

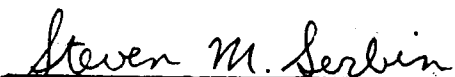
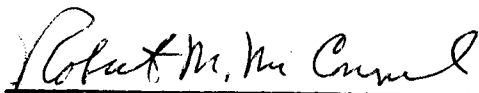
To the Graduate Council:

I am submitting herewith a thesis written by James F. Litsey entitled "Finding a Block Upper Trapezoidal Form of a Rectangular Matrix." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.



Robert J. Plemmons, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Vice Chancellor

Graduate Studies and Research

FINDING A BLOCK UPPER TRAPEZOIDAL
FORM OF A RECTANGULAR MATRIX

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

James F. Litsey

December 1980

3046695

ACKNOWLEDGEMENTS

The author wishes to express his appreciation for the helpful and humane criticism and guidance provided by his advisor, Dr. Robert J. Plemmons. The author is also greatly indebted to his typist, Sarah E. Kelley, for her patience and expertise exhibited during the preparation of this thesis. Finally, the author would like to acknowledge one of the principal molders of his mathematical perspective, Dr. David Hume.

To my mother

ABSTRACT

An algorithm is presented whereby permutation matrices P and Q are found such that a rectangular matrix A of dimensions $m \times n$, $m \geq n$, is permuted to block upper trapezoidal form as shown next

$$PAQ = \begin{bmatrix} A_{11} & A_{12} & \cdot & \cdot & \cdot & A_{1k} \\ 0 & A_{22} & \cdot & \cdot & \cdot & A_{2k} \\ 0 & 0 & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & A_{kk} \end{bmatrix},$$

where the A_{ii} have dimensions $m_i \times n_i$, $m_i \geq n_i$ for each $1 \leq i \leq k$. This algorithm extends the work of I. S. Duff and J. K. Reid who have provided algorithms and codes in the Harwell sparse matrix package MA28, for block triangularizing square matrices.

A special case of the block upper trapezoidal form, called the dual angular form, is discussed and an algorithm presented to find a permutation matrix which may be incorporated into the equation above to yield a permutation of A to dual angular form as shown next

$$P'AQ' = \begin{bmatrix} A_{11} & 0 & 0 & \cdots & 0 & A_{1 \ k+1} \\ 0 & A_{22} & 0 & \cdots & 0 & A_{2 \ k+1} \\ 0 & 0 & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & 0 & \cdot \\ 0 & 0 & & & A_{kk} & A_{k \ k+1} \end{bmatrix},$$

where the A_{ii} have dimensions $m_i \times n_i$, $m_i \geq n_i$, for each $1 \leq i \leq k$. Each of these two forms facilitate the block orthogonal decomposition of sparse matrices involved in least squares computations.

The first two appendices contain examples showing permutations of sparse rectangular matrices to block upper trapezoidal form. FORTRAN codes of the main program and of subroutines used to implement these algorithms are given in the remaining appendices.

TABLE OF CONTENTS

CHAPTER	PAGE
INTRODUCTION	1
1. THE SQUARE CASE	3
2. PERMUTING A RECTANGULAR MATRIX TO BLOCK UPPER TRAPEZOIDAL FORM	14
Selecting an A_1	19
Selecting an A'_1	20
Block Triangularizing A'_1	24
Row Permutation to Final Form	24
Uniqueness of Final Block Structure	25
3. DUAL ANGULAR FORM	26
4. SUMMARY AND COMMENTS	34
LIST OF REFERENCES	39
APPENDICES	41
APPENDIX 1. EXAMPLE A	42
APPENDIX 2. EXAMPLE B	54
APPENDIX 3. THE MODIFIED MC21A SUBROUTINE	65
APPENDIX 4. THE MIN SUBROUTINE	70
APPENDIX 5. THE MODIFIED MC13D SUBROUTINE	75
APPENDIX 6. THE PRMBLK SUBROUTINE	79
APPENDIX 7. THE ABSORB SUBROUTINE	84
APPENDIX 8. PRINT AND LPRINT SUBROUTINES	87
APPENDIX 9. MAIN PROGRAM	89
VITA	92

INTRODUCTION

The primary purpose of this thesis is to develop and discuss an algorithm for permuting a rectangular matrix A of dimensions $m \times n$, $m \geq n$, and column rank n , into a block upper trapezoidal form given by

$$PAQ = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ 0 & A_{22} & \cdots & A_{2k} \\ 0 & 0 & \ddots & \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & A_{kk} \end{bmatrix}.$$

In the above equation P and Q represent permutation matrices, while the A_{ii} , $1 \leq i \leq k$, are rectangular matrices of dimensions $m_i \times n_i$, with $m_i \geq n_i$.

The square case, where $m = n$ and $m_i = n_i$ for $1 \leq i \leq k$, is examined in Chapter 1. An algorithm developed by Duff [1979] for permuting a matrix to obtain a zero-free main diagonal is presented. Also an algorithm by Duff and Reid [1978] for permuting a matrix with a zero-free main diagonal to block upper triangular form is presented.

The rectangular case, where $m > n$ and $m_i \geq n_i$ for $1 \leq i \leq k$ is examined in Chapter 2, where the algorithms presented in Chapter 1 are modified to accomodate a rectangular matrix. An algorithm, MIN, is provided to select the rows with fewest nonzero entries when obtaining the zero-free diagonal mentioned above. A heuristic justification is given for this algorithm. Also an algorithm ABSORB is

presented. This algorithm incorporates the rows not included in the intermediate block upper triangular structure to effect a final block upper trapezoidal structure. Following the presentation of this algorithm is a brief discussion on uniqueness.

Chapter 3 deals with a special case of the block upper trapezoidal final form, called the "dual angular" form. A heuristic algorithm by Weil and Kettler [1971] for permuting a matrix A of dimensions $m \times n$, with $m < n$ to a similar form called "block angular" is presented. Problems with the implementation of such an algorithm are discussed. Finally, an algorithm PRMBLK is presented. PRMBLK permutes the blocks of a matrix in block upper trapezoidal form to possibly achieve a dual angular form. Limitations of PRMBLK are discussed.

Chapter 4 summarizes the thesis and presents a number of related topics for future investigation. Appendix 1 and Appendix 2 contain some examples of dimension 75×50 . Appendix 3 contains a listing of Harwell subroutine MC21A with slight modifications. Appendix 4 contains a flowchart and computer listing of subroutine MIN. Appendix 5 contains a computer listing of a slightly modified Harwell subroutine MC13D. Appendix 6 contains a flowchart and computer listing of subroutine ABSORB. A flowchart and computer listing for PRMBLK is found in Appendix 7. Appendix 8 contains the computer listing of the PRINT and optional LPRINT subroutines, and finally, the main program is listed in Appendix 9.

CHAPTER 1

THE SQUARE CASE

The problem of permuting a square matrix A to block upper triangular form with square, indecomposable, diagonal blocks is reviewed in this chapter. Algorithms for determining certain permutation matrices R and Q such that $Q^t R A Q$ is block upper triangular have already been developed [Duff and Reid, 1979].

The advantages of expressing a square sparse matrix A in a block upper triangular form have to do mainly with the following facts:

1. The eigenvalues of A are those of the diagonal blocks A_{ii} and hence those of A can be computed more easily.
2. When solving $Ax = b$ with A block upper triangular by Gaussian Elimination, the elimination process can be restricted to the diagonal blocks.
3. A matrix in block upper triangular form can be stored by blocks. Depending upon the sizes of the diagonal blocks, storage of only the nonzero upper blocks can reduce storage requirements by nearly one-half in many computational problems.

Advantages also exist for permuting rectangular matrices to block upper trapezoidal form. These advantages involve least squares computations and will be described later.

The following definitions will facilitate the remaining discussion. A matrix A is said to be decomposable if there exist permutation matrices P and Q such that

$$PAQ = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} and A_{22} are square. Otherwise A is indecomposable. A matrix A is said to be reducible if there exists a permutation matrix Q such that

$$Q^t A Q = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} and A_{22} are square. Otherwise A is irreducible. Thus a decomposable matrix can be asymmetrically permuted to block upper triangular form, while a reducible matrix can be symmetrically permuted to such a form. Indecomposability implies irreducibility, but the converse of this statement is not true.

A transversal of a matrix is a set of nonzero elements of the matrix, no two of which are on the same row or column. The length of a transversal refers to the number of elements included in it. A maximal transversal of a matrix is a transversal of greatest length. Of course a matrix can have more than one maximal transversal. A matrix A with full column rank n must have at least one transversal of length n . This fact can be shown in the following way. Suppose A has full column rank n and has a maximal transversal of length $k < n$. Then there exist $n - k$ columns of A that do not include a transversal element. If the rows and columns of A are now permuted by P and Q so that

$$PAQ = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11} is a $k \times k$ matrix with main diagonal consisting of the k transversal elements, then the $n - k$ columns of $\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$ do not contain any transversal elements. It follows from the assumption that the maximal transversal has length k , that all entries of the $(n - k) \times (n - k)$ matrix A_{22} are zero. Now form the set V of the k columns of $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$. There exists a subset S of V consisting of columns which contain the transversal element in a row in which one of the columns of $\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$ has a nonzero entry. The columns in S contain no nonzeros below their k^{th} entry, otherwise the assumption that the maximum transversal is of length k is violated. The columns of S are linearly independent and form a basis for a vector space of which the columns of $\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$ are vectors. Therefore the columns of PAQ are not linearly independent and A cannot have column rank n . A contradiction has been reached. Thus every matrix A of full column rank n must have a transversal of length n . It should be noted that this transversal is generally not unique, and also that a matrix of column rank less than n may still possess a transversal of length n . Once a transversal is selected, row permutations are sufficient to bring the transversal elements to the main diagonal.

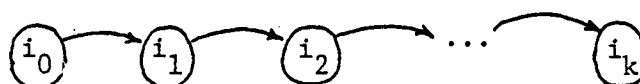
A simple proof of the following well-known theorem can be found in Duff and Reid [1978].

Theorem 1. If a square matrix A has a zero-free main diagonal, then A is irreducible if and only if it is indecomposable.

Thus given that a square matrix in block upper triangular form has a zero-free main diagonal, irreducible diagonal blocks are also indecomposable. The usefulness of Theorem 1 will become apparent momentarily.

The Duff-Reid algorithms for transversal selection and block triangularization of square matrices are briefly summarized below. A more complete description can be found in Duff [1978] and Duff and Reid [1978].

The Duff approach to the selection of a transversal consists of two phases. In the cheap assignment phase, each row is examined and within that row the locations of the nonzeros are determined. As soon as a nonzero is found in a column with no transversal element assigned to it, that nonzero is included in the transversal, and the next row is examined. After completion of the cheap assignment phase, the transversal is generally not of maximum length. The task of selecting a maximal transversal falls to the second phase of the algorithm, the depth first search. In the depth first search phase, each row i_0 not already containing a transversal element is examined and a graph theoretical reassignment chain is constructed as follows. The chain of row indices



implies that element $a_{i_1 j_1}$ is currently a transversal element and $a_{i_0 j_1}$ is nonzero; $a_{i_2 j_2}$ is currently a transversal element and $a_{i_1 j_2}$ is nonzero; etc. If such a chain can be formed such that there exists a nonzero $a_{i_k j_0}$ which is in a column j_0 not currently containing a transversal element, a reassignment is performed. This means that the elements $\{a_{i_1 j_1}, a_{i_2 j_2}, \dots, a_{i_k j_k}\}$ belonging to the previous transversal are deleted from the transversal and the elements $\{a_{i_0 j_1}, a_{i_1 j_2}, \dots, a_{i_k j_0}\}$ replace them, thereby increasing the length of the transversal by one. For a matrix of full column rank n , sufficient repetitions of this scheme will produce a transversal of length n . Finally a row permutation is performed, bringing the transversal entries to the main diagonal.

In the example given by (1.1), the circled elements illustrate a transversal of length eight which might be selected during the cheap assignment phase.

$$A = \begin{bmatrix} 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \textcircled{1} & 1 \\ 0 & 0 & 0 & \textcircled{1} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.1)$$

During the depth first search phase the reassignment chain $\textcircled{8} \rightarrow \textcircled{6}$ would be constructed and a reassignment performed, yielding the maximal transversal of length nine depicted in (1.2).

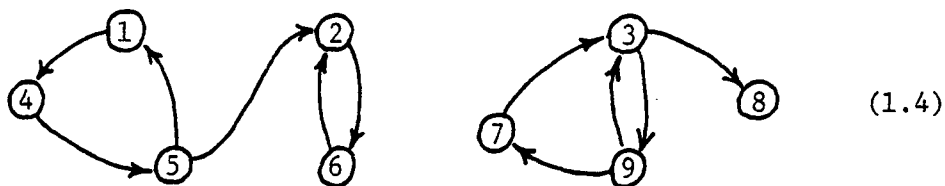
$$A = \begin{bmatrix} 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 \\ \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \textcircled{1} \\ 0 & 0 & 0 & \textcircled{1} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 \\ 1 & 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.2)$$

Finally, a row permutation matrix R brings the transversal elements to the main diagonal in (1.3).

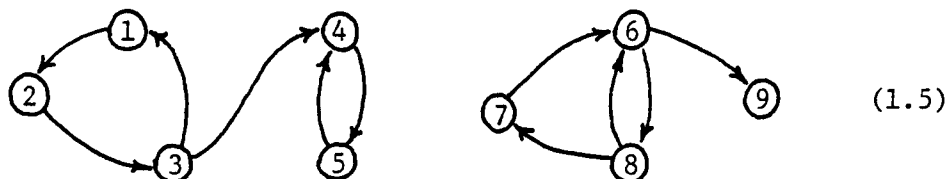
$$RA = \begin{bmatrix} \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \textcircled{1} & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \textcircled{1} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \textcircled{1} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & \textcircled{1} \end{bmatrix} \quad (1.3)$$

The Duff and Reid [1978] implementation of the Tarjan algorithm for permuting a matrix to block lower triangular form is easily adapted to the task of permuting a matrix to block upper triangular form. The algorithm is best applied to matrices already permuted so that the main diagonal is zero-free. For such matrices, Theorem 1 implies that a symmetric (rather than an asymmetric) permutation will be sufficient to put the matrix in block triangular form with indecomposable diagonal blocks. To construct the permutation matrix, a directed graph of the matrix A is formed with one vertex of the digraph corresponding to each row of the matrix. Edges of the directed graph are defined in the following way. The edge  means $a_{ij} \neq 0$, so that a one-to-one correspondence exists between off-diagonal nonzeros of the matrix and edges of the graph. A set of vertices any one of which may be reached from any other in the set by travelling along a set of edges is said to be a connected component. If no other vertex may be added to a component without destroying its connectedness, the set is said to be a maximal connected component. Every directed graph has a unique set of maximal connected components and once these are found for the directed graph under consideration, a symmetric permutation is formed to relabel the vertices such that all vertices within a maximal connected component are labelled consecutively and such that if there exists an edge from a vertex in component i to a vertex in component j , then all vertices in component i are labelled before all those in component j .

In the following example, the directed graph of the matrix RA in (1.3) is presented in (1.4).



These vertices are relabelled in (1.5).



This relabelling leads to the row permutation matrix Q^t given in (1.6).

$$Q^t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.6)$$

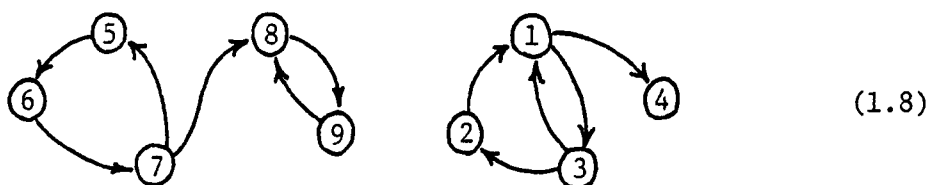
Finally the symmetric permutation is performed, leading to the block upper triangular form given by (1.7).

$$Q^t R A Q = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.7)$$

To give a block lower triangular form, the order of the components would be reversed when performing the relabelling in (1.5). The original algorithm by Duff and Reid gives a block lower triangular form.

Although Theorem 1 implies that the diagonal blocks are already indecomposable and thus no permutation of the final form $Q^t R A Q$ given in (1.7) would result in smaller diagonal blocks, this does not mean that the final form is unique. This can be shown using the previous example. There are two different methods by which alternative final matrices can be produced. First, the consecutive labels within each maximal connected component can be permuted. This will result in a symmetric permutation within the diagonal blocks of $Q^t R A Q$, but will not affect the block structure. Secondly there may sometimes be several ways to satisfactorily choose the order of the components themselves in the ordering. Changing this order would change the order of the blocks on the diagonal from that shown in (1.7).

Shown below is an alternate relabelling of the vertices.



The resulting final form is shown in (1.9). The permutation matrix Q_1 used to effect this form would, of course, be different from the matrix Q in (1.7).

$$Q_1^t R A Q_1 = \begin{bmatrix} \boxed{\begin{matrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}} & \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} & \boxed{1} & \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} & \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} & \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \end{bmatrix} \quad (1.9)$$

Thus, although the final block upper triangular form is not unique, it is unique up to symmetric permutations within diagonal blocks and order the diagonal blocks along the diagonal. Another important fact is that the final form is independent of which maximum transversal is selected. This is because if rows (or columns) i and j can be interchanged in RA and a zero-free diagonal maintained, then vertices i and j will be in the same connected component and all four elements a_{ii} , a_{ij} , a_{ji} , a_{jj} will end up in one of the

diagonal blocks regardless of which transversal is utilized. The scheme discussed here for permuting a square matrix to block upper triangular form is modified in the next chapter to produce a scheme for permuting rectangular matrices to block upper trapezoidal form.

CHAPTER 2

PERMUTING A RECTANGULAR MATRIX TO BLOCK UPPER TRAPEZOIDAL FORM

In this chapter an algorithm is given which permutes a matrix with more rows than columns to block upper trapezoidal form. One area where such a permutation is desirable is in solving the linear least squares problem $\min_x \|Ax - b\|_2$, where A is m by n , $m \geq n$ and is assumed to have full column rank. The classical approach to solving this problem is by forming and solving the system of normal equations $A^t Ax = A^t b$. The matrix $A^t A$, being symmetric and positive definite, can be factored using Cholesky's method into $R^t R$, where R is upper triangular. The solution x is then computed by solving the two triangular systems $R^t y = b$ and $Rx = y$.

The following features make the normal equations approach attractive (see George and Heath [1980]);

1. No pivoting is required for numerical stability during the Cholesky factorization, therefore the columns of A can be

ordered so as to promote sparsity in $A^t A$. There exists well developed software for exploiting sparsity in such systems. A "symbolic factorization" can be performed in advance of the actual factorization, allowing use of a static data structure.

2. The row order of A is irrelevant. Therefore the rows of A may be processed sequentially from an auxiliary input file in arbitrary order, with only one in fast storage at any given time.
3. The explicit computation of the Cholesky factor R provides access to the matrix $(A^t A)^{-1} = (R^t R)^{-1}$, called the variance-covariance matrix, which contains valuable statistical information.

The normal equations approach also has the following unattractive aspects, however:

1. Unless extended precision is employed, a serious loss of information may result from forming and processing the normal equations.
2. The condition number of $A^t A$ is the square of the condition number of A , so the normal equations may be quite ill-conditioned.
3. Sparsity in A does not imply sparsity in $A^t A$.

The best known stable alternative to the normal equations is orthogonal factorization. An orthogonal matrix Q , usually either a Gram-Schmidt orthogonalization matrix or a product of Householder or Givens transformations, is applied to A and b resulting in

$$QA = \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where R is upper triangular, and

$$Qb = \begin{bmatrix} y \\ z \end{bmatrix}.$$

Since the application of an orthogonal matrix does not change the two-norm,

$$\|Ax - b\|_2 = \|QAx - Qb\|_2 = \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} x - \begin{bmatrix} y \\ z \end{bmatrix} \right\|,$$

and the x minimizing $\left\| \begin{bmatrix} R \\ 0 \end{bmatrix} x - \begin{bmatrix} y \\ z \end{bmatrix} \right\|$ may be found by solving the triangular system $Rx = y$.

The fact that the use of Householder transformations or Gram-Schmidt orthogonalization can cause severe intermediate fill-in has led to some reluctance in using these methods of orthogonal factorization for solution of sparse linear least squares problems. However, George, Heath and Plemmons [1980] have developed a method based upon Givens rotations whereby a sparse least squares problem is solved by first permuting the matrix A to block upper trapezoidal form, shown below.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdot & \cdot & \cdot & A_{1k} \\ 0 & A_{22} & \cdot & \cdot & \cdot & A_{2k} \\ 0 & 0 & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & A_{kk} \end{bmatrix} \quad (2.1)$$

Here each A_{ii} has dimension $m_i \times n_i$ with $m_i \geq n_i$. The matrix is orthogonally factored to $A = QR$ in the following manner:

1. The matrix A_1 consisting of A_{11} and all non-null columns of A_{1i} , $2 \leq i \leq k$, is formed.
2. A_1 is orthogonally factored using Givens rotations into

$$A_1 = Q_1 R_1 = Q_1 \left[\begin{array}{c|c} \text{shaded triangle} & \text{shaded triangle} \\ \hline & \text{shaded triangle} \end{array} \right] = Q_1 \begin{bmatrix} R_{11} \\ 0 & R_{12} \end{bmatrix}$$

The first n_1 rows of R_1 are labelled R_{11} and form the first n_1 rows of R . The remaining upper triangular submatrix in the "southeast" corner of R_1 is labelled R_{12} .

3. Each column in R_{12} originally came from one of the submatrices A_{1i} , $2 \leq i \leq k$. Each is now appended to the top of the corresponding column of A_{2i} , $2 \leq i \leq k$.
4. A_2 is formed from A_{22} together with the non-null columns from A_{2i} , $3 \leq i \leq k$, and the process is repeated, gradually forming the matrix R .

This scheme of first permuting A to block upper trapezoidal form and then orthogonally decomposing A by the above technique can permit a savings in work as well as permitting A to be stored by blocks, most of which can be in auxiliary storage at any given time during the QR decomposition. The method used by George, Heath, and Plemmons for permuting A to block upper trapezoidal form is called

incomplete nested dissection partitioning which is described more fully in George, Heath, and Plemmons [1980].

This thesis presents an alternative algorithm for permuting a matrix A of dimension $m \times n$, $m \geq n$ with full column rank, to block upper trapezoidal form. The algorithm consists of four main steps. The first and third steps are modifications of the two steps of the algorithm given in Chapter 1 for permuting square matrices to block upper triangular form. The four steps are as follows:

1. A transversal of length n is chosen and a permutation matrix R formed so that

$$RA = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix},$$

where A_1 is a square $n \times n$ matrix with a zero-free main diagonal.

2. In general, matrix A can have more than one transversal of length n . This implies that the matrix A_1 chosen above is not unique. In this step the rows of RA are permuted so that A_1 is replaced by A'_1 which contains as few nonzeros as possible and still maintains a zero-free main diagonal. At this point in the algorithm

$$R'(RA) = R' \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} A'_1 \\ A'_2 \end{bmatrix},$$

where A'_1 consists of the n rows of the original matrix

A which contain the fewest number of nonzeros and can still be permuted to yield a zero-free diagonal.

3. In this step a permutation matrix Q is formed to symmetrically permute the square matrix A'_1 to block upper triangular form,

$$\begin{bmatrix} Q^t & 0 \\ 0 & I_{m-n} \end{bmatrix} R'RAQ = \begin{bmatrix} A_1^* \\ A_2^* \end{bmatrix},$$

where A_1^* is block upper triangular with indecomposable diagonal blocks.

4. Finally, a block upper trapezoidal form is achieved by forming a permutation matrix P such that

$$P \begin{bmatrix} Q^t & 0 \\ 0 & I_{m-n} \end{bmatrix} R'RAQ = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ 0 & A_{22} & \cdots & A_{2p} \\ 0 & 0 & & \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{pp} \end{bmatrix},$$

where each A_{ii} is a rectangular block of dimension $m_i \times n_i$, with $m_i \geq n_i$.

Each step is now examined in more detail.

Selecting an A_1

Selection of a transversal for A is achieved by slightly modifying the Harwell subroutine MC21A (Duff [1978]) so that all m

rows may be scanned during the cheap assignment and depth first search phases. Selecting a transversal of length n is generally faster for an $m \times n$ matrix of rank n with $m > n$ than for an $n \times n$ matrix of rank n ; for, since there are more rows to scan during the cheap assignment phase, one would expect a greater number of transversal elements to be found during cheap assignment in the rectangular matrix, thereby reducing the number which must be found by the more expensive depth first search. Output of the modified MC21A subroutine is an m -vector defining a row permutation of A to $\begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$. The permutation matrix R is, of course, never formed.

Selecting an A'_1

The matrix A'_1 , as noted earlier, is a member with fewest non-zeroes of the set of all possible $n \times n$ matrices with zero-free diagonal which may be formed by choosing n rows of A . A'_1 is not unique, a fact that will be discussed later.

The selection of A'_1 is motivated by heuristic considerations. If it is agreed that a final block upper trapezoidal structure with more zeroes below the diagonal blocks is more desirable than an alternate block upper trapezoidal structure with fewer such zeroes, then it is often better to have narrower (and thus more) diagonal blocks. Since the width of the diagonal blocks is completely determined by the size of the square diagonal blocks in A^*_1 at the end of step three, it follows that one would like to promote, if possible, smallness of these blocks. This may be done, in a heuristic sense, by

choosing a matrix A'_1 for block upper triangularization in step three for which the associated graph has few edges and therefore tends to have more, smaller, connected components rather than a few large ones. The size of the connected components determines the size of the diagonal blocks. Using an arbitrary A_1 rather than A'_1 can have a drastic effect on the block structure of the final form, as shown in the examples in the first two appendices.

In the example below, the rows of RA in (2.2), are permuted to yield $R'RA$ in (2.3). The effect of this permutation is to replace an A_1 submatrix with the A'_1 submatrix.

$$RA = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} A_1 \\ \hline A_2 \end{bmatrix} \quad (2.2)$$

$$R'RA = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad (2.3)$$

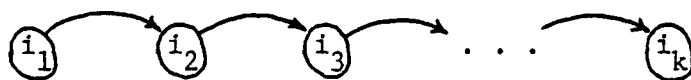
Note that the A_1 submatrix in (2.2) is in block upper triangular form, with one 3×3 diagonal block and one 1×1 diagonal block.

The A'_1 submatrix in (2.3) is not yet in indecomposable block upper triangular form. This form is shown below. Note the four 1×1 diagonal blocks.

$$\begin{bmatrix} Q^t & 0 \\ 0 & I_{m-n} \end{bmatrix} R'RAQ = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad (2.4)$$

The subroutine MIN listed in Appendix 4 finds an A'_1 using a graph theoretical technique: In the discussion below, a path is sought in a directed graph with m vertices, each corresponding to one of the m rows of the matrix RA , and where the edge $(i) \rightarrow (j)$ corresponds to a nonzero entry a_{ji} . Note that the existence of an edge from i to j means that if row i is currently in A_1 , row j could replace row i and A_1 will still have a zero-free main diagonal. If row j had previously been in A_2 and had fewer nonzeros than row i , then exchanging row i and row j would have the desired effect of reducing the number of nonzeros in A_1 while preserving the zero-free main diagonal.

In general, a path corresponding to a desirable row permutation is of the form



where all vertices except i_k correspond to rows in A_1 and row i_k has fewer nonzeros than row i_1 . The cyclic row permutation defined

by this path would replace each row by the row after it on the path and would replace row i_k by row i_1 . This would result in a net exchange of one row between A_1 and A_2 . The permutation corresponding to each such path is performed as the path is found.

The path can end without defining a desirable permutation. This can happen either of two ways. The path may reach a vertex corresponding to a row in A_1 from which there are no leaving edges, or the path may reach a vertex corresponding to a row in A_2 which does not contain fewer nonzeros than the row at the beginning of the path. If either event occurs, the last edge on the path is removed and a replacement sought. This process is called backtracking.

The following path could be constructed using the matrix in (2.2), $\textcircled{1} \rightarrow \textcircled{3} \rightarrow \textcircled{2} \rightarrow \textcircled{5}$. The path ends at 5 since $5 > 4 = n$, and the fifth row contains fewer nonzeros than does the first row. Performing the reassignment indicated by this chain involves replacing row 1 by row 3, row 3 by row 2, row 2 by row 5, and row 5 by row 1. The resulting matrix is shown below.

$$R'RA = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (2.5)$$

Performing the permutation results in a net gain of one zero for A_1 . At each stage in the construction of a path, MIN will attempt to add

a vertex corresponding to a row in A_2 before adding one corresponding to a row in A_1 , thus tending to find the shortest and simplest permissible path. A flowchart for subroutine MIN can be found in Appendix 4.

Block Triangularizing A'_1

The square submatrix A'_1 can be permuted to upper block triangular form by the modified Harwell subroutine MC130 (Duff and Reid [1978]) discussed in Chapter 1. No special modifications need be made for the rectangular case since A'_1 is square. The user must remember, however, not only to symmetrically permute the rows and columns of A'_1 , but also the columns of A_2^* as well.

Row Permutation to Final Form

After the matrix has been permuted so that A_1^* is block upper triangular with square indecomposable diagonal blocks, the rows of the lower submatrix A_2^* must be permuted into the A_1^* matrix in such a manner as to obtain the block upper trapezoidal form. This will make some of the diagonal blocks rectangular and will remove all nonzeros from beneath the diagonal blocks. The subroutine ABSORB, listed in Appendix 6, determines the row permutation. Each row in A_2^* is examined to determine the column index of its first nonzero. The row is then inserted into A_1^* just above the row containing the transversal element in that column. If another row in A_2^* has its first nonzero in the same column, this row is inserted into A_1^* just above the row

previously inserted. For example, ABSORB would permute the matrix in (2.4) in the following manner.

$$P \begin{bmatrix} Q^t & 0 \\ 0 & I_2 \end{bmatrix} R'RAQ = \begin{bmatrix} \boxed{1} & 0 & 1 & 1 \\ 1 & \boxed{1} & 1 & 0 \\ 1 & 1 & \boxed{0} & 0 \\ 0 & \boxed{1} & 1 & 0 \\ 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 0 & \boxed{1} \end{bmatrix} \quad (2.6)$$

Uniqueness of Final Block Structure

The final block upper trapezoidal structure of a given matrix is of course not unique. In general there are several possible maximal transversals yielding several different A'_1 submatrices, each having the minimum number of nonzeros possible. Another cause for lack of uniqueness of the final block structure is the fact that in general there may be more than one way in which the diagonal blocks may be permuted among themselves and still preserve the block upper trapezoidal form. Some of these permutations are more desirable than others, as they result in a "dual angular" form discussed in the following chapter.

CHAPTER 3

DUAL ANGULAR FORM

Although this thesis is concerned primarily with developing an algorithm for permuting a matrix to a general block upper trapezoidal form, there is a special case of this form which has also received attention, the dual angular form. A matrix in dual angular form has rectangular diagonal blocks with no nonzeros either above or below the diagonal blocks, followed by a block bordering the right side of the matrix as in (3.1).

$$A = \begin{bmatrix} \boxed{A_{11}} & & & \\ & \boxed{A_{22}} & & \\ & & \ddots & \\ & & & \boxed{A_{kk}} & \\ & & & & \boxed{D} \end{bmatrix} \quad (3.1)$$

Matrices of this type arise in situations where a large problem may be defined in terms of several smaller problems (each resulting in an A_{ii} submatrix) tied together in a well-defined mathematical way (the D matrix). An example can be found in geodetic adjustments where an observation matrix can be expressed in dual angular form by a process known as "Helmert blocking," described by Golub and Plemmons [1980] and Joyner [1980]. A dual angular representation of a matrix is attractive for the same reasons a block upper trapezoidal representation is--even more so. Savings in time and core storage requirements during solution of a linear least squares problem with dual angular

observation matrix result by limiting orthogonal decomposition to the diagonal blocks and some rows of the border block D , as described by Golub and Plemmons [1980] and Joyner [1980].

A form similar to the dual angular form, called block angular form is achieved by an algorithm developed by Weil and Kettler [1971]. Their algorithm was motivated mainly by consideration of underdetermined systems of linear equations in mathematical programming problems, but it can be modified to make it applicable to least square problems as well. The block angular form is different from the dual angular form in that the dense block D lies across the top border of the matrix as shown in (3.2) rather than along the right border.

$$A = \begin{bmatrix} \text{---} D \text{---} \\ A_{11} \\ & A_{22} \\ & & A_{kk} \end{bmatrix} \quad (3.2)$$

A brief examination of Weil and Kettler's algorithm is justifiable at this point since it may provide insight into a similar algorithm for permuting a matrix to dual angular form.

The Weil and Kettler algorithm consists of five steps. The original matrix A is of dimension $m \times n$ with $m \leq n$ and no condition is put on the rank of A . In the first step, a maximal transversal, which might be of length $p < m$, is selected and the matrix A

is permuted so that a $p \times p$ submatrix A_{11}^* exists in the "northwest corner" of the resulting matrix and has the transversal elements on its main diagonal. In the second step, A_{11}^* is block upper triangularized obtaining indecomposable diagonal blocks. The matrix now has the form shown in (3.3).

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & 0 \end{bmatrix} = \begin{bmatrix} \begin{array}{c|c} \begin{array}{ccc} \boxed{\text{diagonal block}} & \dots & \boxed{\text{diagonal block}} \\ & \ddots & \\ & & \boxed{\text{diagonal block}} \end{array} & \boxed{\text{diagonal block}} \\ \hline \boxed{\text{diagonal block}} & 0 \end{array} \end{bmatrix} \quad (3.3)$$

In the third step, each column of A_{12} is processed in one of the following ways:

1. The column is permuted into one of the diagonal blocks.
This may only be done if the permutation introduces no nonzeros to the left of the diagonal blocks of A_{11} .
2. The column is permuted into A_{11} and used to merge two diagonal blocks. Once again, this is only permissible if it leaves no nonzeros to the left of the new diagonal blocks.
3. The column is left where it is and the nonzeros dealt with in step 5.

In the fourth step the modified rows of A_{21} are processed.

Each of these is processed in one of three ways:

1. The row is permuted into one of the diagonal blocks above, if such a permutation introduces no nonzeros to the left of any of the diagonal blocks.
2. The row is used to merge two of the diagonal blocks above, if such a permutation introduces no nonzeros to the left of any of the diagonal blocks in the resulting block structure.
3. The row is permuted above the diagonal block structure into D , forming an upper border block.

The block angular form is now nearly achieved. There are no rows below the diagonal blocks at this point and all that remains is to deal with the nonzeros to the right of the diagonal blocks. Each of these nonzeros is dealt with in two possible ways in the fifth step:

1. The entire row containing the nonzero is permuted up into D .
2. The nonzero is "absorbed" into the diagonal block structure by the merger of two of the diagonal blocks.

Once all these nonzeros are processed, dual angular form is achieved.

There are some obvious difficulties associated with a computer implementation of such an algorithm. Step two, the permutation of A_{11}^* to block triangular form, is the only step in which no arbitrary decisions are apparently made, but even here the order of the k blocks along the diagonal may be chosen $k!$ different ways. In step one, while Weil and Kettler note the choice of transversal influences the final form, no basis for making this choice is given. In steps three

through five there are generally at least two ways to deal with a particular row, column or nonzero, and the final form is certainly influenced by the alternative chosen. The question of when it is worthwhile to merge diagonal blocks is particularly difficult, since each merger incorporates more zeroes into the diagonal blocks, which is contrary to the general aim of the exercise. Weil and Kettler [1971] give a criterion function shown in (3.3) which assigns a value to a block angular matrix based on its block structure.

$$f = \sum_{i=1}^k (d + n_i)^2 \quad (3.3)$$

Here n_i is the number of rows in diagonal block i , $1 \leq i \leq k$, and d is the number of rows in D . This gives a method for comparing alternative block structures for a given matrix, with lower f values being perhaps more desirable in linear programming applications. The function f , however, is of limited value since it allows only after-the-fact comparisons and is of little help in making the decisions which must be made during the permutation.

An algorithm for permuting a matrix A of dimension $m \times n$ with $m \geq n$ and column rank n to dual angular form might parallel Weil and Kettler's and consist of the following four steps.

In the first step a maximal transversal is chosen and A permuted to bring the transversal elements to the main diagonal of A_1 , the $n \times n$ upper submatrix of $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$. In step two the A_1 matrix is block upper triangularized.

In step three, each row of A_2 is processed in one of three possible ways:

1. The row is permuted into one of the diagonal blocks of A_1 , if such a permutation could be performed without introducing nonzeros below the diagonal blocks.
2. The row is permuted into A_1 and used to merge two diagonal blocks, if such a permutation would not introduce nonzeros below the diagonal blocks.
3. The row is left below A_1 and the nonzeros in it dealt with in step four.

In step four, the nonzeros not in the diagonal blocks A_{ii} are dealt with in one of the following two ways:

1. The entire column containing such a nonzero is permuted to the extreme right of the matrix. These columns form the D block.
2. Some such nonzeros are incorporated into the diagonal blocks by the merger of two diagonal blocks.

After each of the nonzeros not in an A_{ii} or in D have been dealt with, the algorithm is complete, and the matrix is in a dual angular form.

Just as with Weil and Kettler's method, such an algorithm would be difficult to implement due to the many decisions which must be made arbitrarily and which influence the final block structure. The most difficult such decision is whether or not to merge blocks in steps three and four. Merging blocks seems heuristically to be a poor choice unless the blocks are small. The merger of large blocks

potentially incorporates into the new block a number of zero entries equal to the total number of entries in the original blocks.

There is a method for modifying the algorithm discussed in Chapter 2 so that it selects a special dual angular form when permuting A to a general block upper trapezoidal form. In step three of the algorithm, where the A_1 submatrix is block upper triangularized, the decision of order along the diagonal of the diagonal blocks was made arbitrarily. It may, however, be possible to examine the order chosen and reorder the blocks to achieve a dual angular form. The subroutine PRMBLK, listed in Appendix 6, is designed to be called after MC13D and before ABSORB. PRMBLK examines the order of the diagonal blocks and rearranges it if a rearrangement is more desirable. The method used by PRMBLK is simple. Each column of the matrix is examined and "marked" if it contains a nonzero which will not be in a diagonal block after ABSORB is applied. Next the diagonal blocks are examined and divided into two sets. One set contains the diagonal blocks whose columns are all unmarked, and the other set contains those diagonal blocks which contain at least one marked column. The matrix is then symmetrically permuted so that all diagonal blocks in the first set occur on the main diagonal before all those in the second. The subroutine ABSORB is then called. The matrix is now in a dual angular form if the columns of all the diagonal blocks in the second set are considered to be the border block D .

Of course, since block merger is not considered, the dual angular form of the matrix may not be "optimal." The advantage of PRMBLK is that it is an inexpensive way to examine a matrix which is being

permuted to block upper trapezoidal form and see if it may be easily permuted to block upper trapezoidal form. If there is an inherent dual angular form due to the nature of the problem, PRMBLK will discover it. Otherwise, the matrix is permuted to a general block upper trapezoidal form. It should be noted that although the dual angular form would seem more attractive as a starting point for the orthogonal QR decomposition in the solution of a least squares problem, the algorithm developed by George, Heath, and Plemmons for QR decomposition of a block upper trapezoidal matrix takes advantage of sparsity in the portion of the matrix above the diagonal blocks. Therefore, unless a dual angular form is "natural" for the particular problem under consideration, it is probably unwise to try to force the matrix into any special form beyond general block upper trapezoidal.

CHAPTER 4

SUMMARY AND COMMENTS

The subroutine MIN discussed in Chapter 2, takes a rectangular matrix $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ where A_1 has a zero-free main diagonal and permutes A so that in $RA = \begin{bmatrix} A'_1 \\ A'_2 \end{bmatrix}$, the A'_1 submatrix not only has a zero-free main diagonal but also contains the greatest possible number of zeroes. Hopefully, A'_1 will eventually yield a more desirable block structure than A_1 . Sometimes, however, A'_1 yields a less desirable block triangular structure than some other A_1 . For example, in the matrix A given by (4.1), block triangularization of A_1 would yield a 1×1 block and a 3×3 block.

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 \end{bmatrix} \quad (4.1)$$

However, block triangularization of the A'_1 submatrix shown below would yield a single 4×4 block.

$$RA = \begin{bmatrix} A'_1 \\ A'_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 1 \end{bmatrix}$$

Therefore, the justification for MIN given in Chapter 2 is only heuristic. In general, however, finding A'_1 is worthwhile. This can be readily seen in the examples in the first two appendices. In Illustration 1 of Appendix 1, a random matrix A of dimension 75×50 and nonzero density of approximately 6.2% is permuted to block upper trapezoidal form, calling MIN as in the algorithm presented in Chapter 2. The final form consists of one diagonal block with 11 columns and 39 diagonal blocks with one column each. For comparison purposes, the same matrix is again permuted to a block upper triangular in Illustration 2 of Appendix 1. This time, however, the subroutine MIN is omitted from the algorithm. The resulting final form has only 12 diagonal blocks with single columns and one diagonal block with 38 columns. Heuristically, the effect of deleting MIN from the algorithm is to make the graph of A_1 have more edges and, as in the case of this example, also have larger connected components. This leads to fewer and larger diagonal blocks. The final form found in Illustration 1 is clearly more desirable, having 1414 zeroes below the diagonal whereas the final form in Illustration 2 has only 761 such zeroes. In Illustration 3 of Appendix 1 the same matrix is block triangularized without first finding a zero-free main diagonal. The

final block upper trapezoidal form consists of only 2 diagonal blocks and is clearly little improvement over the original unpermuted matrix. So for this example, finding A_1 is important and finding A'_1 is also worthwhile. In Appendix 2 the above procedure is repeated for a random matrix of nonzero density approximately equal to 8.3% and dimension 75×50 . The results are similar although the greater density results in fewer and larger blocks in each case. In Illustration 1 of Appendix 2 where MIN is called, the final form has 11 diagonal blocks with single columns and one diagonal block with 39 columns. In Illustration 2 of Appendix 2 final form has only two diagonal blocks with one column each and one diagonal block with 48 columns. In Illustration 3 of Appendix 2 the final form has only one diagonal block and once again is little improvement over the original unpermuted matrix.

The subroutine MIN is thus shown to be a useful component of an algorithm for permuting a general matrix to block upper trapezoidal form. It should be noted, however, that the subroutine MIN requires additional computer time. Depending on the particular qualities of the matrix being permuted, MIN can require more time than either MC21A or MC13D (both ABSORB and PRMBLK are relatively inexpensive), particularly for matrices with many more rows than columns. Furthermore, for matrices in which the number of nonzeros per row is constant or nearly constant the A'_1 matrix is not significantly different from A_1 , and calling the MIN subroutine is ill-advised. Therefore, the subroutine MIN should be called only if variation in row density is significant. Matrices where the row density is nearly constant are common in least

squares problems associated with the adjustment of geodetic data and with other network related problems.

The minimal amount of variation necessary to justify calling the MIN subroutine is a question that should be investigated further. The justification should be a function not only of the dimensions of the matrix, but probably also of the nonzero density of the matrix.

The subroutine MIN selects a transversal which minimizes the number of nonzeros in the square submatrix A'_1 . There could be other motivations for choosing a particular transversal, however, which would lead to a different A'_1 . One such motivation might be to pick an A'_1 so that each row of A_2^* , defined in Chapter 2, has its first nonzero as far to the right as possible. This would mean more of the zeroes in A_2^* would end up to the left of the diagonal blocks than would if the rows of A_2^* had nonzeros in the first few columns. The expected value of c , the column index of the first nonzero in a row of a random matrix of nonzero density p is shown in (4.2).

$$E(c) = \sum_{t=1}^n tp(1-p)^{t-1} + (n+1)(1-p)^n \quad (4.2)$$

This expected value is fairly sensitive to changes in p , as shown by (4.2),

$$\frac{\partial E(c)}{\partial p} = \frac{p(n+1)(1+p)^n + (1-p)^{n+1} - 1}{p^2} \quad (4.3)$$

and its value for the rows of A_2^* could hopefully be increased by choosing the rows of A'_1 as dense as possible. Of course this choice of A'_1 is completely contrary to the choice of A'_1 performed by the MIN subroutine, and would probably only be advantageous in situations

where A is extremely sparse and has little variation in row density.

Another area in which future work should be done is the incorporation of some scheme for block merger into the subroutine PRMBLK. The ability of PRMBLK to optionally merge blocks would make it much more powerful and in certain instances would allow a matrix to be permuted into a "better" dual angular form than would otherwise be possible. Block merger would probably only be effective if the blocks involved are small.

Finally, although this thesis has dealt only with permutations of matrices where the maximal transversal has length n , a similar algorithm for the permutation of matrices with a maximum transversal of length $k < n$ should be developed. Applications of this situation occur in mathematical programming problems and in solving rank deficient least squares problems.

LIST OF REFERENCES

LIST OF REFERENCES

1. I. S. Duff, "On Algorithms for Obtaining a Maximum Transversal," AERE Harwell Report CSS49, (1978).
2. I. S. Duff and J. K. Reid, "An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix," ACM Transactions on Mathematical Software, vol. 4 (1978), 137-147.
3. I. S. Duff and J. K. Reid, "Some Design Features of a Sparse Matrix Code," ACM Transactions on Mathematical Software, vol. 5 (1979), 18-35.
4. A. George and M. T. Heath, "Solution of Sparse Linear Least Squares Problems Using Givens Rotations," to appear in Linear Algebra and Its Applications, Special Issue on Large Scale Matrix Problems, (1980).
5. A. George, M. T. Heath, R. J. Plemmons, "Solution of Large Scale Sparse Least Squares Problems Using Auxiliary Storage," submitted to Siam Journal on Scientific and Statistical Computing.
6. G. H. Golub and R. J. Plemmons, "Large Scale Geodetic Least Squares Adjustment by Dissection and Orthogonal Decomposition," to appear in Linear Algebra and Its Applications, Special Issue on Large Scale Matrix Problems, (1980).
7. T. D. Howell, "Partitioning Using PAQ^t ," Sparse Matrix Computations, ed. by J. R. Bunch and D. J. Rose, Academic Press, NY (1976).
8. C. Joyner, "Block Decomposition Algorithms for Matrices in Dual Angular Form," thesis for the M.S. degree, Dept. of Math., Univ. of TN, Knoxville, TN (1980).
9. R. L. Weil and P. C. Kettler, "Rearranging Matrices to Block-Angular Form for Decomposition (and Other) Algorithms," Management Science, vol. 18, (1971), 98-108.

APPENDICES

APPENDIX 1

EXAMPLE A

This appendix contains three illustrations. In each illustration a different algorithm is used to permute a rectangular matrix to an upper block trapezoidal form. The same matrix is used for all three illustrations and has dimensions 75×50 , a transversal of length 50, and nonzero density of approximately 6.2%. This matrix is labelled A.

In Illustration 1, the algorithm which involves MIN given in Chapter 2 is employed. In Illustration 2, the subroutine MIN is not employed during the permutation, leading to a different (and inferior) final form. In Illustration 3, both subroutines MC21A and MIN are omitted from the algorithm. Hence, a zero-free main diagonal is never found during the permutation. The resulting final form is little improvement over the original matrix.

APPENDIX 2

EXAMPLE B

This appendix contains three illustrations. In each illustration a different algorithm is used to permute a rectangular matrix to an upper block trapezoidal form. The same matrix is used for all three illustrations and has dimensions 75×50 , a transversal of length 50, and nonzero density of approximately 8.3%. This matrix is labelled B.

In Illustration 1, the subroutine MIN is employed, as described in Chapter 2. In Illustration 2, the subroutine MIN is not employed during the permutation, leading to a different (and inferior) final form. In Illustration 3, both subroutines MC21A and MIN are omitted from the algorithm. Hence, a zero-free main diagonal is never found during the permutation. The resulting final form is no improvement over the original matrix.

APPENDIX 3

THE MODIFIED MC21A SUBROUTINE

This appendix contains an argument list and FORTRAN listing for the Harwell subroutine MC21A , modified slightly to accommodate a rectangular matrix.

ARGUMENT LIST

- M must be set to the row index of the matrix. It is not altered by MC21A .
- N must be set to the column index of the matrix. It is not altered by MC21A .
- LICN must be set equal the number of nonzeros in the matrix. It is not altered by MC21A .
- ICN is an integer array of length LICN which must be set by the user to contain the column indices of the nonzeros. Those belonging to a single row must be contiguous, but the order of rows and the order of column indices within each row is irrelevant. ICN is not altered by MC21A .
- IP is an integer array of length M which must be set by the user so that IP(I) contains the position in ICN of the first column index of a nonzero in row I , for $I = 1, 2, \dots, M$. It is updated by MC21A to reflect the row permutation R .
- LENR is an integer array of length M which must be set by the user so that LENR(I) contains the number of nonzeros in

row I , for $I = 1, 2, \dots, M$. It is updated by MC21A to reflect the row permutation R .

IPERM is an integer array of length M in which the row permutation is output. IPERM(I) gives the position in the original matrix of row I in the permuted matrix for $I = 1, 2, \dots, M$.

NUMNZ is an output variable which gives the number of nonzeros on the main diagonal of the reordered matrix. If the matrix has full column rank, MC21A will set NUMNZ to N .

LEN, IPP, and IW are work arrays.

```

SUBROUTINE MC21A(M,N,LEN,IPP,ICN,LICN,IP,LENR,IPERM,
1  NUMNZ,IW)

```

```

C
C   ALL SECTIONS OF THIS SUBROUTINE BOUNDED ABOVE BY
C   A LINE * * * *, AND BOUNDED BELOW BY A LINE *****
C   CONTAIN MODIFICATIONS IN THE ORIGINAL HARWELL
C   SUBROUTINE.

```

```

C * * * * *

```

```

      INTEGER LEN(M),IPP(M)

```

```

      INTEGER IP(M),ICN(LICN),LENR(M),IPERM(M),IW(M,4)

```

```

C *****

```

```

      CALL MC21B(M,N,LEN,IPP,ICN,LICN,IP,LENR,IPERM,

```

```

1  NUMNZ,IW(1,1),IW(1,2),IW(1,3),IW(1,4))

```

```

C * * * * *

```

```

C   THIS SECTION OF THE PROGRAM WILL MAKE IPERM INTO
C   A "COMPLETE" ROW PERMUTATION.

```

```

C

```

```

      MM=N+1

```

```

      DO 2 K=1,M

```

```

2      LEN(K) = 0

```

```

5      K=1

```

```

10     INT=0

```

```

      DO 20 L=1,N

```

```

      IF(IPERM(L).EQ.K) GO TO 15

```

```

      GO TO 20

```

```

15     INT=1

```

```

      GO TO 25

```

```

20     CONTINUE

```

```

25     IF(INT.EQ.0.AND.LEN(K).EQ.0) GO TO 30

```

```

      K=K+1

```

```

      GO TO 10

```

```

30     IPERM(M) = K

```

```

      LEN(K)=1

```

```

      IF(MM.EQ.M) GO TO 35

```

```

      MM=MM+1

```

```

      GO TO 5

```

```

35     CONTINUE

```

```

C

```

```

C

```

```

      THIS SECTION UPDATES LENR AND IP

```

```

C

```

```

      DO 40 K=1,M

```

```

      LEN(K)=LENR(IPERM(K))

```

```

40     IPP(K)=IP(IPERM(K))

```

```

      DO 45 K=1,M

```

```

      LENR(K)=LEN(K)

```

```

45     IP(K)=IPP(K)

```

```

C *****

```

```

      RETURN

```

```

      END

```

```

SUBROUTINE MC216(M,N,LEN,IPP,ICN,LICN,IP,LENR,Iperm,
1  NUMNZ,PR,ARP,CV,OUT)
C * * * * *
INTEGER IP(M)
INTEGER LEN(M),IPP(M)
INTEGER ICN(LICN),LENR(M),Iperm(M),PR(M),CV(M),
1  ARP(M),OUT(M)
DO 10 I=1,M
C *****
ARP(I) = LENR(I)-1
CV(I)=0
10  Iperm(I)=0
NUMNZ = 0
C * * * * *
DO 130 JORD=1,M
C *****
J=JORD
PR(J)=-1
DO 100 K=1,JORD
IN1=ARP(J)
IF (IN1.LT.0) GO TO 60
IN2=IP(J)+LENR(J)-1
IN1=IN2-IN1
DO 50 II=IN1,IN2
I = ICN(II)
IF(Iperm(I).EQ.0) GO TO 110
50  CONTINUE
ARP(J)=-1
60  OUT(J)=LENR(J)-1
DO 90 KK=1,JORD
IN1=OUT(J)
IF(IN1.LT.0) GO TO 80
IN2=IP(J)+LENR(J)-1
IN1=IN2-IN1
DO 70 II=IN1,IN2
I=ICN(II)
IF (CV(I).EQ.JORD) GO TO 70
J1 = J
J=Iperm(I)
CV(I)=JORD
PR(J)=J1
OUT(J1) = IN2-II-1
GO TO 100
70  CONTINUE
80  J=PR(J)
IF(J.EQ.-1) GO TO 130
90  CONTINUE
100 CONTINUE
110 Iperm(I)=J
ARP(J)=IN2-II-1
NUMNZ=NUMNZ+1
DO 120 K=1,JORD
J =PR(J)

```

```
IF(J.EQ. -1) GO TO 130
II=IP(J)+LENR(J)-OUT(J)-2
I=ICN(II)
IPERM(I)=J
120  CONTINUE
    IF(NUMNZ.EQ.N) RETURN
130  CONTINUE
    IF(NUMNZ.EQ.N) RETURN
    DO 140 I=1,N
140  ARP(I)=0
      K=0
      DO 160 I=1,N
        IF(IPERM(I).NE.0) GO TO 150
        K=K+1
        OUT(K)=I
        GO TO 160
150  J= IPERM(I)
        ARP(J)=I
160  CONTINUE
      K=0
      DO 170 I=1,N
        IF (ARP(I).NE.0) GO TO 170
        K=K+1
        ICUTK=OUT(K)
        IPERM(ICUTK)=I
170  CONTINUE
      RETURN
      END
```

APPENDIX 4

THE MIN SUBROUTINE

This appendix contains an argument list, FORTRAN listing, and flowchart of the subroutine MIN . The argument list is shown below, followed by the FORTRAN listing and flowchart.

ARGUMENT LIST

NN is an output variable set by MIN to be the number of passes around the main loop required to find the desired permutation.

LINK, RLINK, MARK are work arrays of length M .

The remaining arguments are defined in Appendix 3.

```

SUBROUTINE MIN(NN,N,M,LICN,LINK,RLINK,
1  LENR,IP,ICN,MARK)
  INTEGER RLINK(M),MARK(M)
  DIMENSION LINK(M),LENR(M),IP(M),ICN(LICN)
  INTEGER CHANGE,BEST,PATH
  NN=0
1  JJ = 1
  NN=NN+1
  CHANGE = 0
5  DO 10 L=1,M
  LINK(L) = 0
10  RLINK(L) = 0
12  PATH = JJ
  LENGTH = LENR(JJ)
C
C  IN FORMING A SUBROUTINE "MAX" WHICH MAXIMIZES
C  RATHER THAN MINIMIZES THE NUMBER OF NONZEROCES
C  IN THE A1' MATRIX, THE FOLLOWING LINE SHOULD
C  BE DELETED.
C
  IF(LENGTH.EQ.1) GO TO 30
  IF(LINK(JJ).NE.0) GO TO 30
13  II = M
  MM=0
C
C  THE FOLLOWING SECTION SEARCHES A2 FOR
C  A ROW WITH FEWER NONZEROCES THAN ROW JJ.
C  IF SUCH A ROW EXISTS, IT IS EXAMINED TO
C  DETERMINE IF IT CAN BE ADDED TO THE PATH.
C
C  THE "GE" IN THE FOLLOWING LINE SHOULD BE
C  CHANGED TO "LE" IN A SUBROUTINE MAX.
C
15  IF(LENR(II).GE.LENGTH) GO TO 25
  MM=1
  IPL = IP(II)
  IPLL = IPL + LENR(II) - 1
  DO 20 L=IPL,IPLL
20  IF(ICN(L).EQ.PATH) GO TO 35
25  II = II-1
  IF(II.GT.N) GO TO 15
C
C  IF A ROW IN A2 HAS FEWER NONZEROCES THAN
C  ROW JJ BUT CAN'T BE ADDED TO THE PATH, THE
C  PROGRAM ATTEMPTS TO ADD A ROW IN A1 TO
C  THE PATH.
C
  IF(MM.EQ.0) GO TO 30
26  IF(LINK(II).NE.0) GO TO 28

```

```

IF(II.EQ.PATH) GO TO 28
IPL = IP(II)
IPLL = IPL + LENR(II) - 1
DO 27 L=IPL,IPLL
27 IF(ICN(L).EQ.PATH) GO TO 35
28 II = II - 1
IF(II.GE.1) GO TO 26

```

C
C
C
C
C
C
C

IF NO ROW CAN BE ADDED TO THE PATH,
THE FOLLOWING SECTION PERFORMS A
"BACKTRACK" STEP.

```

IF(PATH.EQ.JJ) GO TO 30
LINK(PATH) = PATH
PATH = RLINK(PATH)
II = N
GO TO 26
30 LINK(JJ) = JJ
IF(JJ.EQ.N) GO TO 90
JJ = JJ+1

```

C
C
C
C

THE "LE" IN THE FOLLOWING LINE SHOULD
BE CHANGED TO "GE" IN THE SUBROUTINE MAX.

```

IF(LENR(JJ).LE.LENGTH) GO TO 12
GO TO 5

```

C
C
C
C
C
C

THE FOLLOWING SECTION ADDS A ROW IN
A1 TO THE PATH.

```

35 IF(II.GT.N) GO TO 40
LINK(PATH) = II
RLINK(II) = PATH
PATH = II
GO TO 13

```

C
C
C
C
C
C
C

IN THIS SECTION, OF ALL ROWS IN A1 WHICH MAY
BE ADDED TO THE PATH, THE ONE WITH FEWEST
NONZERGES IS FOUND AND ADDED.

```

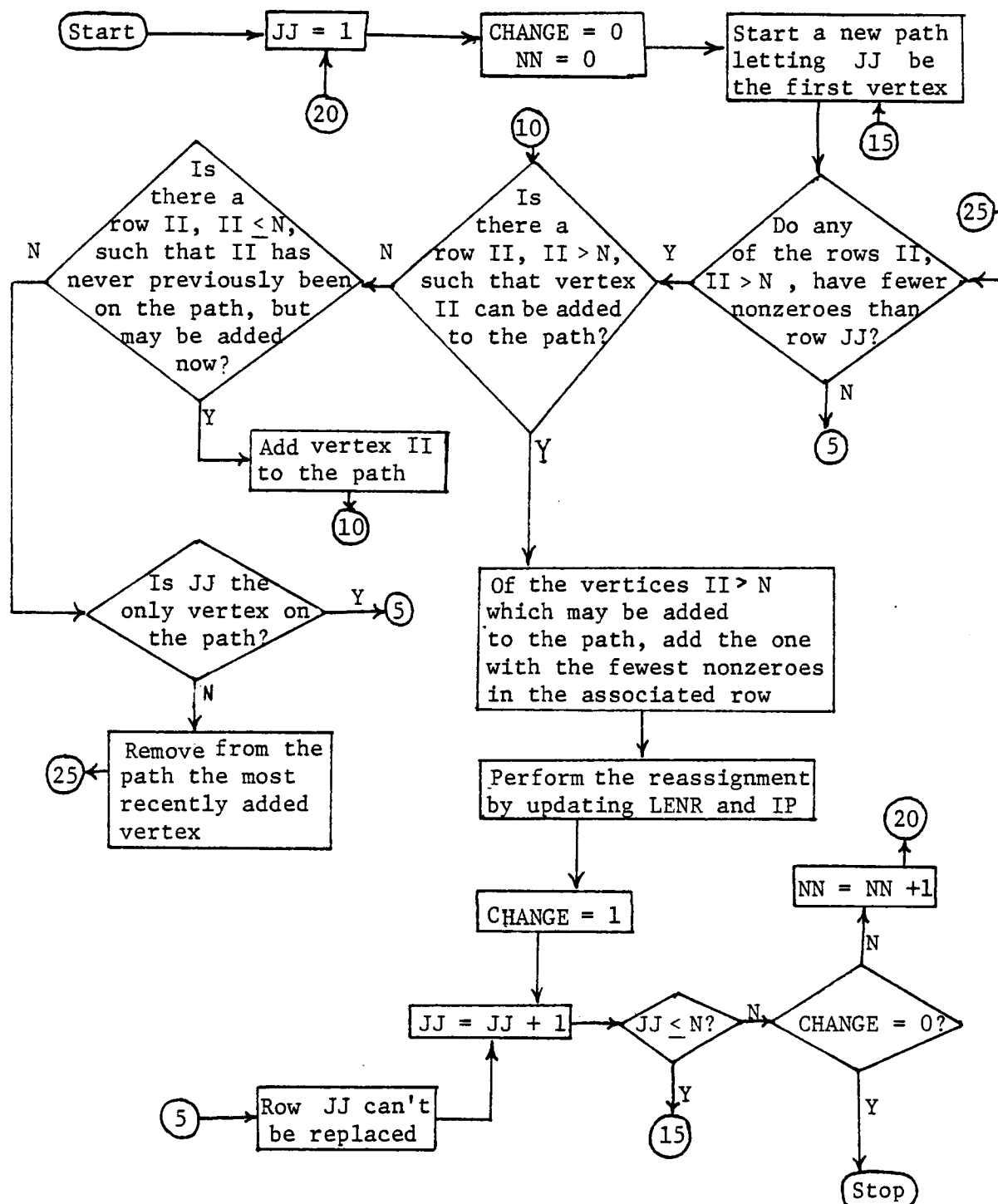
40 BEST = II
45 II = II - 1
IF(II.EQ.N) GO TO 60

```

C
C
C
C

THE "GE" IN THE FOLLOWING LINE SHOULD BE
"LE" IN THE SUBROUTINE MAX.

FLOWCHART FOR MIN



APPENDIX 5

THE MODIFIED MC13D SUBROUTINE

This appendix contains an argument list and FORTRAN listing for the Harwell subroutine MC13D modified slightly to accommodate rectangular matrices.

ARGUMENT LIST

ICN is defined in Appendix 3 and is altered by MC13D to reflect the column permutation.

IOR is an output array of length M which is set by MC13D so that IOR(I) contains the position in the input matrix of row (and column) I in the symmetrically permuted output matrix for $I = 1, 2, \dots, N$. For $N < I \leq M$, $IOR(I) = I$.

NUM is an output variable set to equal the number of diagonal blocks found.

IB is an output array of length NUM set by MC13D so that IB(I) is the row number in the permuted matrix of the beginning of diagonal block I, for $I = 1, 2, \dots, NUM$.

The remaining arguments are defined in previous appendices.

```

SUBROUTINE MC13D(M,N,LEN,IPP,ICN,LICN,IP,LENR,ICR, 76
1  IB,NUM,IW)
C * * * * *
INTEGER LEN(M),IPP(M)
INTEGER IP(M)
INTEGER ICN(LICN),LENR(M),ICR(M),IB(M),IW(M,3)
C *****
CALL MC13E(M,N,LEN,IPP,ICN,LICN,IP,LENR,ICR,IB,NUM,
1  IW(1,1),IW(1,2),IW(1,3))
IB(1) = 1
RETURN
END

```

```

SUBROUTINE MC13E(M,N,LEN,IPP,ICN,LICN,IP,LENR,ARP,
1  IB,NUM,LOWL,NUMB,PREV)
INTEGER STP,DUMMY
C * * * * *
INTEGER IPP(M),LEN(M)
INTEGER IP(M)
INTEGER ICN(LICN),LENR(M),ARP(M),IB(M),LOWL(M),
1  PREV(M),NUMB(M)
C *****
ICNT = 0
NUM=0
NNM1=N+N-1
DO 10 J = 1 ,N
NUMB(J)=0
ARP(J)=LENR(J)-1
10 CONTINUE
DO 90 ISN=1,N
IF(NUMB(ISN).NE.0) GO TO 90
IV = ISN
IST = 1
LOWL(IV)=1
NUMB(IV)=1
IB(N)=IV
DO 80 DUMMY=1,NNM1
I1=ARP(IV)
IF(I1.LT.0) GO TO 30
I2 = IP(IV) + LENR(IV) - 1
I1 = I2-I1
DO 20 II=I1,I2
IW=ICN(II)
IF(NUMB(IW).EQ.0) GO TO 70
IF(LOWL(IW).LT.LOWL(IV)) LOWL(IV)=LOWL(IW)
20 CONTINUE
ARP(IV)=-1
30 IF(LOWL(IV).LT.NUMB(IV)) GO TO 60
NUM=NUM+1
IST1 = N+1-IST
LCNT = ICNT+1
DO 40 STP= IST1,N

```

```

      IW = IB(STP)
      LOWL(IW)=N+1
      ICNT=ICNT+1
      NUMB(IW)=ICNT
      IF(IW.EQ.IV) GO TO 50
40    CONTINUE
50    IST = N-STP
      IB(NUM)=LCNT
      IF(IST.NE.0)GO TO 60
      IF (ICNT.LT.N)GO TO 90
      GO TO 100
60    IW=IV
      IV=PREV(IV)
      IF(LOWL(IW).LT.LOWL(IV)) LOWL(IV)=LOWL(IW)
      GO TO 80
70    ARP(IV) = I2-II-1
      PREV(IW)=IV
      IV=IW
      IST=IST+1
      LOWL(IV)=IST
      NUMB(IV)=IST
      K=N+1-IST
      IB(K)=IV
80    CONTINUE
90    CONTINUE
100   DO 110 I=1,N
      II=NUMB(I)
      II=N-II+1
      ARP(II)=I
110   CONTINUE
C * * * * *
      DO 120 KMK=1,N
      KNK=ARP(KMK)
      LEN(KMK)=LENR(KNK)
      IPP(KMK)=IP(KNK)
120   CONTINUE
      DO 130 KOK=1,N
      LENR(KOK)=LEN(KOK)
      IP(KOK)=IPP(KOK)
130   CONTINUE
      DO 150 K=1,LICN
      ICNK=ICN(K)
      DO 140 J=1,N
      IF(ICNK.EQ.ARP(J)) ICN(K)=J
140   CONTINUE
150   CONTINUE
C
C     THESE LAST LINES WILL REASSIGN VALUES TO IB.
C
      IF(NUM.GE.2) GO TO 160
      IB(1) = 1
      GO TO 190

```

```
160      NUMM = NUM - 1
      LOWL(1) = N - IB(NUM) + 1
      DO 170 LM=1,NUMM
      MM = LM + 1
      L = NUM-LM+1
      LL = NUM-LM
170      LOWL(MM) = IB(L) - IB(LL)
      IB(1)=1
      DO 180 LM=2,NUM
      MM=LM-1
180      IB(LM)=IB(MM)+LOWL(MM)
190      CONTINUE
C *****
      RETURN
      END
```

APPENDIX 6

THE PRMBLK SUBROUTINE

This appendix contains an argument list, FORTRAN listing, and flowchart for the subroutine PRMBLK . Below is the argument list, followed by the FORTRAN listing and flowchart.

ARGUMENT LIST

NUORD is an output array of length NUM set by PRMBLK so that
NUORD(I) contains the order in the input matrix of diagonal
block I in the permuted matrix.

MARK is an output array of length M set by PRMBLK so that
MARK(I) contains the position in the symmetrically permuted
output matrix of row (and column) I in the input matrix for
 $I = 1, 2, \dots, M$.

The remaining arrays are defined in previous appendices.

```

SUBROUTINE PRMBLK(M,N,LICN,NUM,IB,IBR,IP,
1  LENR,ICN,MARK,NUCRD,LEN,IPP)
  INTEGER IB(M),IP(M),LENR(M),ICN(LICN),MARK(M)
  INTEGER IBR(M),LEN(M),IPP(M),NUCRD(M)
  IB(NUM+1)=N+1
  DO 180 K=1,M
180  MARK(K) = 0
  LM=1

  C
  C
  C  SEARCH EACH ROW IN THE UPPER SQUARE SUBMATRIX.
  C  IF IT CONTAINS A NONZERO OUTSIDE ONE OF THE
  C  DIAGONAL BLOCKS, MARK THE COLUMN CONTAINING
  C  THAT NONZERO.
  C
190  DO 205 L=1,NUM
    IF(IB(L).GT.LM) GO TO 210
205  CONTINUE
210  IBL=IB(L)
  C
  C  IBL IS THE INDEX OF THE FIRST COLUMN IN
  C  THE NEXT BLOCK AFTER THE BLOCK CONTAINING ROW LM.
  C
    K= IP(LM)
    JJ=K+LENR(LM)-1
    DO 230 L=K,JJ
      IF(ICN(L).LT.IBL) GO TO 230
      MARK(ICN(L)) = 1
230  CONTINUE
    LM=LM+1
    IF(LM.LT.IB(NUM)) GO TO 190

  C
  C
    NN=N+1
    DO 237 L=NN,M
      LL=IP(L)
      LLL=IP(L) + LENR(L) - 1
      DO 235 LM=LL,LLL
235  MARK(ICN(LM)) = 1
237  CONTINUE
  C
  C
    L=1
    K=1

  C
  C  NUCRD(I) IS THE ORDER IN THE ORIGINAL MATRIX
  C  OF THE I-TH BLOCK IN THE PERMUTED MATRIX.
  C
  C  IF NONE OF THE COLUMNS IN THE I-TH BLOCK
  C  ARE MARKED, MOVE THIS BLOCK "UP" THE
  C  DIAGONAL.

```

```

240      IBL=IB(L)
      LL=L+1
      IBLL=IB(LL) -1
      DO 250 KK=IBL,IBLL
      IF(MARK(KK).EQ.1) GO TO 260
250      CONTINUE
      NUORD(K)=L
      K=K+1
260      L = L+1
      IF(L.LE.NUM) GO TO 240

C
C      IF ONE OF THE COLUMNS IN A BLOCK IS MARKED,
C      MOVE THAT BLOCK "DOWN" THE DIAGONAL.
C
      L = 1
270      IBL = IB(L)
      LL = L+1
      IBLL = IB(LL)-1
      DO 280 KK=IBL,IBLL
      IF(MARK(KK).EQ.1) GO TO 290
280      CONTINUE
      GO TO 285
290      NUORD(K) = L
      K=K+1
285      L= L+1
      IF(L.LE.NUM) GO TO 270

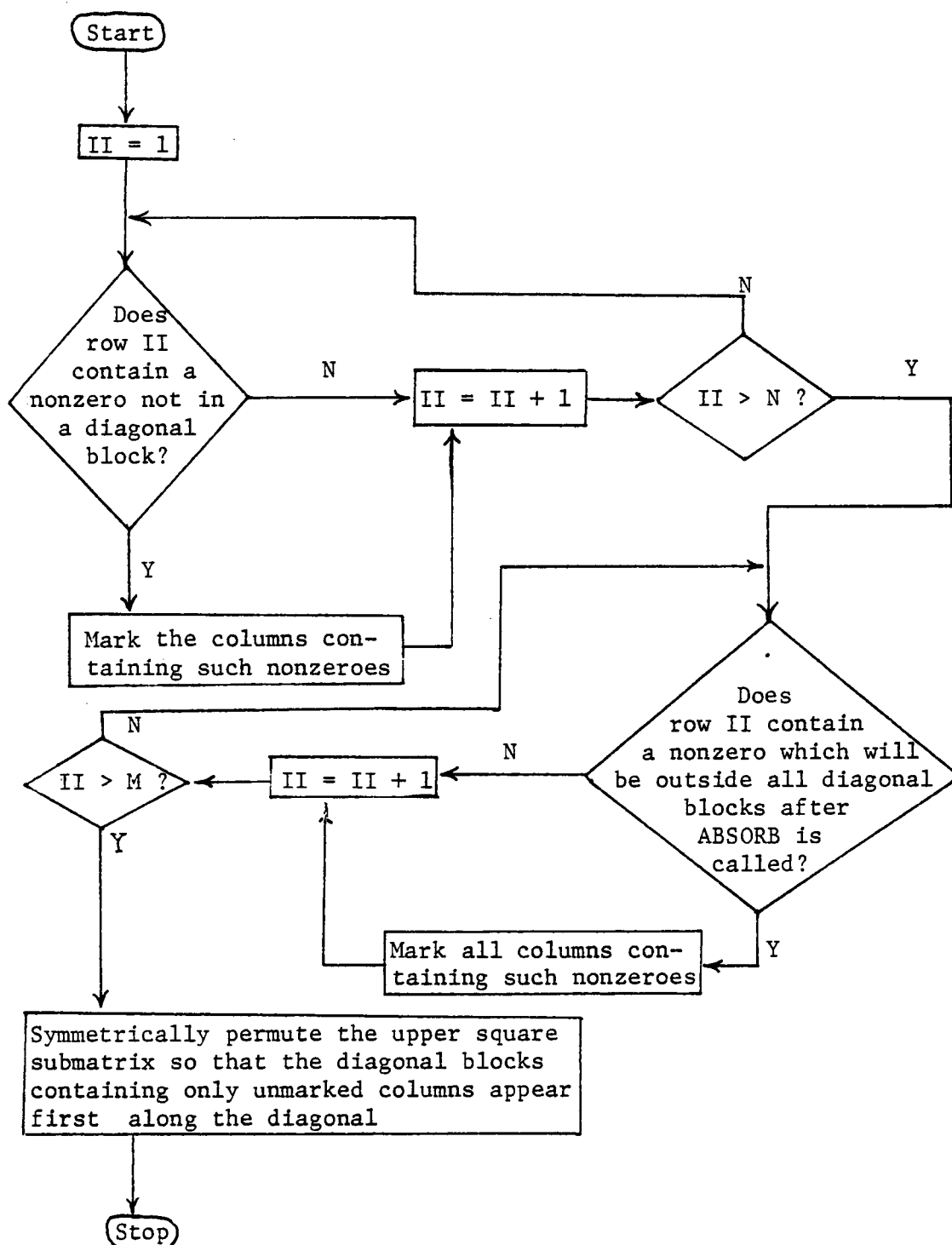
C
C      DEFINE ARRAY MARK SUCH THAT MARK(I)=K
C      MEANS THAT ROW (AND COLUMN) K WILL
C      REPLACE ROW (AND COLUMN) I.
C
      K = 1
      DO 300 L = 1,NUM
      IBL = IB(NUORD(L))
      LL = NUORD(L) + 1
      IBLL = IB(LL) - 1
      DO 295 MM = IBL,IBLL
      MARK(MM) = K
295      K = K+1
300      CONTINUE

C
C      PERFORM THE PERMUTATION BY UPDATING THE
C      ARRAYS IP AND LENR.
C
      DO 44 K=1,M
44      IF(MARK(K).EQ.0) MARK(K)=K
      DO 43 K=1,LICN
43      ICN(K) = MARK(ICN(K))
      DO 45 K=1,M
      LEN(MARK(K))=LENR(K)
      IF(K.LE.NUM) IBR(MARK(K)) = IB(K)
45      IPP(MARK(K)) = IP(K)

```

```
DO 47 K = 1,M  
  LENR(K) = LEN(K)  
  IP(K) = IPP(K)  
  IF(K.LE.NUM) IB(K)=IBR(K)  
RETURN  
END
```

FLOWCHART FOR PRMBLK



APPENDIX 7

THE ABSORB SUBROUTINE

This appendix contains an argument list, FORTRAN listing, and flowchart of the subroutine ABSORB . The argument list is below, followed by the FORTRAN listing and flowchart.

ARGUMENT LIST

IFR is an integer output array of length M set by ABSORB such that IFR(I) contains the location in the permuted output matrix of row I in the input matrix for $I = 1, 2, \dots, M$.

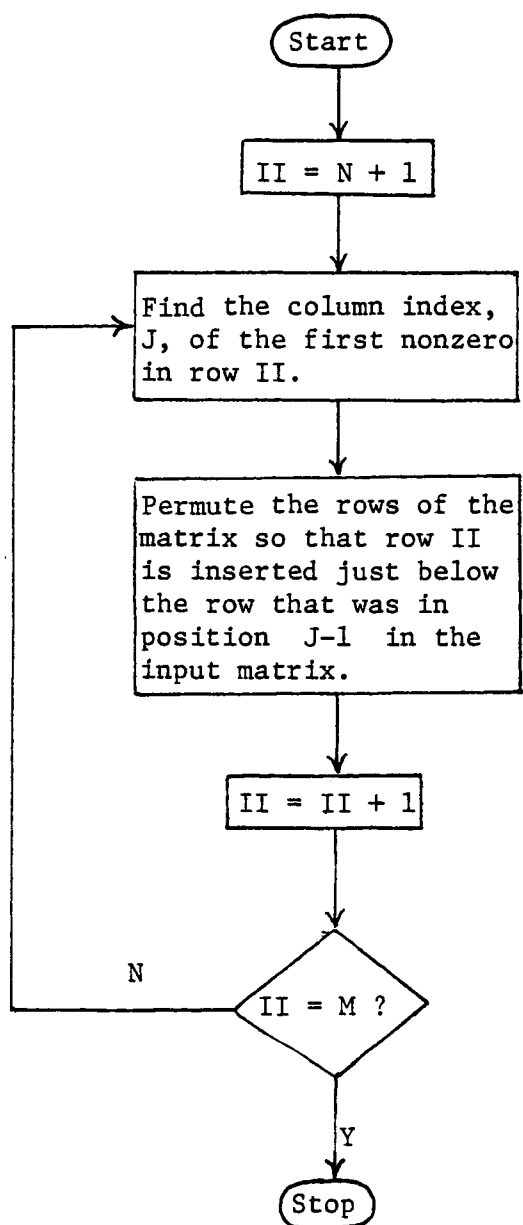
All other arrays are defined in previous appendices.

```

SUBROUTINE ABSORB(M,N,LENR,IP,IB,ICN,LICN,
1 NUM,IFR,IBR,IPP,LEN)
  INTEGER LENR(M),IPP(M),LEN(M),IBR(NUM)
  INTEGER IP(M),IB(NUM),ICN(LICN),IFR(M)
  J=N+1
  DO 1 II=1,N
1   IFR(II)=II
5   L=IP(J)
   LL=L+LENR(J)-1
   FIRST=N
   DO 10 K=L,LL
10  IF(ICN(K).LT.FIRST) FIRST=ICN(K)
   PLACE = IFR(FIRST) + 1
   IFR(J) = PLACE
   JJ = J-1
   DO 15 K=1,JJ
15  IF(IFR(K).GE.PLACE .AND. IFR(K).LE.JJ)
1   IFR(K)=IFR(K)+1
   IF(J.EQ.M) GO TO 30
   J = J+1
   GO TO 5
30  CONTINUE
   DO 40 L=1,M
   IPP(IFR(L))=IP(L)
40  LEN(IFR(L))=LENR(L)
   DO 50 L=1,M
   IP(L)=IPP(L)
50  LENR(L)=LEN(L)
  RETURN
  END

```

FLOWCHART FOR ABSORB



APPENDIX 8

PRINT AND LPRINT SUBROUTINES

This appendix contains the FORTRAN listings for the PRINT and optional LPRINT subroutines. The PRINT subroutine gives output in the matrix form shown in the first two appendices. For matrices too large to be output in this manner, the LPRINT subroutine gives only the column indices of the nonzeros in each row.

88

```

SUBROUTINE PRINT(IC,M,N,IPR,ICN,LICN,IP,LENR)
DIMENSION ICN(LICN),IP(M),LENR(M),IPR(M)
IF(IC.EQ.1) GO TO 10
IF(IC.EQ.2) GO TO 20
IF(IC.EQ.3) GO TO 30
IF(IC.EQ.4) GO TO 45
IF(IC.EQ.10) GO TO 47
10  WRITE(6,100)
    GO TO 35
20  WRITE(6,200)
    GO TO 35
30  WRITE(6,300)
    GO TO 35
45  WRITE(6,350)
    GO TO 35
47  WRITE(6,355)
    GO TO 35
100  FORMAT('1','THE INITIAL FORM OF THE MATRIX B IS:')
200  FORMAT('1','AFTER CALLING THE SUBROUTINE MC21A:')
300  FORMAT('1','AFTER CALLING THE SUBROUTINE MC13D:')
350  FORMAT('1','AFTER CALLING THE SUBROUTINE ABSORB THE'
*    , ' FINAL FORM IS:')
355  FORMAT('1','AFTER CALLING THE SUBROUTINE MIN:')
35  DO 60 L=1,M
    IPL = IP(L)
    IPLL = IPL + LENR(L) - 1
    DO 65 K = 1,N
65    IPR(K) = 0
    DO 70 K = IPL,IPLL
70    IPR(ICN(K))=1
    WRITE(6,400)(IPR(MM),MM=1,N)
400  FORMAT(1X,100I1)
60  CONTINUE
    RETURN
    END
SUBROUTINE LPRINT(LENR,IP,ICN,LICN,M,N)
DIMENSION LENR(M),IP(M),ICN(LICN)
DO 10 II=1,M
    LENGTH = LENR(II)
    IPL=IP(II)
    IPLL=IPL + LENGTH - 1
    WRITE(6,100) II
100  FORMAT(1X,'THE NONZEROS IN ROW',I5,' ARE '
1    , 'FOUND IN THE FOLLOWING COLUMNS:')
    WRITE(6,200)(ICN(L),L=IPL,IPLL)
200  FORMAT(1X,100I5)
    10  CONTINUE
    RETURN
    END

```

APPENDIX 9

MAIN PROGRAM

This appendix contains the FORTRAN listing for the main program which calls all subroutines in Appendices 3 through 9 to permute a rectangular matrix to a block upper trapezoidal form.

```

      DIMENSION I(450),J(450),ICN(450),IP(75) ,LENR(75) ,
1     IPR(75) ,IPR(75) ,IW(75,4) ,IB(75)
1     ,IPR(75) ,IPR(75) ,LEN(75) ,IPP(75) ,NUORD(75),
1     MARK(75),IBF(75),LINK(75)
      INIECEK RLINK(75)
      IG=1

C
C     READ THE ROW INDEX AND COLUMN INDEX OF THE MATRIX.
C
      READ(5,100)M,N
100    FORMAT(2I5)
      DO 5 II=1,M
      MARK(II)=0
      LENR(II)=0
      LICN=0
      K = 1

C
C     READ THE ROW INDEX AND COLUMN INDEX
C     OF THE NEXT NONZERO.
C
10    READ(5,200) I(K),J(K)
      IF(I(K).EQ.0) GO TO 25

C
C     AS THE INDICES OF THE NONZERGES ARE READ, THE
C     VECTORS ICN, LENR, AND IP ARE FILLED.
C
200    FORMAT(2I5)
      LICN=LICN+1
      ICN(K) = J(K)
      LENR(I(K)) = LENR(I(K)) + 1
      IF(K.NE.1) GO TO 15
      IP(I(K)) = K
      GO TO 20
15    IF(I(K).NE.I(K-1)) IP(I(K)) = K
      K = K + 1
      GO TO 10

C
C     PRINT THE INITIAL FORM OF THE MATRIX.
C
25    IF(N.GT.150) CALL LPRINT(LENR,IP,ICN,LICN,M,N)
      IF(N.LE.150) CALL PRINT(IG,M,N,IPR,ICN,LICN,IP,LENR)

C
C     PERMUTE THE MATRIX TO ACHIEVE A ZERO-FREE MAIN
C     DIAGONAL AND PRINT THE MATRIX.
C
      CALL PC21A(M,N,LEN,IPP,ICN,LICN,IP,LENR,IPERM,
1     NUORD,IK)
      IG=2
      IF(N.GT.150) CALL LPRINT(LENR,IP,ICN,LICN,M,N)
      IF(N.LE.150) CALL PRINT(IG,M,N,IPR,ICN,LICN,IP,LENR)

```

```

C      FIND THE SPARSEST POSSIBLE UPPER SQUARE SUBMATRIX.
C
C      CALL MINIM(N,N,M,LICN,LINK,PLINK,LENR,IP,ICN,MARK)
      IG = 10
      IF(N.GT.150) CALL LPRINT(LENR,IP,ICN,LICN,M,N)
      IF(N.LE.150) CALL PRINT(IG,M,N,IPR,ICN,LICN,IP,LENR)
C
C      BLOCK UPPER TRIANGULARIZE THE UPPER SQUARE SUBMATRIX.
C
C      CALL MC130(M,N,LEN,IPP,ICN,LICN,IP,LENR,IGR,
1      IG,NUM,IA)
      IG=3
      IF(N.GT.150) CALL LPRINT(LENR,IP,ICN,LICN,M,N)
      IF(N.LE.150) CALL PRINT(IG,M,N,IPR,ICN,LICN,IP,LENR)
      IF(NUM.EQ.1) GO TO 99
C
C      PERMUTE THE BLOCKS ALONG THE DIAGONAL TO ACHIEVE AN
C      EVENTUAL DUAL ANGULAR FORM IF POSSIBLE.
C
      CALL PIMBLK(M,N,LICN,NUM,IB,ISR,IP,LENR,ICN,MARK,
1      NUORD,LEN,IPP)
C
C      PROCESS THE ROWS BELOW THE UPPER SQUARE SUBMATRIX
C      TO ACHIEVE A FINAL BLOCK UPPER TRAPEZOIDAL FORM.
C
      CALL ABSORB(M,N,LENR,IP,IB,ICN,LICN,NUM,IFR,IBR,IPP,
1      LEN)
      IG=4
      IF(N.GT.150) CALL LPRINT(LENR,IP,ICN,LICN,M,N)
      IF(N.LE.150) CALL PRINT(IG,M,N,IPR,ICN,LICN,IP,LENR)
99  CONTINUE
      STOP
      END

```

VITA

James F. Litsey was born in Owensboro, Kentucky, on November 11, 1955. He attended the public schools in Daviess County, Kentucky, and graduated from Daviess County High School in 1973. He entered the University of Kentucky in 1973 and graduated in December of 1977 with a degree of Bachelor of Arts with a major in Mathematics. He taught at Lafayette High School in Lexington, Kentucky, until July 1978.

In September of 1978 he entered graduate school at The University of Tennessee, Knoxville and received the Master of Science degree with a major in Mathematics in December of 1980.