Abstract This report presents subroutines that implement the envelope algorithm for the solution of sparse linear systems.

Subroutines for Envelope Solution of Sparse Linear Systems

Stanley C. Eisenstat and Andrew H. Sherman

Research Report #35

October 1974

This research was supported in part by the Office of Naval Research, N0014-67-A-0097-0016.

Subroutines for Envelope Solution of Sparse Linear Systems

1. Introduction

Consider the system of linear equations

$$A\underset{\sim}{x} = \underset{\sim}{b} \tag{1.1}$$

where A is a sparse N by N matrix. If A is nonsymmetric, assume that it may be factored into the product LU where L is lower triangular and U is unit upper triangular. If A is symmetric, assume that it may be factored into the product LDL^t where L is unit lower triangular and D is diagonal.*

The solution may then be obtained by successively solving either

$$L\underset{\sim}{y} = \underset{\sim}{b} , \quad U\underset{\sim}{x} = \underset{\sim}{y} \tag{1.2}$$

or

$$L\underset{\sim}{y} = \underset{\sim}{b} , \quad D\underset{\sim}{z} = \underset{\sim}{y} , \quad L^t\underset{\sim}{x} = \underset{\sim}{z} . \tag{1.3}$$

This paper describes a set of programs which implement the envelope algorithm for such direct solutions for (1.1).

---

*To be efficient, an algorithm must attempt to reduce the storage and work required by taking advantage of the known zero structure of A. To do this it may be advantageous to solve the permuted system

$$PAP^t (P\underset{\sim}{x}) = P\underset{\sim}{b}$$

instead of (1.1), and for any permutation matrix P, we assume that the permuted system may be solved in exactly the same manner as (1.1). For convenience we may refer to the matrix A or the system (1.1) when we actually mean the permuted matrix or system. This involves no loss of generality since the systems are assumed to have similar numerical properties.

There are three main factorization algorithms: band, envelope, and general sparse. The first two of these take advantage of the zero structure of A by storing only those elements of A which lie within particular regions of the matrix. The regions are defined so that all the nonzero elements in A and its factorization lie within these regions and so that the data structures for the matrix storage are quite simple. In contrast, the general sparse algorithm stores and operates on only those elements of A and its factorization which are actually nonzero, so that it is generally far more efficient in terms of arithmetic operations than a band or envelope algorithm. However, this efficiency is gained only at the cost of additional complexity in the data structures and programs required for implementation. This paper deals with the envelope factorization algorithm because it combines practical efficiency with simplicity. Compared with the band algorithm, it often requires much less storage and work, yet it is no more difficult to implement. And compared with the general sparse algorithm, it requires less complex data structures and programs, while still achieving a large degree of practical efficiency.

In what follows we discuss the theoretical background of the envelope algorithm and describe the algorithm and its associated data structures. Appendix A contains listings of the actual FORTRAN subroutines, and Appendix B contains the listing of a driver program which illustrates the use of the subroutines.

2. Envelope Methods

Let A be a given N by N matrix, and let $f_i(A)$ $(f_i^t(A))$ denote the column (row) index of the first nonzero element of the i-th row (column) of A;

$$f_i(A) \equiv \min\{j : a_{ij} \neq 0\} \; ; \tag{2.1}$$

$$f_i^t(A) \equiv \min\{j : a_{ji} \neq 0\} \; . \tag{2.2}$$

We then define the "bandwidth" $\beta_i(A)$ ($\beta_i^t(A)$) of the i-th row (column) of A:

$$\beta_i(A) \equiv i - f_i(A) \; ; \tag{2.3}$$

$$\beta_i^t(A) \equiv i - f_i^t(A) \; ; \tag{2.4}$$

and the "frontwidth" $\omega_i(A)$ ($\omega_i^t(A)$) of the i-th row (column) of A:

$$\omega_i(A) \equiv |\{k : k>i \text{ and } \exists \ell \leq i \text{ such that } a_{k\ell} \neq 0\}|^* ; \tag{2.5}$$

$$\omega_i^t(A) \equiv |\{k : k>i \text{ and } \exists \ell \leq i \text{ such that } a_{\ell k} \neq 0\}| \; . \tag{2.6}$$

If A is nonsymmetric, then the envelope of A is the region of A defined by the following set of ordered pairs denoting positions in A:

$$\text{Env}(A) \equiv \{(i,j) : f_i(A) \leq j \leq i \text{ and } f_j^t(A) \leq i \leq j\} \; . \tag{2.7}$$

Its size may be expressed in terms of the bandwidths or the frontwidths:

$$|\text{Env}(A)| = N + \sum_{i=1}^{N} (\beta_i(A) + \beta_i^t(A)) = N + \sum_{i=1}^{N} (\omega_i(A) + \omega_i^t(A)) . \tag{2.8}$$

If A is symmetric, then only the lower triangle of the envelope need be stored, and the symmetric envelope of A is defined by

---

* For a set S, we denote the number of elements in S by $|S|$.

$$\text{Senv}(A) \equiv \{(i,j): f_i(A) \leq j \leq i\} . \tag{2.9}$$

The size of the symmetric envelope may also be expressed in terms of the bandwidths or frontwidths:

$$|\text{Senv}(A)| = N + \sum_{i=1}^{N} \beta_i(A) = N + \sum_{i=1}^{N} \omega_i(A) . \tag{2.10}$$

Where there is no chance of confusion, we will use the term envelope (symmetric envelope) to refer to the actual nonzeros in Env(A) (Senv(A)) as well as to the positions in Env(A) (Senv(A)).

When A is factored into a product LU or LDL^t some of the zero entries in A fill in (i.e., become nonzero entries in L or U). It is well known, however, (see [7]) that for nonsymmetric matrices A

$$\text{Env}(L + U) \subseteq \text{Env}(A) , \tag{2.11}$$

while for symmetric matrices A,

$$\text{Senv}(L) \subseteq \text{Senv}(A) . \tag{2.12}$$

Hence the number of locations required to store all the nonzero entries in the factorization of A is no larger than the size of the envelope or symmetric envelope of A. For envelope methods, which do not exploit zeros inside the envelope, the storage required is exactly equal to $|\text{Env}(A)|$ or $|\text{Senv}(A)|$.

Algorithms 2.1 and 2.2, respectively, give algorithms for the LU and LDL^t factorizations. The lower bounds on the summations reflect the fact that exactly the envelopes of the matrices involved are stored.

<u>Algorithm 2.1</u>:

```
For  i ← 1 to N  do
        [For  j ← $f_i$ to i-1  do
                $\ell_{ij} \leftarrow a_{ij} - \sum_{k=\max(f_i,f_j^t)}^{j-1} \ell_{ik}\, u_{kj}$ ;
         For  j ← $f_i^t$ to i-1  do
                $u_{ji} \leftarrow \frac{1}{\ell_{jj}} (a_{ji} - \sum_{k=\max(f_i^t,f_j)}^{j-1} \ell_{jk} u_{ki})$ ;
         $u_{ii} \leftarrow 1$ ;
         $\ell_{ii} \leftarrow a_{ii} - \sum_{k=\max(f_i,f_i^t)}^{i-1} \ell_{ik}\, u_{ki}$ ] ;
```

<u>Algorithm 2.2</u> ([10]):

```
For  i ← 1 to N  do
        [For j ← $f_i$ to i-1  do
                $a'_{ij} \leftarrow a_{ij} - \sum_{k=\max(f_i,f_j)}^{j-1} a'_{ik}\, \ell_{jk}$ ;
         For  j ← $f_i$ to i-1  do
                $\ell_{ij} \leftarrow a'_{ij} \,/\, d_{jj}$ ;
         $d_{ii} \leftarrow a_{ii} - \sum_{k=f_i}^{i-1} a'_{ik}\, \ell_{ik}$ ] ;
```

A (symmetric) matrix A is said to have a monotone (symmetric) envelope if

$$i \le j \Rightarrow f_i(A) \le f_j(A) \quad \text{and} \quad f_i^t(A) \le f_j^t(A). \tag{2.13}$$

The following theorems, similar to results of George [6], characterize the work required for the factorization of A with Algorithms 2.1 and 2.2. (For proofs, see [14].)

<u>Theorem 2.1</u>: If the LU factorization of the nonsymmetric matrix A requires $\Theta(A)$ multiplications, then

$$\Theta(A) = \sum_{i=1}^{N} \omega_i(A)[\omega_i^t(A) + 1] \le \sum_{i=1}^{N} \beta_i(A)[\beta_i^t(A) + 1] , \tag{2.14}$$

with equality exactly when A has a monotone envelope.

<u>Theorem 2.2</u>: If the LDL^t factorization of the symmetric matrix A requires $\Theta(A)$ multiplications, then

$$\Theta(A) = \sum_{i=1}^{N} \omega_i(A)[\omega_i(A) + 3]/2 \le \sum_{i=1}^{N} \beta_i(A)[\beta_i(A) + 3]/2 , \tag{2.15}$$

with equality exactly when A has a monotone symmetric envelope.

To reduce the amount of storage or work required by Algorithms 2.1 and 2.2, it is necessary to select a permutation matrix P (corresponding to an ordering of the variables and equations of (1.1)) so that $|\mathrm{Env}(PAP^t)|$ or $\Theta(PAP^t)$ is small. We restrict this discussion to systems (1.1) in which the zero structure of A is symmetric (i.e., $a_{ij} \ne 0$ implies $a_{ji} \ne 0$), since little research has been done for more general matrices. Even with this restriction, exhaustive search is the only means known for optimally ordering the variables and equations of (1.1). But there are several algorithms which seem to give good results in practice (see [2], [3], [5], [8], [11], [12], [13]). Of these, we will describe only the Reverse Cuthill-McKee (RCM) algorithm which appears to offer a good practical tradeoff between the cost of obtaining the ordering and the resulting values of $|\mathrm{Env}(PAP^t)|$ and $\Theta(PAP^t)$.

It is convenient to introduce the directed graph $G(A)$ associated with the matrix A. That graph $G(A) = (X(A), E(A))$ is defined as the set of vertices $X(A) = \{x_1,x_2,\ldots,x_N\}$ and the set of directed edges $E(A) = \{(x_i,x_j):a_{ij} \neq 0,\ i \neq j\}$ joining pairs of vertices in $X(A)$. The vertices in $X(A)$ correspond to and are labelled as the rows of A. The adjacency of a vertex x_i in $X(A)$ is defined by

$$\mathrm{Adj}(x_i) = \{x_j:(x_i,x_j) \in E(A)\} . \qquad (2.16)$$

Since A has symmetric zero structure, $(x_i,x_j) \in E(A)$ if and only if $(x_j,x_i) \in E(A)$, so that we may define the degree of a vertex x_i in $X(A)$ as

$$\mathrm{Deg}(x_i) = |\mathrm{Adj}(x_i)| . \qquad (2.17)$$

For any permutation matrix P, the graphs $G(A)$ and $G(PAP^t)$ are identical up to a relabelling of the vertices.

The version of the RCM algorithm given here assumes that $G(A)$ is connected (see [9, p. 13]). Algorithm 2.3 obtains the RCM ordering by reversing the Cuthill-McKee (CM) ordering [4], which corresponds to the breadth-first generation of a spanning tree for G(A) (see [9, pp. 11,32]) in a level-by-level fashion. The root of the spanning tree is labelled first. As each vertex is labelled, its unmarked neighbors are marked and added to a list of marked vertices in order of increasing degree. This list is kept in a first-in-first-out data structure or queue, and the vertices are labelled and added to the tree in the order in which they appear in the queue.

Algorithm 2.3:

```
      Choose a vertex of minimum degree, mark it,
      and place it in the queue.  $k \leftarrow N$.
LOOP: Remove the oldest vertex $s$ from the queue, and
      label it as vertex $k$.
      $k \leftarrow k - 1$.
      Mark the unmarked neighbors of $s$, and add them
      to the queue in order of increasing degree.  If
      the queue is not empty, then go to LOOP.
      Otherwise, stop.
```

In general, the RCM ordering is not optimal, but experiments performed by George [5] and Liu and Sherman [13] indicate that it is quite effective in practice, particularly on problems arising in the numerical solution of partial differential equations.

3. Program Descriptions

The programs described in this section are based on the algorithms of Section 2 and on the data structures used to represent the graph and envelope forms of matrices.

The graph G(C) of an N by N matrix C is stored in adjacency list form -- for each of the N vertices we keep a list of the vertices to which it is adjacent. Two data arrays are required: IA to contain the vertex adjacency lists stored sequentially, and IV to contain a pointer for each vertex to its adjacency list in IA. By convention, the adjacency lists are stored so that IV(I) > IV(J) if I > J; IV(N+1) points to the first unused entry in IA; and each vertex is included in its own adjacency list. The matrix C is said to be stored in adjacency list form if its nonzero elements are stored in another array A so that A(K) contains c_{IJ} if IA(K) is the entry for vertex J in the adjacency list of vertex I. This scheme allows for storage of both symmetric and nonsymmetric matrices and graphs (see Figure 3.1).

The storage of the envelope form of the matrix C requires arrays both to describe the structure of the envelope and to contain the actual nonzero elements of C. It is most illustrative here to give the representation of a nonsymmetric matrix. If C has symmetric zero structure, then only the pointers for the lower triangle need be kept, and if C is symmetric, then only its symmetric envelope (i.e. the lower triangle of its envelope) and the associated pointers are stored. The elements of the envelope of the strict lower (upper) triangle of C are stored in PL (PU) row by row (column by column), and the diagonal of C is stored in D. An array IRL (IRU) is defined so that IRL(I) (IRU(I)) points to the nonexistent c_{I0} (c_{0I}) element of the I th row (column) of the strict lower (upper) triangle of C in PL (PU). In effect IRL(I) (IRU(I)) is the base address for the I th row (column) of C in PL (PU). Then c_{IJ} and c_{JI} $(I > J)$, if they are stored, are easily located:

$$c_{IJ} = \mathrm{PL(IRL(I) + J)} \tag{3.1}$$

$$c_{JI} = \mathrm{PU(IRU(I) + J)} \ . \tag{3.2}$$

Figures 3.2, 3.3, and 3.4, respectively, show examples of this storage scheme for the three types of matrices discussed above.

All of the subroutines described here are written in ANSI Standard FORTRAN [1], and standard type defaults have been used for all variables. No double precision subroutines have been included, but it is easy to modify the given routines by declaring REAL variables to be DOUBLE PRECISION where appropriate. The descriptions are given from a user's point of view, so detailed comments have been left to the program listings which appear in Appendix A. Appendix B contains a driver program which illustrates the use of the subroutines described here.

Subroutine -- RCM

Purpose --

The subroutine RCM computes the Reverse Cuthill-McKee (RCM) ordering of a graph using Algorithm 2.3.

Calling Sequence -- CALL RCM(N, IV, IA, IORD, IPOS)

Parameters --

N	is an integer equal to the number of vertices in the graph to be ordered.
IV	is an integer array of length N+1. For $1 \le I \le N$, IV(I) points to the adjacency list of the I-th vertex in IA. IV(N+1) points to the first unused entry of IA.
IA	is an integer array containing the adjacency lists for the vertices of the graph to be ordered.
IORD	is an integer array of length N which on output contains the RCM ordering.
IPOS	is an integer array of length N which on output contains the inverse of the RCM ordering (i.e. IPOS(IORD(I)) = I).

Discussion of Method --

The RCM ordering is computed using Algorithm 2.3. In the subroutine the queue is kept in IORD, since no more than N elements are ever placed in it. IPOS(I) is used as a flag for vertex I. Initially, IPOS(I) = 0; when vertex I is marked, IPOS(I) is set to the negative of the degree of vertex I; when vertex I is ordered as the K-th vertex in the RCM ordering, IPOS(I) is set to K. When more than one vertex is added to the queue at once, a simple insertion sort is used to add them in order of increasing degree in the graph.

Subroutine -- GENENV

Purpose --

Given the adjacency list form of an input matrix C and two arrays describing the vertex ordering, the subroutine GENENV constructs the ordered envelope form of $P\ C\ P^t$, where the permutation matrix P corresponds to the input vertex ordering.

Calling Sequence --

```
CALL GENENV(N,MAXPL,PL,D,MAXPU,PU,IRL,IRU,IV,IA,A,IORD,IPOS,IFLAG)
```

Parameters --

N	is an integer equal to the number of rows in C.
MAXPL	is an integer equal to the maximum storage available for PL.
PL	is a real array which on output contains the elements of the strict lower triangle of the envelope of $P\ C\ P^t$.
D	is a real array which on output contains the elements of the diagonal of $P\ C\ P^t$.
MAXPU	is an integer equal to the maximum storage available for PU.
PU	is a real array which on output contains the elements of the strict upper triangle of the envelope of $P\ C\ P^t$.
IRL	is an integer array of length N which on output contains pointers to the nonexistent c_{I0} elements in PL.
IRU	is an integer array of length N which on output contains pointers to the nonexistent c_{0I} elements in PU.
IV	is an integer array of length N+1. For $1 \le I \le N$, IV(I) points to the adjacency list of the I-th vertex in IA. IV(N+1) points to the first unused entry of IA.
IA	is an integer array which contains the adjacency lists of the vertices of G(C) (the graph of C).
A	is a real array which on input contains the nonzero elements corresponding to the graph form of the input matrix C.

IORD	is an integer array of length N which contains the ordering of the vertices of G(C) corresponding to the permutation matrix P.
IPOS	is an integer array of length N which contains the inverse of IORD (i.e. IPOS(IORD(I)) = I).
IFLAG	is an integer variable which is used to return error indications.

IFLAG = 0	if no errors are encountered;
IFLAG = -1	if insufficient storage is available for PL;
IFLAG = +1	if insufficient storage is available for PU.

Discussion of Method --

The entries of IRL and IRU are computed first. If insufficient storage is available for PL (PU), IFLAG is set to -1 (+1), and processing is terminated. (IFLAG reflects the first error which occurs.) Otherwise, all the elements in the envelope of $P\ C\ P^t$ are stored in PL and PU in one pass through the data in A. If C has symmetric zero structure, set IRU = IRL and MAXPU = MAXPL when calling subroutine GENENV. If C is symmetric, set PU = PL, IRU = IRL, and MAXPU = MAXPL when calling subroutine GENENV.

Subroutine -- PLU

Purpose --

The subroutine PLU computes the L U factorization of an input matrix C stored in envelope form. L is lower triangular, and U is unit upper triangular. If C is symmetric, use subroutine PLDLT instead of subroutine PLU.

Calling Sequence -- CALL PLU(N,PL,D,PU,IRL,IRU)

Parameters --

N	is an integer equal to the number of rows in C.
PL	is a real array which on input contains the elements of the strict lower triangle of the envelope of C, and on output contains the elements of the strict lower triangle of the envelope of L.
D	is a real array which on input contains the elements of the diagonal of C, and on output contains the reciprocals of the elements of the diagonal of L (D(I) = $1/\ell_{II}$).
PU	is a real array which on input contains the elements of the strict upper triangle of the envelope of C, and on output contains the elements of the strict upper triangle of the envelope of U.
IRL	is an integer array of length N which contains pointers to the nonexistent c_{I0} elements in PL.
IRU	is an integer array of length N which contains pointers to the nonexistent c_{0I} elements in PU.

Discussion of Method --

The factorization is performed using Algorithm 2.1. L and U overwrite PL, D, and PU. If C has symmetric zero structure, set IRU = IRL when calling subroutine PLU.

Subroutine -- PLDLT

Purpose --

The subroutine PLDLT computes the $L\,D\,L^t$ factorization of a symmetric input matrix C stored in envelope form. L is unit lower triangular, and D is diagonal.

Calling Sequence -- CALL PLDLT(N, PL, D, IRL)

Parameters --

N	is an integer equal to the number of rows in C.
PL	is a real array which on input contains the elements of the strict lower triangle of the envelope of C, and on output contains the elements of the strict lower triangle of the envelope of L.
D	is a real array which on input contains the elements of the diagonal of C, and on output contains the reciprocals of the elements of the diagonal of D $(D(I) = 1/d_{II})$.
IRL	is an integer array of length N which contains pointers to the nonexistent c_{I0} elements in PL.

Discussion of Method --

The factorization is performed using Algorithm 2.2. L and D overwrite PL and D, respectively.

Subroutine -- PLUB

Purpose --

The subroutine PLUB solves the system $P\ C\ P^t\ P\underset{\sim}{x} = P\underset{\sim}{b}$, by performing the backsolving operations necessary to solve $L\ U\ P\underset{\sim}{x} = P\underset{\sim}{b}$, where $P\ C\ P^t = L\ U$. L is lower triangular, U is unit upper triangular, and both matrices are stored in envelope form.

Calling Sequence -- CALL PLUB(N, PL, D, PU, IRL, IRU, X, B, IORD)

Parameters --

N	is an integer equal to the number of rows in L and U.
PL	is a real array which contains the elements of the strict lower triangle of the envelope of L.
D	is a real array which contains the reciprocals of the elements of the diagonal of L ($D(I) = 1/\ell_{II}$).
PU	is a real array which contains the elements of the strict upper triangle of the envelope of U.
IRL	is an integer array of length N which contains pointers to the nonexistent ℓ_{I0} elements in PL.
IRU	is an integer array of length N which contains pointers to the nonexistent u_{0I} elements in PU.
X	is a real array of length N which on output contains the solution vector.
B	is a real array of length N which contains the right hand side.
IORD	is an integer array of length N which contains the ordering of the rows and columns of C corresponding to the permutation matrix P.

Discussion of Method --

This routine successively solves $L\ \underset{\sim}{y} = P\underset{\sim}{b}$ and $U\ \underset{\sim}{x} = \underset{\sim}{y}$. The solution vector X is reordered corresponding to P. If the zero structure of U is the transpose of that of L, set IRU = IRL when calling subroutine PLUB.

Subroutine -- PLDLTB

Purpose --

The subroutine PLDLTB obtains the solution to the system $P\,C\,P^t\,P\underset{\sim}{x} = P\underset{\sim}{b}$ by performing the backsolving operations necessary to solve $L\,D\,L^t\,P\underset{\sim}{x} = P\underset{\sim}{b}$, where $P\,C\,P^t = L\,D\,L^t$. L is unit lower triangular, $\underset{\sim}{D}$ is diagonal, and L is stored in envelope form.

Calling Sequence -- CALL PLDLTB(N, PL, D, IRL, X, B, IORD)

Parameters --

N	is an integer equal to the number of rows in L and D.
PL	is a real array which contains the elements of the strict lower triangle of the envelope of L.
D	is a real array which contains the reciprocals of the elements of the diagonal of D ($D(I) = 1/d_{II}$).
IRL	is an integer array of length N which contains pointers to the nonexistent ℓ_{I0} elements in PL.
X	is a real array of length N which on output contains the solution vector.
B	is a real array of length N which contains the right hand side.
IORD	is an integer array of length N which contains the ordering of the rows and columns of C corresponding to the permutation matrix P.

Discussion of Method --

This routine successively solves $L\,\underset{\sim}{y} = P\underset{\sim}{b}$, $D\,\underset{\sim}{z} = \underset{\sim}{y}$, and $L^t\,\underset{\sim}{x} = \underset{\sim}{z}$. The solution vector X is reordered corresponding to P.

References

[1] American National Standards Institute, American National Standard FORTRAN, ANS X3.9 - 1966, New York, 1966.

[2] J.H. Bolstad, G.K. Leaf, A.J. Lindeman, and H.G. Kaper, An Empirical Investigation of Reordering and Data Management for Finite Element Systems of Equations, Argonne National Laboratory, Technical Report #ANL - 8056, Argonne, Illinois, 1973.

[3] E. Cuthill, Several Strategies for Reducing the Bandwidth of Matrices, Sparse Matrices and their Applications, D.J.Rose and R.A. Willoughby, eds., Plenum Press, New York, 1972, pp. 157-166.

[4] E. Cuthill and J. McKee, Reducing the Bandwidth of Sparse Symmetric Matrices, Proc. 24th Nat. Conf. of the ACM, ACM Publication P-69, 1122 Ave. of the Americas, New York, 1969, pp. 157-172.

[5] J.A. George, Computer Implementation of the Finite Element Method, Stanford Computer Science Dept., Technical Report STAN-CS-71-208, Stanford, California, 1971.

[6] J.A. George, A Survey of Sparse Matrix Methods in the Direct Solution of Finite Element Equations, Proc. Summer Simulation Conf., Montreal, Canada, July 17-19, 1973, pp. 15-20.

[7] J.A. George and W.H. Liu, A Note on Fill for Sparse Matrices, submitted to SIAM J. Numer. Anal.

[8] N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer, An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix, Department of Mathematics, College of William and Mary, Technical Report #5, Williamsburg, Virginia, 1974, submitted to SIAM J. Numer. Anal.

[9] F. Harary, Graph Theory, Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.

[10] A. Jennings, A Compact Storage Scheme for the Solution of Symmetric Simultaneous Equations, Comput. J. 9(1966), pp. 281-285.

[11] I.P. King, An Automatic Re-Ordering Scheme for Simultaneous Equations Derived from Network Analyses, Inter. J. Numer. Meth. Engrg. 2(1970), pp. 523-533.

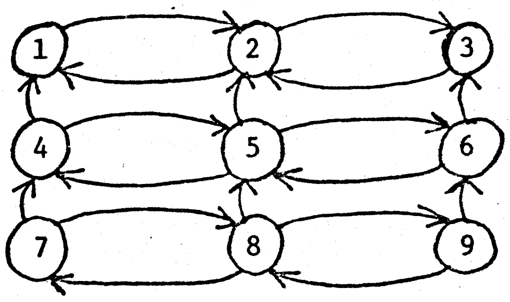
[12] R. Levy, Restructuring the Structural Stiffness Matrix to Improve Computational Efficiency, JPL Tech. Rev. 1(1971), pp. 61-70.

[13] W.H. Liu and A.H. Sherman, Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices, Department of Computer Science, Yale University, Technical Report #28, New Haven, Connecticut, 1974, submitted to SIAM J. Numer. Anal.

[14] A.H. Sherman, Ph.D. dissertation, Department of Computer Science, Yale University, New Haven, Connecticut, to appear.

$$C = \begin{bmatrix}
4.0 & -0.8 & & & & & & & \\
-0.9 & 4.0 & -0.8 & & & & \bigcirc & & \\
0.0 & -0.9 & 4.0 & & & & & & \\
-1.0 & 0.0 & 0.0 & 4.0 & -0.8 & & & & \\
 & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 & & & \\
 & & -1.0 & 0.0 & -0.9 & 4.0 & & & \\
 & & & -1.0 & 0.0 & 0.0 & 4.0 & -0.8 & \\
 & \bigcirc & & & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 \\
 & & & & & -1.0 & 0.0 & -0.9 & 4.0
\end{bmatrix}$$

G(C):



I	IV(I)	IA(I)	A(I)
1	1	1	4.0
2	3	2	-0.8
3	6	1	-0.9
4	8	2	4.0
5	11	3	-0.8
6	15	2	-0.9
7	18	3	4.0
8	21	1	-1.0
9	25	4	4.0
10	28	5	-0.8
11		2	-1.0
12		4	-0.9
13		5	4.0
14		6	-0.8
15		3	-1.0
16		5	-0.9
17		6	4.0
18		4	-1.0
19		7	4.0
20		8	-0.8
21		5	-1.0
22		7	-0.9
23		8	4.0
24		9	-0.8
25		6	-1.0
26		8	-0.9
27		9	4.0

Figure 3.1: Adjacency List Form of a Nonsymmetric Matrix

$$C = \begin{bmatrix} 4.0 & -0.8 & & & & & & & \\ -0.9 & 4.0 & -0.8 & & & & \bigcirc & & \\ 0.0 & -0.9 & 4.0 & & & & & & \\ -1.0 & 0.0 & 0.0 & 4.0 & -0.8 & & & & \\ & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 & & & \\ & & -1.0 & 0.0 & -0.9 & 4.0 & & & \\ & & & -1.0 & 0.0 & 0.0 & 4.0 & -0.8 & \\ & \bigcirc & & & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 \\ & & & & & -1.0 & 0.0 & -0.9 & 4.0 \end{bmatrix}$$

I	IRL(I)	PL(I)	D(I)	IRU(I)	PU(I)
1	0	-0.9	4.0	0	-0.8
2	0	-0.9	4.0	0	-0.8
3	0	-1.0	4.0	0	-0.8
4	2	0.0	4.0	0	-0.8
5	4	0.0	4.0	-1	-0.8
6	6	-1.0	4.0	-1	-0.8
7	8	0.0	4.0	-1	
8	10	-0.9	4.0	-2	
9	12	-1.0	4.0	-2	
10		0.0			
11		-0.9			
12		-1.0			
13		0.0			
14		0.0			
15		-1.0			
16		0.0			
17		-0.9			
18		-1.0			
19		0.0			
20		-0.9			

Figure 3.2: Envelope Storage of a Nonsymmetric Matrix

$$C = \begin{bmatrix}
4.0 & -0.8 & 0.0 & -0.7 & & & & & \\
-0.9 & 4.0 & -0.8 & 0.0 & -0.7 & & & \bigcirc & \\
0.0 & -0.9 & 4.0 & 0.0 & 0.0 & -0.7 & & & \\
-1.0 & 0.0 & 0.0 & 4.0 & -0.8 & 0.0 & -0.7 & & \\
 & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 & 0.0 & -0.7 & \\
 & & -1.0 & 0.0 & -0.9 & 4.0 & 0.0 & 0.0 & -0.7 \\
 & \bigcirc & & -1.0 & 0.0 & 0.0 & 4.0 & -0.8 & 0.0 \\
 & & & & -1.0 & 0.0 & -0.9 & 4.0 & -0.8 \\
 & & & & & -1.0 & 0.0 & -0.9 & 4.0
\end{bmatrix}$$

I	IRL(I)	PL(I)	D(I)	PU(I)
1	0	-0.9	4.0	-0.8
2	0	-0.9	4.0	-0.8
3	0	-1.0	4.0	-0.7
4	2	0.0	4.0	0.0
5	4	0.0	4.0	0.0
6	6	-1.0	4.0	-0.7
7	8	0.0	4.0	0.0
8	10	-0.9	4.0	-0.8
9	12	-1.0	4.0	-0.7
10		0.0		0.0
11		-0.9		-0.8
12		-1.0		-0.7
13		0.0		0.0
14		0.0		0.0
15		-1.0		-0.7
16		0.0		0.0
17		-0.9		-0.8
18		-1.0		-0.7
19		0.0		0.0
20		-0.9		-0.8

Figure 3.3: Envelope Storage of a Matrix with Symmetric Zero Structure

$$C = \begin{bmatrix} 4.0 & -1.0 & 0.0 & -1.0 & & & & & \\ -1.0 & 4.0 & -1.0 & 0.0 & -1.0 & & & \bigcirc & \\ 0.0 & -1.0 & 4.0 & 0.0 & 0.0 & -1.0 & & & \\ -1.0 & 0.0 & 0.0 & 4.0 & -1.0 & 0.0 & -1.0 & & \\ & -1.0 & 0.0 & -1.0 & 4.0 & -1.0 & 0.0 & -1.0 & \\ & & -1.0 & 0.0 & -1.0 & 4.0 & 0.0 & 0.0 & -1.0 \\ & & & -1.0 & 0.0 & 0.0 & 4.0 & -1.0 & 0.0 \\ & \bigcirc & & & -1.0 & 0.0 & -1.0 & 4.0 & -1.0 \\ & & & & & -1.0 & 0.0 & -1.0 & 4.0 \end{bmatrix}$$

I	IRL(I)	PL(I)	D(I)
1	0	-1.0	4.0
2	0	-1.0	4.0
3	0	-1.0	4.0
4	2	0.0	4.0
5	4	0.0	4.0
6	6	-1.0	4.0
7	8	0.0	4.0
8	10	-1.0	4.0
9	12	-1.0	4.0
10		0.0	
11		-1.0	
12		-1.0	
13		0.0	
14		0.0	
15		-1.0	
16		0.0	
17		-1.0	
18		-1.0	
19		0.0	
20		-1.0	

Figure 3.4: Envelope Storage of a Symmetric Matrix

Appendix A

This appendix contains the listings of the subroutines described in Section 3. Machine readable versions are currently available from:

Andrew H. Sherman
Yale University
Department of Computer Science
10 Hillhouse Avenue
New Haven, Connecticut 06520

```
      SUBROUTINE RCM(N,IV,IA,IORD,IPOS)
      DIMENSION IA(1),IV(1),IORD(1),IPOS(1)
C
C
C   THIS ROUTINE OBTAINS A REVERSE CUTHILL-MCKEE ORDERING OF THE
C   VERTICES OF THE CONNECTED SYMMETRIC GRAPH IA.  IA IS A GRAPH
C   IN ADJACENCY LIST FORM, WITH IV(I) POINTING TO THE START OF THE
C   ADJACENCY LIST OF THE I-TH VERTEX.  ON RETURN, IORD(I) IS THE I-TH
C   VERTEX IN THE RCM ORDERING, AND IPOS(IORD(I)) = I.  N IS THE NUMBER
C   OF VERTICES, AND IV(N+1) POINTS TO THE FIRST UNUSED ENTRY OF IA.
C
C
C   INITIALIZATION
C
      DO 10 I=1,N
        IPOS(I) = 0
   10   CONTINUE
C
C   PICK A MINIMUM DEGREE STARTING VERTEX FOR CM
C
      IMD = 0
      MD = N + 1
      DO 20 I=1,N
        IF ((IV(I+1) - IV(I)) .GE. MD) GO TO 20
        MD = IV(I+1) - IV(I)
        IMD = I
   20   CONTINUE
C
C   STARTING VERTEX IS IMD WITH DEGREE MD.
C   PERFORM CM ORDERING AND REVERSE TO GET RCM ORDERING.
C   IPOS(I) .EQ. -D MEANS VERTEX I HAS BEEN ADDED TO QUEUE
C   WITH DEGREE D.  IPOS(I) = +D MEANS VERTEX I HAS BEEN
C   ORDERED AS VERTEX D IN RCM.
C   IORD(KF) IS FIRST VERTEX IN QUEUE.
C   IORD(KL) IS LAST VERTEX IN QUEUE.
C   N - K IS THE ORDERING NUMBER OF NEXT VERTEX IN RCM.
C
      IPOS(IMD) = N
      KL = 0
      KMAX = N - 1
      DO 80 K=1,KMAX
        IMIN = IV(IMD)
        IMAX = IV(IMD+1) - 1
        KN = KL + 1
C
C   ADD UNSCANNED NEIGHBORS OF IMD TO IORD QUEUE IN ORDER
C   OF INCREASING DEGREE (WITH AN INSERTION SORT)
C
        DO 70 I=IMIN,IMAX
          IAI = IA(I)
C
C   IPOS(IAI) .NE. 0 MEANS VERTEX IAI HAS BEEN SCANNED
C
          IF (IPOS(IAI) .NE. 0) GO TO 70
          IAID = IV(IAI) - IV(IAI+1)
C
C   SET IPOS(IAI) = - DEGREE(IAI) TO MARK IT SCANNED
C
          IPOS(IAI) = IAID
C
```

```
C    INSERT IAI IN QUEUE IN PROPER PLACE
C    (KL .LT. KN MEANS IAI IS THE FIRST TO BE ADDED)
C
           IF (KL .LT. KN) GO TO 50
           M = KL
           DO  30 J=KN,KL
             IORDJ = IORD(J)
             IF (IAID .GT. IPOS(IORDJ)) GO TO 40
   30        CONTINUE
C
C    PLACE IAI AT END OF QUEUE
C
           GO TO 50
C
C    MOVE VERTICES IN QUEUE TO MAKE ROOM FOR IAI
C
   40        IORD(M+1) = IORD(M)
             M = M - 1
             IF (M .GE. J) GO TO 40
           IORD(J) = IAI
           GO TO 60
   50      IORD(KL+1) = IAI
   60      KL = KL + 1
   70      CONTINUE
C
C    PICK NEXT VERTEX FROM FRONT OF QUEUE
C
         IMD = IORD(K)
         IPOS(IMD) = N - K
   80    CONTINUE
C
C    COMPUTE VALUES FOR IORD
C
       DO 90 I=1,N
         IPOSI = IPOS(I)
         IORD(IPOSI) = I
   90    CONTINUE
       RETURN
       END
```

```
      SUBROUTINE GENENV
     C  (N,MAXPL,PL,D,MAXPU,PU,IRL,IRU,IV,IA,A,IORD,IPOS,IFLAG)
      DIMENSION IA(1),IV(1),A(1),IORD(1),IPOS(1)
      DIMENSION PL(1),D(1),PU(1),IRL(1),IRU(1)
C
C   THIS SUBROUTINE TAKES AS INPUT A MATRIX C IN ADJACENCY LIST FORM
C   AND AN ORDERING IORD OF THE ROWS AND COLUMNS OF C CORRESPONDING TO A
C   PERMUTATION MATRIX P. IT PRODUCES AS OUTPUT THE ENVELOPE FORM OF THE
C   MATRIX PC = P C PT.  ENVELOPE FORM IS AS FOLLOWS:
C
C        PL    LIST OF ELEMENTS IN THE STRICT LOWER TRIANGLE OF THE
C                 ENVELOPE OF PC IN ROW MAJOR ORDER
C        PU    LIST OF ELEMENTS IN THE STRICT UPPER TRIANGLE OF THE
C                 ENVELOPE OF PC IN COLUMN MAJOR ORDER
C        D     D(I) = C(I,I)
C        IRL (IRU)  VECTOR OF POINTERS TO THE NONEXISTENT PCI0 (PC0I)
C                 ELEMENTS OF THE ROWS (COLUMNS) OF PL (PU)
C
C   ON INPUT, IA IS THE ADJACENCY LIST OF THE GRAPH OF A, IV(I)
C   POINTS TO THE START OF THE ADJACENCY LIST OF THE I-TH VERTEX,
C   AND A(I) IS THE REAL ENTRY CORRESPONDING TO IA(I).
C   MAXPL (MAXPU) IS THE MAXIMUM STORAGE AVAILABLE FOR PL (PU).
C   IFLAG IS USED TO RETURN ERROR INDICATIONS:
C       IFLAG = -1   MEANS NOT ENOUGH STORAGE FOR PL
C       IFLAG =  0   MEANS NO ERRORS ENCOUNTERED
C       IFLAG = +1   MEANS NOT ENOUGH STORAGE FOR PU
C
C
      DO 10 I=1,N
        IRL(I) = I
        IRU(I) = I
   10   CONTINUE
C
C   COMPUTE LOWEST OFF-DIAGONAL INDEX IN
C   EACH ROW OF PL (COLUMN OF PU)
C
      DO 40 I=1,N
        IORDI = IORD(I)
        KMIN = IV(IORDI)
        KMAX = IV(IORDI+1) - 1
        DO 40 K=KMIN,KMAX
          IAK = IA(K)
          IPIAK = IPOS(IAK)
C
C   IGNORE DIAGONAL ELEMENTS
C
          IF (IPIAK - I) 20,40,30
C
C   (I,IPIAK) WILL BE IN I-TH ROW OF PL
C
   20     IRL(I) = MIN0(IRL(I),IPIAK)
          GO TO 40
C
C   (I,IPIAK) WILL BE IN IPIAK-TH COLUMN OF PU
C
   30     IRU(IPIAK) = MIN0(IRU(IPIAK),I)
   40     CONTINUE
C
C   COMPUTE FINAL VALUES FOR IRL, IRU
C   AT THIS POINT IRL AND IRU CONTAIN THE LOWEST OFF-DIAGONAL
C   INDEX.  THE LOOP COMPUTES THE LOCATION OF THE NONEXISTENT
C   0-TH ELEMENT OF THE ROW OR COLUMN. A TEMPORARY (IRUI) IS
C   USED IN THE LOOP IN CASE IRU IS THE SAME VECTOR AS IRL IN
```

```
C   THE CALLING PROGRAM.
C
        IRL(1) = 0
        IRU(1) = 0
        DO 50 I=2,N
          IRUI = IRU(I)
          IRL(I) = I - 1 + IRL(I-1) - IRL(I)
          IRU(I) = I - 1 + IRU(I-1) - IRUI
   50     CONTINUE
C
C   INITIALIZE PL AND PU TO ZERO
C
      IMAX = IRL(N) + N - 1
      IF (IMAX .GT. MAXPL) GO TO 1001
      DO 60 I=1,IMAX
        PL(I) = 0
   60   CONTINUE
      IMAX = IRU(N) + N - 1
      IF (IMAX .GT. MAXPU) GO TO 2001
      DO 70 I=1,IMAX
        PU(I) = 0
   70   CONTINUE
C
C   GO THROUGH ADJACENCY STRUCTURE AND STORE MATRIX ELEMENTS
C
      DO 110 I=1,N
        IORDI = IORD(I)
        KMIN = IV(IORDI)
        KMAX = IV(IORDI+1) - 1
        IRLI = IRL(I)
        DO 110 K=KMIN,KMAX
          IAK = IA(K)
          IPIAK = IPOS(IAK)
          IF (IPIAK - I) 80,90,100
C
C   STORE ELEMENT (I,IPIAK) IN LOWER TRIANGLE
C
   80     IJ = IRLI + IPIAK
          PL(IJ) = A(K)
          GO TO 110
C
C   STORE DIAGONAL ELEMENT IN D
C
   90     D(I) = A(K)
          GO TO 110
C
C   STORE ELEMENT (I,IPIAK) IN UPPER TRIANGLE
C
  100     IJ = IRU(IPIAK) + I
          PU(IJ) = A(K)
  110   CONTINUE
C
      IFLAG = 0
      RETURN
C
C   ERROR RETURNS
C
1001  IFLAG = -1
      RETURN
2001  IFLAG = 1
      RETURN
C
      END
      SUBROUTINE PLU(N,PL,D,PU,IRL,IRU)
      DIMENSION PL(1),D(1),PU(1),IRL(1),IRU(1)
C
C   THIS SUBROUTINE PERFORMS A PROFILE L U DECOMPOSITION ON THE
C   MATRIX C WITH SYMMETRIC ZERO STRUCTURE WHICH IS STORED
C   IN PL, D, AND PU IN PROFILE FORM (SEE SUBROUTINE GENENV).
C   THE ROWS (COLUMNS) OF THE LOWER (UPPER) TRIANGLE OF A FROM THE FIRST
C   NONZERO UP TO, BUT NOT INCLUDING THE DIAGONAL, ARE STORED
C   SEQUENTIALLY IN PL (PU). THE DIAGONAL ENTRIES OF A ARE STORED IN D.
C   IRL(I) (IRU(I)) POINTS TO THE NONEXISTENT CI0 (C0J) ELEMENT OF THE
C   I-TH ROW (COLUMN). ON RETURN, THE STRICT LOWER (UPPER) TRIANGLE
C   OF L (U) IS STORED IN PL (PU), AND THE INVERSES OF THE DIAGONAL
C   ELEMENTS OF L ARE STORED IN D. (U IS UNIT UPPER TRIANGULAR.)
C
      D(1) = 1/D(1)
      DO 100 I=2,N
        IRLI = IRL(I)
        IRUI = IRU(I)
C
C   IFLI (IFUI) IS THE LOWEST OFF-DIAGONAL INDEX IN THE
C   I-TH ROW (COLUMN). SIMILAR COMPUTATIONS ARE USED FOR OTHER
C   ROWS AND COLUMNS BELOW.  THE FIRST OFF-DIAGONAL ELEMENT IN THE
C   I-TH ROW (COLUMN) NEVER REQUIRES AN INNER PRODUCT.
C
        IFLI = I - 1 + IRL(I-1) - IRLI
        JMINL = IFLI + 1
        IFUI = I - 1 + IRU(I-1) - IRUI
        JMINU = IFUI + 1
        JMAX = I - 1
C
C   COMPUTE L(I,J) FOR J IN I-TH ROW
C
        IF (JMINL .GE. I) GO TO 30
        DO 20 J=JMINL,JMAX
          IRUJ = IRU(J)
          IFUJ = J - 1 + IRU(J-1) - IRUJ
          KMIN = MAX0(IFLI,IFUJ)
          IF (KMIN .GE. J) GO TO 20
          IJ = IRLI + J
C
C   PLIJ = -PL(IJ) TO FORCE GOOD CODE IN LOOP
C
          PLIJ = -PL(IJ)
          KMAX = J - 1
C
C   COMPUTE INNER PRODUCT FOR L(I,J)
C
          DO 10 K=KMIN,KMAX
            IK = IRLI + K
            KJ = IRUJ + K
            PLIJ = PLIJ + PL(IK)*PU(KJ)
   10       CONTINUE
          PL(IJ) = -PLIJ
   20     CONTINUE
C
C   COMPUTE U(J,I) FOR J IN I-TH COLUMN
C
   30   IF (JMINU .GT. I) GO TO 70
C
C   COMPUTE FIRST OFF-DIAGONAL ELEMENT OF COLUMN
```

```
C
        JI = IRUI + JMINU - 1
        PU(JI) = PU(JI) * D(JMINU-1)
        IF (JMINU .EQ. I) GO TO 70
        DO 60 J=JMINU,JMAX
          IRLJ = IRL(J)
          IFLJ = J - 1 + IRL(J-1) - IRLJ
          KMIN = MAX0(IFUI,IFLJ)
          JI = IRUI + J
C
C   PUJI = -PU(JI) TO FORCE GOOD CODE IN LOOP
C
          PUJI = -PU(JI)
          IF (KMIN .GE. J) GO TO 50
          KMAX = J - 1
C
C   COMPUTE INNER PRODUCT FOR U(J,I)
C
          DO 40 K=KMIN,KMAX
            KI = IRUI + K
            JK = IRLJ + K
            PUJI = PUJI + PL(JK)*PU(KI)
   40       CONTINUE
   50     PU(JI) = -PUJI * D(J)
   60     CONTINUE
C
C   COMPUTE L(I,I)
C
   70   JMIN = MAX0(IFLI,IFUI)
        DI = -D(I)
        IF (JMIN .GT. JMAX) GO TO 90
        DO 80 J=JMIN,JMAX
          IJ = IRLI + J
          JI = IRUI + J
          DI = DI + PL(IJ)*PU(JI)
   80     CONTINUE
C
C   STORE 1/L(I,I) IN D(I)
C
   90   D(I) = -1/DI
  100   CONTINUE
      RETURN
      END
      SUBROUTINE PLDLT(N,PL,D,IRL)
      DIMENSION PL(1),D(1),IRL(1)
C
C   THIS SUBROUTINE PERFORMS A PROFILE L D LT DECOMPOSITION ON THE
C   MATRIX C STORED IN PL AND D.  THE ROWS OF PL FROM THE FIRST NONZERO
C   UP TO, BUT NOT INCLUDING THE DIAGONAL, ARE STORED SEQUENTIALLY
C   IN PL.  THE DIAGONAL OF C IS STORED IN D.  IRL(I) POINTS TO
C   THE NONEXISTENT CI0 ELEMENT OF THE I-TH ROW. ON RETURN, THE STRICT
C   LOWER TRIANGLE OF L IS STORED IN PL, AND THE INVERSE OF D IS
C   STORED IN D.
C
      D(1) = 1/D(1)
      DO 60 I=2,N
        IRLI = IRL(I)
C
C   IFLI IS THE LOWEST OFF-DIAGONAL INDEX IN THE
C   I-TH ROW. SIMILAR COMPUTATIONS ARE USED
C   FOR OTHER ROWS BELOW. THE FIRST OFF-DIAGONAL ELEMENT
C   REQUIRES NO INNER PRODUCTS.
C
        IFLI = I - 1 + IRL(I-1) - IRLI
        JMIN = IFLI + 1
        JMAX = I - 1
C
C     COMPUTE A'(I,J) = L(I,J)*D(J,J) FOR J IN I-TH ROW
C
        IF (JMIN .GE. I) GO TO 30
        DO 20 J=JMIN,JMAX
          IRLJ = IRL(J)
          IFLJ = J - 1 + IRL(J-1) - IRLJ
          KMIN = MAX0(IFLI,IFLJ)
          IF (KMIN .GE. J) GO TO 20
          IJ = IRLI + J
C
C     PLIJ = -PL(IJ) TO FORCE GOOD CODE GENERATION IN LOOP
C
          PLIJ = -PL(IJ)
                  KMAX = J - 1
        C
        C     COMPUTE INNER PRODUCT FOR A'(I,J)
        C
                  DO 10 K=KMIN,KMAX
                    IK = IRLI + K
                    JK = IRLJ + K
                    PLIJ = PLIJ + PL(IK)*PL(JK)
           10       CONTINUE
          PL(IJ) = -PLIJ
   20     CONTINUE
C
C     COMPUTE L(I,J) = A'(I,J)/D(J,J) AND D(I,I)
C
   30   DI = -D(I)
        IF (IFLI .GE. I) GO TO 50
        DO 40 J=IFLI,JMAX
          IJ = IRLI + J
          PLIJ = PL(IJ)
          PL(IJ) = PLIJ * D(J)
          DI = DI + PLIJ * PL(IJ)
   40     CONTINUE
   50   D(I) = -1/DI
   60   CONTINUE
      RETURN
      END
      SUBROUTINE PLUB(N,PL,D,PU,IRL,IRU,X,B,IORD)
      DIMENSION PL(1),D(1),PU(1),IRL(1),IRU(1)
      DIMENSION X(1),B(1),IORD(1)
C
C   THIS SUBROUTINE PERFORMS THE BACKSOLVES FOR THE SOLUTION OF
C   L U P X = P B.  L AND U ARE STORED IN PL, D, AND PU AS
C   DESCRIBED IN SUBROUTINE PLU.
C
C
C   SOLVE L X = P B
C
      IORDJ = IORD(1)
      X(1) = B(IORDJ) * D(1)
      DO 30 J=2,N
        IORDJ = IORD(J)
        XJ = -B(IORDJ)
        IRLJ = IRL(J)
C
C   KMIN IS THE LOWEST OFF-DIAGONAL INDEX IN J-TH ROW OF PL.
C   SIMILAR COMPUTATIONS ARE USED FOR OTHER ROWS AND COLUMNS BELOW
C
        KMIN = J - 1 + IRL(J-1) - IRLJ
        IF (KMIN .GE. J) GO TO 20
        KMAX = J - 1
        DO 10 K=KMIN,KMAX
          JK = IRLJ + K
          XJ = XJ + PL(JK)*X(K)
   10     CONTINUE
   20   X(J) = - XJ * D(J)
   30   CONTINUE
C
C   SOLVE U X = X
C
      IMAX = N - 1
      DO 50 I=1,IMAX
        J = N + 1 - I
        IRUJ = IRU(J)
        KMIN = J - 1 + IRU(J-1) - IRUJ
        IF (KMIN .GE. J) GO TO 50
        KMAX = J - 1
        XJ = -X(J)
        DO 40 K=KMIN,KMAX
          JK = IRUJ + K
          X(K) = X(K) + XJ * PU(JK)
   40     CONTINUE
   50   CONTINUE
C
C   REORDER X TO SOLVE P X = X
C
      DO 70 I=1,N
        K = I
C
C   IORD(I) .LT. 0 MEANS THAT X(I) IS PROPER ELEMENT ALREADY.
C   OTHERWISE, INTERCHANGE X(K) AND X(IORD(I)).  THE EFFECT
C   OF THIS IS TO ROTATE EVERY CYCLE OF THE PERMUTATION ONE
C   POSITION SO THAT IT IS PROPERLY ORIENTED.
C
        IF (IORD(I) .LT. 0) GO TO 70
   60     IORDI = IORD(I)
          T = X(IORDI)
          X(IORDI) = X(K)
          X(K) = T
          IORD(I) = -IORD(I)
          I = IORDI
          IF (I .NE. K) GO TO 60
   70   CONTINUE
C
C   AT THIS POINT, ALL ENTRIES OF IORD HAVE BEEN NEGATED
C
        DO 80 I=1,N
          IORD(I) = -IORD(I)
   80     CONTINUE
      RETURN
      END
      SUBROUTINE PLDLTB(N,PL,D,IRL,X,B,IORD)
      DIMENSION PL(1),D(1),IRL(1),X(1),B(1),IORD(1)
C
C   THIS SUBROUTINE PERFORMS THE BACKSOLVES FOR THE SOLUTION OF
C   L D LT P X = P B.  L AND THE INVERSE OF D ARE STORED IN PL
C   AS DESCRIBED IN SUBROUTINE PLDLT.
C
C
C   SOLVE L X = P B
C
      IORDJ = IORD(1)
      X(1) = B(IORDJ)
      DO 30 J=2,N
        IORDJ = IORD(J)
        XJ = -B(IORDJ)
        IRLJ = IRL(J)
C
C   KMIN IS THE LOWEST OFF-DIAGONAL INDEX IN J-TH ROW OF PL.
C   SIMILAR COMPUTATIONS ARE USED FOR OTHER ROWS BELOW
C
        KMIN = J - 1 + IRL(J-1) - IRLJ
        IF (KMIN .GE. J) GO TO 20
        KMAX = J - 1
        DO 10 K=KMIN,KMAX
          JK = IRLJ + K
          XJ = XJ + PL(JK)*X(K)
   10     CONTINUE
   20   X(J) = -XJ
   30   CONTINUE
C
C   SOLVE D X = X
C
      DO 40 I=1,N
        X(I) = X(I) * D(I)
   40   CONTINUE
C
C   SOLVE LT X = X
C
      IMAX = N - 1
      DO 60 I=1,IMAX
        J = N + 1 - I
        IRLJ = IRL(J)
        KMIN = J - 1 + IRL(J-1) - IRLJ
        IF (KMIN .GE. J) GO TO 60
        KMAX = J - 1
        XJ = -X(J)
        DO 50 K=KMIN,KMAX
          JK = IRLJ + K
          X(K) = X(K) + XJ * PL(JK)
   50     CONTINUE
   60   CONTINUE
C
C   REORDER X TO SOLVE P X = X
C
      DO 80 I=1,N
        K = I
C
C   IORD(I) .LT. 0 MEANS THAT X(I) IS PROPER ELEMENT ALREADY.
C   OTHERWISE, INTERCHANGE X(K) AND X(IORD(I)).  THE EFFECT
C   OF THIS IS TO ROTATE EVERY CYCLE OF THE PERMUTATION ONE
```

```
C   POSITION SO THAT IT IS PROPERLY ORIENTED.
C
        IF (IORD(I) .LT. 0) GO TO 80
   70     IORDI = IORD(I)
          T = X(IORDI)
          X(IORDI) = X(K)
          X(K) = T
          IORD(I) = -IORDI
          I = IORDI
          IF (I .NE. K) GO TO 70
   80   CONTINUE
      DO 90 I=1,N
        IORD(I) = -IORD(I)
   90   CONTINUE
      RETURN
      END
```

Appendix B

This appendix contains a driver program which demonstrates the proper calling sequences for the subroutines presented in Appendix A. The program solves the system (1.1) where A is a block tridiagonal matrix arising in the solution of the Poisson equation over the unit square. In the actual example given (corresponding to Figure 3.4),

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \qquad (B.1)$$

and the right hand side $\underset{\sim}{b}$ is computed so that the exact solution $\underset{\sim}{x}$ is

$$\underset{\sim}{x} = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]^t\ . \qquad (B.2)$$

The first section of the driver program generates the matrix of coefficients and calls RCM to obtain the Reverse Cuthill-McKee ordering of the system. The system is then solved three times to illustrate the different procedures for systems which are symmetric, nonsymmetric with symmetric zero structure, and fully nonsymmetric.

For a symmetric system, the strict lower triangle and the strict upper triangle of the envelope of A are identical (i.e. PU = PL and IRU = IRL). Hence only one of them needs to be computed and stored, and GENENV is called with MAXPU = MAXPL, PU = PL, and IRU = IRL. The factorization of A and the backsolution are performed using PLDLT and PLDLTB, respectively.

For a nonsymmetric system with symmetric zero structure, the strict lower triangle and the strict upper triangle of the envelope of A have identical structure (i.e. IRU = IRL). Hence GENENV is called with MAXPU = MAXPL and IRU = IRL. Since the zero structure of U is the transpose of the zero structure of L, the factorization of A and the backsolution are performed by calling PLU and PLUB, respectively, with IRU = IRL.

Finally, for a fully nonsymmetric system, the strict lower triangle and the strict upper triangle of the envelope of A are entirely different, so GENENV is called with no replication of variables. The factorization of A and the backsolution are performed using PLU and PLUB, respectively. Note that RCM is not designed for use with systems having nonsymmetric zero structure and that the results of using it on such systems are unpredictable.

```
      DIMENSION IORD(9),IPOS(9)
      DIMENSION PL(20),PU(20),IRL(9),IRU(9),D(9)
      DIMENSION IA(50),IV(10),A(50),X(9),B(9)
C
      DATA M/3/,MAXPL/20/,MAXPU/20/
C
      INDEX(I,J) = M * I + J - M
C
C
      N = M * M
      WRITE (6,1) N
    1 FORMAT(23H NUMBER OF EQUATIONS:  ,I3)
C
C     FORM COEFFICIENT MATRIX IN ADJACENCY LIST FORMAT
C
      IAPTR = 1
      DO 20 I=1,M
        DO 20 J=1,M
          IVP = INDEX(I,J)
          IV(IVP) = IAPTR
          KMIN = MAX0(1,I-1)
          KMAX = MIN0(M,I+1)
          LMIN = MAX0(1,J-1)
          LMAX = MIN0(M,J+1)
          DO 10 K=KMIN,KMAX
            DO 10 L=LMIN,LMAX
              IF (((K-I) * (L-J)) .NE. 0) GO TO 10
              IVQ = INDEX(K,L)
              IA(IAPTR) = IVQ
              A(IAPTR) = -1
              IF (IVP .EQ. IVQ) A(IAPTR) = 4
              IAPTR = IAPTR + 1
   10         CONTINUE
   20     CONTINUE
      IV(N+1) = IAPTR
C
C     COMPUTE RCM ORDERING FROM ADJACENCY STRUCTURE
C
      CALL RCM(N,IV,IA,IORD,IPOS)
C
C     PUT SYMMETRIC COEFFICIENT MATRIX IN ENVELOPE FORM,
C     COMPUTE RIGHT HAND SIDE B, AND SOLVE.
C
      CALL GENENV
     C    (N,MAXPL,PL,D,MAXPL,PL,IRL,IRL,IV,IA,A,IORD,IPOS,IFLAG)
      IF (IFLAG) 1001,30,2001
   30 CALL GENB(N,IV,IA,A,B)
      CALL PLDLT(N,PL,D,IRL)
      CALL PLDLTB(N,PL,D,IRL,X,B,IORD)
C
C     COMPUTE NORM OF ERROR IN SOLUTION
C
      Z = 0
      DO 40 I=1,N
        Z = Z + (X(I)-I)**2
   40   CONTINUE
      Z = SQRT(Z)
      WRITE (6,2)
      WRITE (6,3) Z
      WRITE (6,4) (I,X(I),I=1,N)
```