-1
RETURN
C END
**I/O ROUTINE FOR NONSYMMETRIC DECOMPOSED MATRIX**

**SUBROUTINE NVWI**

The lower and upper triangular matrices of the decomposed nonsymmetric matrix are contained in files 12 and 13, respectively, as random access files. They are in the form of buffered double arrays. The entries are as follows:

- **NVWF(A1,A2)** - Initialize for write
- **NVWB(A1,A2)** - Write A1,A2 as next entry on file 12
- **NVBR(A1,A2)** - Read previous entry from file 13
- **NVRF(Al,A2)** - Read next entry from file 12

**FILE 12 IS READ FORWARD, FILE 13 BACKWARD.**

**PARAMETER NX = 100**
**PARAMETER MX = 250**
**PARAMETER MXX = 2*MX**

**NX IS THE MAXIMUM NUMBER OF RECORDS,**
**MXX IS THE RECORD SIZE**

**DIMENSION B12(2,MAX), B13(2,MAX)**

**ENTRY NVWF(A1,A2)**

**ENTRY NVWB(A1,A2)**

**ENTRY NVRF(A1,A2)**

**ENTRY NVRF(A1,A2)**
C ENTER NVWE
C
C ***********
C *
C *****************
C
C
C
071 IF (J12, EQ, 1) GO TO 10
072 N12 = N12 + 1
073 WRITE (12'N12) B12
074 JJ = MX
075 IF (J13, EQ, 1) RETURN
076 N13 = N13 + 1
077 WRITE (13'N13) B13
078 JJ = J13 - 1
079 RETURN
080 C
081 C ENTER NVR1
082 C ***********
083 C * INITIALIZE READ-IN *
084 C
085 C
086 C
087 C
088 C
089 C
090 N2 = U
091 N3 = N13 + 1
092 J2 = MX
093 J3 = U
094 RETURN
095 C
096 C ENTER NVRF(A1, A2)
097 C ***********
098 C * READ A1, A2 FROM FILE 12 *
099 C
100 C
101 J2 = J2 + 1
102 IF (J2, LE, MX) GO TO 20
103 N2 = N2 + 1
104 READ (12'N2) A12
105 J2 = 1
106 A1 = B12(1, J2)
107 A2 = B12(2, J2)
108 RETURN
109 C
110 C ENTER NVRB(A1, A2)
111 C ***********
112 C * READ PREVIOUS ENTRY A1, A2 FROM FILE 13 *
113 C
114 C
115 IF (J3, GT, 0) GO TO 30
116 N3 = N3 - 1
117 READ (13'N3) B13
118 J3 = MX
119 IF (N3, EQ, N13) J3 = JJ
120 A1 = B13(1, J3)
121 A2 = B13(2, J3)
122 J3 = J3 - 1
123 RETURN
124 C
125 C ENO
References


Further Programs for the Solution of Large Sparse Systems of Linear Equations

C. K. Mesztenyi
W. C. Rheinboldt

Computer Science Center
University of Maryland
College Park, Maryland 20742

August 1975

Approved for Public Release; Distribution Unlimited.

sparse linear systems
triangular systems
FORTAN programs

A package of FORTAN subroutines is presented for the solution of non-symmetric or symmetric sparse linear systems by triangular decomposition. Two principal aims are (1) to handle matrices which originally fit into primary core storage but do no longer after decomposition, and (2) to solve a sequence of linear systems all of which have the same sparsity structure by generating--in secondary storage--a record of the decomposition process in the form of an integer array. Some experimental results using the package are included.