To the Graduate Council:

I am submitting herewith a thesis written by David Hume entitled "Ordering Sparse Least Squares Problems." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert J. Plemmons, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

ORDERING SPARSE LEAST SQUARES PROBLEMS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

David Hume

August 1981

ACKNOWLEDGMENTS

It has been an honor and a pleasure to work with Robert J. Plemmons, the guiding force behind this project. I am particularly grateful to Mike Heath and James Litsey for discussions of their work. Special thanks is due to Bruce Delaney and Alice Beauchene who were always patient, courteous, and interested when I came by for help. Once again, Beth Kelley has done a super job of typing; it is a pleasure to work with her. I would also like to thank Terry Feagin and David Straight for serving on my committee.

I would like to acknowledge the following people (in no particular order) who have provided moral support for my program this year: John Bradley, Rhonda Wooten, Venki, Sheila Mooney, Reese Jacobs, Wallace Mayo, Kusum Soni, Steve Forbes, Scott Keith, Janice Julian, Alice Gordon, Chappy Chapman, Jake Beard, Steve Khleif, Larry Neal, Lisa Baker, Trish Holden, Joe Holden, Susie Hamby, Kay Taylor, Harold Kilpatrick, Robert McConnel, Art Fraas, and all the Humes.

ABSTRACT

Relative performances of three methods for ordering a large sparse least squares problem $Ax = b$, where A is $m \times n$, $m \geq n$, and of rank n, are compared. A system $Ax = b$ is ordered by finding permutation matrices P and Q so that the equivalent reordered system $PAQ = Pb$ suffers little fill-in when forming its Cholesky factor. An ordering method attempts to exploit the sparsity of A by reducing fill-in, operation counts, and execution time required for solving the system $Ax = b$. The system $Ax = b$ is ordered by minimum degree ordering, nested dissection, and block trapezoidal ordering prior to its solution using Givens reduction to compute the Cholesky factor.

Software for implementing the ordering methods is based upon a version of SPARSPAK, a package of sparse matrix routines, modified by George and Heath to implement the Givens reduction process. Routines of Duff, Reid, and Litsey are used to implement block trapezoidal ordering. Results obtained from test problems show that all three ordering methods greatly reduce fill-in, operation counts, and factorization times for solving $Ax = b$. In the case of an 892×261 problem, fill-in, operation counts, and factorization times were reduced 85% to 90% from the amounts required when the problem is solved without any ordering.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	Adjustment of Geodetic Data	1
	Ordering Sparse Least Squares Problems	5
II.	MATHEMATICAL BACKGROUND	8
	The Linear Least Squares Problem	8
	Large Sparse Systems of Linear Equations	10
	Large Sparse Least Squares Problems	13
	Givens Transformations	14
	Use of Givens Transformations in Sparse Least Squares Problems	20
	Minimum Degree Ordering	22
	Nested Dissection	26
	Block Trapezoidal Ordering	31
	Alternative Method for Achieving Block Trapezoidal Form	35
III.	COMPUTER IMPLEMENTATION OF ORDERING OPTIONS	39
	Software Used	39
	Data Structures for Sparse Matrices	40
	General Organization of Computer Programs	42
	Implementation of Minimum Degree Ordering	44
	Implementation of Nested Dissection Ordering	48
	Implementation of No Ordering	48
	Implementation of Block Trapezoidal Ordering	49
IV.	RESULTS	52
	Basis for Comparison of Ordering Methods	52
	Test Results	54
	Discussion of Test Results	58
	Conclusions	60
LIST OF REFERENCES		62
APPENDICES		64
1.	Stages of Block Trapezoidal Ordering	65
2.	Structure of $P(A^TA)P^T$ After Three Ordering Methods	69
VITA		72

LIST OF TABLES

TABLE		PAGE
1.	Outline of Code for First Job Step	45
2.	Main Driver Program	47
3.	By-Pass Subroutine for No Ordering	49
4.	Block Trapezoidal Ordering Code	50
5.	Storage Requirements for Four Test Problems	55
6.	Fill-in and Operation Counts for Four Test Problems	56
7.	Execution Times for Four Test Problems	57

LIST OF FIGURES

FIGURE		PAGE
1.	Geodetic coordinates on the surface of the earth	1
2.	Typical geodetic network of observations	3
3.	Stations blocked within a geodetic network	4
4.	Block structure of matrix representing a geodetic network	5
5.	Scheme for reducing rows of A by Givens transformations	16
6.	Result of applying Givens transformations to the system Ax = b	16
7.	Example of minimum degree ordering	25
8.	Graph elimination process to achieve minimum degree ordering	27
9.	Example of block trapezoidal ordering scheme	38
10.	SPARSPAK and Harwell schemes for storing a sparse matrix	42
11.	Schematic view of program	43

CHAPTER I

INTRODUCTION

Adjustment of Geodetic Data

Very large systems of equations arise in the context of computing coordinates of points on the surface of the earth. Geodetic data results from observations which attempt to map accurately the relative positions of many observation stations on a large portion of the earth's surface. A basic system of rectangular coordinates for these points may be defined with reference to an ellipsoid of revolution which best fits the earth. The reference ellipsoid has center (x_0, y_0, z_0) where x_0, y_0, z_0 are coordinates with respect to the geocenter--the earth's center of mass. Each station has three geodetic coordinates, latitude ϕ, longitude λ, and height h above the reference ellipsoid, as indicated in Figure 1. Since the earth is not an ideally rigid body,

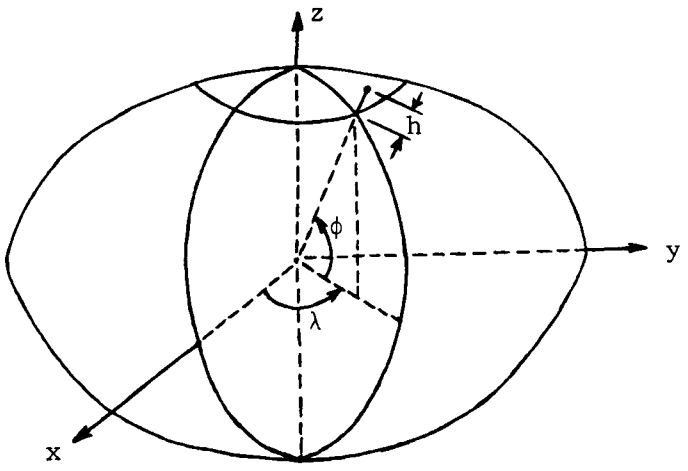


Figure 1. Geodetic coordinates on the surface of the earth.

it deforms under the action of crustal plate movements, resulting in shifts of masses which change the geocenter. Also, greater accuracy in measurement is now possible. Thus, adjustments of geodetic data need to be made periodically. For example, the North American Datum of 1927 will be adjusted to the 1983 North American Datum.

Generation and adjustment of geodetic data is achieved by measuring the relative position of pairs of observation stations on a portion of the earth's surface. These observations are often made by a field survey party which takes measurements relating a group of stations in close proximity. Each observation between two stations results in an equation relating the unknown coordinates of those two stations. There are approximately 225,000 stations involved in the 1983 North American Datum, which means that there are 450,000 unknown coordinates to be determined for latitude and longitude alone. Several million observations relating these stations have been made. After the observation equations have been linearized, a very large overdetermined system of linear equations results. We may think of this system in terms of a matrix equation $Ax = b$ in which each column of A corresponds to an unknown coordinate and each row of A is an equation which represents an observation between two stations. This system of equations is solved in a least squares sense to provide values for the unknown coordinates which are as consistent as possible with all observations. The least squares solution may also be used to locate observations that are badly in error or need re-measurement.

Note that the matrix A generated for this system is very sparse since each row of the matrix contains only a few nonzero elements

corresponding to unknowns associated with the stations related by that observation; the rest of the row contains zero elements. In addition, each station is normally related to only a few neighboring stations so that the large system may be broken down into smaller blocks pertaining to specific geographical areas. The process of breaking the total geodetic network down into several partial blocks has been termed "Helmert blocking" by geodesists. Subdivision into blocks can be done with respect to 1) regional partitioning--grouping geographically adjacent stations together, or 2) special type of observation--some observations may comprise the primary control network while others are part of secondary surveys (such as those for urban areas) which are based upon the primary network. As an example (see Figure 2), a network may look like the following picture in which stations are connected by observations.

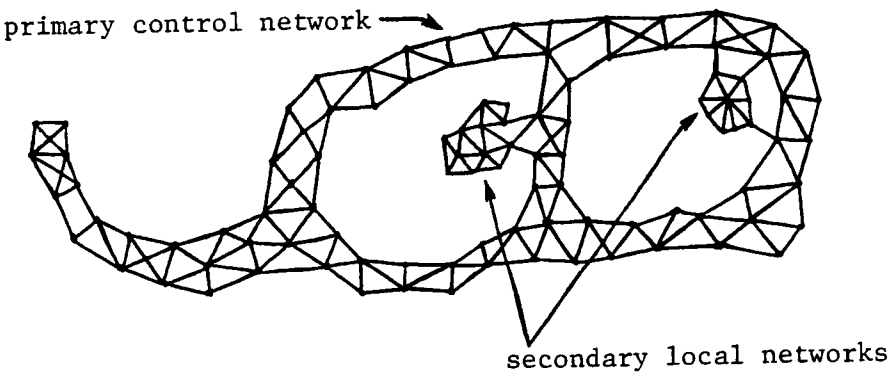


Figure 2. Typical geodetic network of observations.

In order to take advantage of the sparsity of the system $Ax = b$ which arises from observations within a geodetic network, the system may be

ordered (that is, the stations numbered) so that large zero blocks and small nonzero blocks corresponding to secondary local networks are produced in A. Although data may be naturally arranged in a way to facilitate blocking of the system, specific ordering techniques aimed at exploiting sparsity in a matrix can lead to great savings in storage required for the system and number of operations needed to solve the system. In addition, such blocking of the problem permits a large problem to be solved in stages as a sequence of smaller subproblems.

Suppose the stations in the above network are divided into ten blocks, numbered 1 through 10, as shown below in Figure 3 (dashed lines indicate boundaries for the blocks).

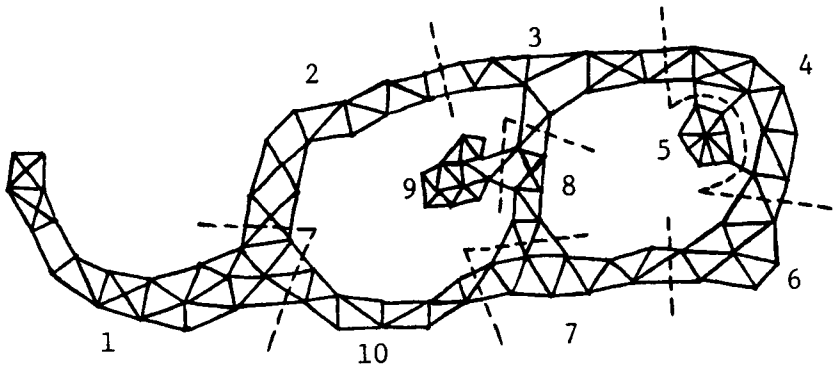


Figure 3. Stations blocked within a geodetic network.

Now, suppose stations in the network are numbered so that all stations in block 1 are first, all stations in block 2 are next, and so forth. Note that this numbering determines a column ordering for the matrix A

generated for this network. The block structure of A is shown below in Figure 4. Blocks are labeled horizontally and vertically. A nonzero rectangular block B_{ij} is indicated and occurs whenever there is an observation relating a station in block i to a station in block j. For instance, there are observations relating stations in block 1 to stations in blocks 1, 2, and 10. Note the large zero blocks in A and the generally triangular structure of A.

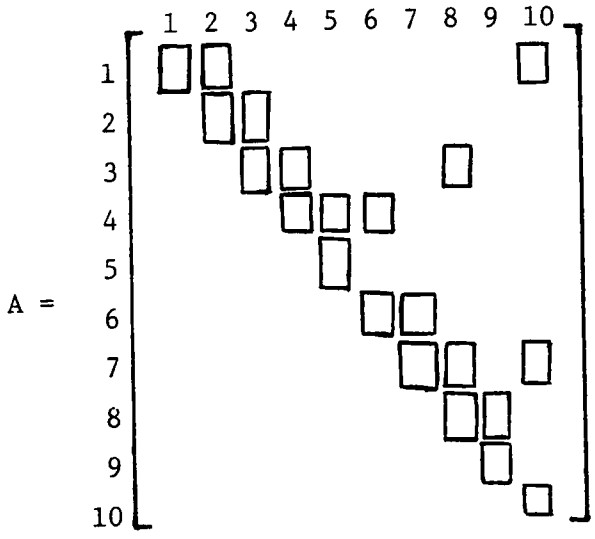


Figure 4. Block structure of matrix representing a geodetic network.

Ordering Sparse Least Squares Problems

This study deals with the question of how to order large sparse least squares problems $Ax = b$, where A is $m \times n$, $m \geq n$, and of rank n. As we have seen, such sparse systems arise in the context of geodetic network adjustment. (See U. S. National Oceanic and Atmospheric Administration [1978] for further details.) They also occur in

structural analysis problems using the finite element method and other areas. In mathematical terms, "ordering" a system $Ax = b$ means that one finds permutation matrices P and Q so that the equivalent reordered system $PAQ = Pb$ may be solved on a computer much more effectively than the original system, in the sense that the cost of storage and execution time is greatly reduced and the accuracy of the solution is better. Any ordering scheme seeks to exploit the sparsity of A by permuting it to take advantage of large zero blocks produced by the permutation. In the case of least squares problems $Ax = b$, we often wish to solve the system $A^TAx = A^Tb$; since A^TA is symmetric positive definite, ordering A^TA involves finding one permutation matrix P so that $P(A^TA)P^T$ has desirable properties with respect to sparsity considerations.

From a computational standpoint, a method for solving a least squares problem has three major phases: 1) order the system to take advantage of sparsity; 2) perform factorization of the system into triangular form; 3) obtain the solution by solving the triangular systems which result from step 2. Step 3 is approximately the same for any method. A central concern in Step 1 is to order the system so that the triangular form of the system resulting from factorization in Step 2 retains the degree of sparseness that was present in the original system. Thus, any ordering scheme attempts to minimize "fill-in" which occurs when zero positions in the original system become nonzeros in the factored triangular form of the system.

There are several schemes available for ordering a sparse system. In this project, the minimum degree, nested dissection, and

block trapezoidal ordering schemes are tested and compared for effectiveness in terms of reducing fill-in and operation counts involved in solving sparse systems. With all three ordering methods, ordering of the system is followed by a factorization process known as Givens reduction. The next three chapters describe, respectively, the mathematical ideas behind the methods used in this project, the software used to implement these methods, and the results obtained by solving several sparse least squares problems using these three ordering methods.

CHAPTER II

MATHEMATICAL BACKGROUND

The Linear Least Squares Problem

Suppose we have an overdetermined system of linear equations in which there may be more equations than unknowns. Such a system may be represented by the matrix equation

$$Ax = b$$

where A is an $m \times n$ matrix of rank n with $m \geq n$ and b is an $m \times 1$ vector. In order to solve this system in the least squares sense, we wish to determine an $n \times 1$ vector x so that the Euclidean norm $\| b - Ax \|_2$ of the residual vector $r = b - Ax$ is minimized.

We can write the vector b uniquely as $b = b_1 + b_2$ where $b_1 \in R(A)$ and $b_2 \in N(A^T)$. (See Stewart [1973, p. 215].) In order to obtain the desired solution x above, we wish to minimize the quantity $d^2 = \| b - Ax \|_2^2 = \| b_1 + b_2 - Ax \|_2^2 = \| b_1 - Ax \|_2^2 + \| b_2 \|_2^2$. Since the vector b_2 is a constant, the minimum value of d^2 is attained when the quantity $\| b_1 - Ax \|_2^2$ is minimized. But, $b_1 \in R(A)$ so we may solve $Ax = b_1$ to make $\| b_1 - Ax \|_2^2 = 0$. The solution x to $Ax = b_1$ is the least squares solution to $Ax = b$, and this solution is unique if A has rank n.

Note that since the solution to $Ax = b$ also satisfies $Ax = b_1$, we have that

$$A^T r = A^T(b - Ax) = A^T b_2 + A^T(b_1 - Ax) = 0$$

(since $b_2 \in N(A^T)$ and $b_1 - Ax = 0$). Thus, we see from the second member of the above equalities that $A^Tb = A^TAx$. The system of equations $A^TAx = A^Tb$ is called the system of normal equations for the least squares problem $Ax = b$. If A has rank n, then A^TA is symmetric positive definite, and the unique solution to the normal equations gives the solution to the associated least squares problem. The normal equations may be solved by using the Cholesky method to factor A^TA into R^TR, where R is upper triangular, and then solving the two triangular systems $R^Ty = A^Tb$ and $Rx = y$.

Another approach to solving the least squares problem $Ax = b$ is based upon obtaining an orthogonal factorization $QA = \begin{bmatrix} R \\ 0 \end{bmatrix}$, where Q is an $m \times m$ orthogonal matrix and R is $n \times n$ and upper triangular. The matrix Q may be obtained as a product of Givens transformations. According to this method, the orthogonal matrix Q is applied directly to A to obtain $QA = \begin{bmatrix} R \\ 0 \end{bmatrix}$ and, correspondingly, Q is applied to b to obtain $Qb = \begin{bmatrix} y \\ z \end{bmatrix}$, where y is an $n \times 1$ vector. The least squares solution is then obtained by solving the triangular system $Rx = y$. Note that $A = Q^T \begin{bmatrix} R \\ 0 \end{bmatrix}$ and that $A^TA = [R^T 0]Q \cdot Q^T \begin{bmatrix} R \\ 0 \end{bmatrix} = R^TR$, so the R obtained by this method is the same factor R which was obtained by the Cholesky method, except possibly for the signs of the rows. Similarly, note that since $b = Q^T \begin{bmatrix} y \\ z \end{bmatrix}$, we have $A^Tb = [R^T 0]Q \cdot Q^T \begin{bmatrix} y \\ z \end{bmatrix} = R^Ty$, or $R^Ty = A^Tb$;

therefore, the y obtained in this factorization method is the same as the y obtained by the Cholesky method. Thus, the x obtained by orthogonal factorization is the same solution which would be obtained using the normal equations. In both methods, the primary goal is to compute the Cholesky factor R.

Large Sparse Systems of Linear Equations

An $m \times n$ matrix $A = (a_{ij})$ is said to be sparse is sufficiently many a_{ij}'s are zero so that computations involving A take advantage of this situation. In the case of sparse systems $Ax = b$, where m and n are very large, considerations of storage requirements, numerical stability, and exploitation of sparsity assume great importance. For example, a system with 1000 equations and 500 unknowns may require up to $1000 \times 500 = 500{,}000$ locations to store the problem. However, if there are no more than 5 nonzero coefficients in each equation, then only 5000 locations are needed to store the problem; if one can maintain the sparsity of a system during the solution process, storage required may be reduced by a factor of 1/100. In the case of a very large problem, even if sparsity is exploited fully, it may be possible to store only part of the problem in main memory at one time; thus, the way in which A is accessed from auxiliary storage (by rows, for instance) may become important in terms of I/O cost.

A primary objective of any Cholesky method for solving a sparse system $Ax = b$, where A is symmetric positive definite, is to minimize the amount of fill-in which occurs. Fill-in occurs whenever the Cholesky factor R of A has nonzeros in positions which are zero in

A . Fill-in in R can be reduced dramatically by applying a symmetric permutation to the rows and columns of A . Such a reordering of A is called an ordering for A . To find a good ordering for A means to find a permutation matrix P $(P^T = P^{-1})$ so that PAP^T has a Cholesky factor which is just as sparse as the upper triangle of A ; that is, no fill-in occurs in passing to the Cholesky factor R which will be used to solve the system $Ax = b$.

An example will serve to illustrate this point. Let A be 1000×1000 and have the following form, where X denotes a nonzero element:

$$A = \begin{bmatrix} X & X & \cdots & X \\ X & X & & \\ \vdots & & \ddots & \\ X & & & X \end{bmatrix} .$$

Then the Cholesky factor for A is:

$$R_A = \begin{bmatrix} X & X & X & \cdots & X \\ & X & X & \cdots & X \\ & & X & & \\ & & & \ddots & \\ & & & & X \end{bmatrix} .$$

Now, A has about 3000 nonzeros, but R_A has 500,500 nonzeros. If permutation matrix

$$P = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ \vdots & & & \vdots \\ \cdot & 1 & 0 \cdots 0 & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & & \cdot \\ 0 & 0 & \cdots 0 & 1 \\ & & & \cdot \\ 1 & 0 & \cdots & 0 \end{bmatrix},$$

then

$$PAP^T = \begin{bmatrix} X & & & X \\ & X & O & \vdots \\ & & \ddots & \vdots \\ & O & & \vdots \\ & & X & X \\ X & \cdots & X & X \end{bmatrix} \quad \text{and} \quad R_{PAP^T} = \begin{bmatrix} X & & & X \\ & X & O & \vdots \\ & & \ddots & \vdots \\ & O & & \vdots \\ & & X & X \\ & & & X \end{bmatrix}.$$

After applying the symmetric permutation, the Cholesky factor R_{PAP^T} has only about 2000 nonzeros. Thus, a good ordering of the matrix A can result in great savings in storage and the number of operations required to solve a system.

In order to solve a sparse system $Ax = b$ (A symmetric positive definite) by a Cholesky-based method, four steps are generally involved:

1) (Ordering) Find a permutation matrix P so that the matrix $PAP^T = R^TR$ yields a sparse Cholesky factor R ; also, permute b to Pb .

2) (Allocation) Determine the zero-nonzero structure of the Cholesky factor R . Set up and allocate storage for a static data structure in which to store R .

3) (Factorization) Factor PAP^T into R^TR to compute numerical values for the entries of R.

4) (Triangular Solution) The original problem $Ax = b$ was reordered to obtain the equivalent system $PAP^TPx = Pb$, or $R^TRPx = Pb$. The solution x may be obtained by solving $R^Ty = Pb$ by forward substitution, $Rz = y$ by back substitution, and then setting $x = P^Tz$. (Thus, $Px = z$, $RPx = y$, and $R^TRPx = Pb$.)

In general, an ordering and data structure are chosen in such a way as to reduce computer storage and execution time. A good choice for P will reduce storage required by R; the data structure chosen for R can allow access to R by rows or by columns.

Large Sparse Least Squares Problems

We now return to the problem of solving a large least squares system $Ax = b$ in the context of sparsity considerations. If one works with the normal equations $A^TAx = A^Tb$, then A^TA is the coefficient matrix of the sparse system to be solved. The normal equations approach has the following advantages. The Cholesky method does not require pivoting for stability; this means that the matrix A^TA may be reordered on the basis of sparsity considerations alone. The matrix A^TA is independent of the order of the rows of A (since if the rows of A are reordered by a permutation P to get PA, we have that $(PA)^T(PA) = A^TP^TPA = A^TA$), so the rows of A may be processed in any order from a secondary storage file during the solution process; only part of A needs to be in main memory at any given time. The normal equations approach has these disadvantages: When the matrix A is

computed explicitly, there may be a loss in information unless costly extended precision is employed. Since the condition number of A^TA is the square of the condition number of A, the solution obtained may be inaccurate if A itself is poorly conditioned. Finally, the fact that A is sparse does not always mean that A^TA will be sparse.

The orthogonal factorization approach described earlier attempts to retain the advantages of the normal equations approach but avoid the problems associated with explicitly computing the matrix A^TA. In exchange for this advantage, one must face the problem of "intermediate fill-in" which is associated with reduction of A to $\begin{bmatrix} R \\ 0 \end{bmatrix}$ by applying a product Q of Givens transformations.

Givens Transformations

A Givens transformation "rotates" two row vectors of length n

$$x = [0, \ldots, 0, x_i, x_{i+1}, \ldots, x_k, \ldots]$$

$$y = [0, \ldots, 0, y_i, y_{i+1}, \ldots, y_k, \ldots]$$

and replaces them be row vectors

$$x' = [0, \ldots, 0, x'_i, x'_{i+1}, \ldots, x'_k, \ldots]$$

$$y' = [0, \ldots, 0, 0, y'_{i+1}, \ldots, y'_k, \ldots]$$

where $x'_k = cx_k + sy_k$, $y'_k = -sx_k + cy_k$, $i \leq k \leq n$, and $c = \dfrac{x_i}{\sqrt{x_i^2 + y_i^2}}$, $s = \dfrac{y_i}{\sqrt{x_i^2 + y_i^2}}$, $(c^2 + s^2 = 1)$. This transformation may be thought of in

terms of a plane rotation through an angle whose cosine and sine are given by c and s. The object of the transformation is to annihilate the leading nonzero entry of row vector y. Note that if zeros appear in x_k and y_k, then x_k' and y_k' remain zero. But, if either x_k or y_k is nonzero, then both x_k' and y_k' become nonzero (assuming no cancellation occurs). This fact leads to intermediate fill-in as we shall see.

A Givens transformation may be applied to two rows x and y of $m \times n$ matrix A by multiplying A on the left by the following orthogonal matrix Q_{ij}. Let x and y be the i^{th} and j^{th} rows of A respectively. Then Q_{ij} is the $m \times m$ matrix obtained by replacing the (i,i), (i,j), (j,i), and (j,j) entries of the $m \times m$ identity matrix by c, s, $-s$, and c respectively.

According to the orthogonal factorization technique for solving a least squares problem $Ax = b$, a sequence Q of Givens transformations is appplied to A to reduce A to $\begin{bmatrix} R \\ 0 \end{bmatrix}$ where R is the Cholesky factor of A. Also, the vector b is transformed by the same sequence to $\begin{bmatrix} y \\ z \end{bmatrix}$. Since $\| b - Ax \|_2 = \| Qb - QAx \|_2 = \left\| \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} x \right\|_2$, the minimum value of $\| b - Ax \|_2$ is obtained when $Rx = y$, and the quantity $\| z \|_2$ gives the root residual of the least squares problem.

The triangular matrix R is built up by processing rows of A one at a time and rotating each row into R, as shown below in Figure 5. One of two cases occurs with a given row a^i of A: (i) If the first nonzero element of a^i is in column j and row j of R has not

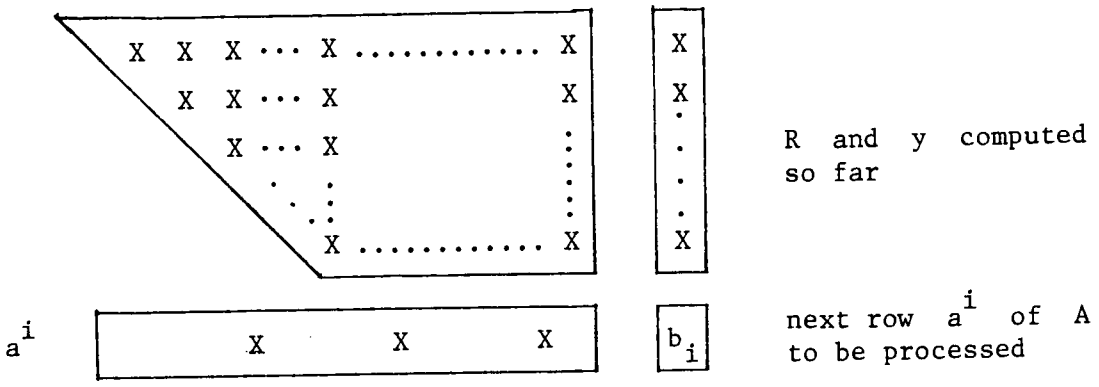


Figure 5. Scheme for reducing rows of A by Givens transformations.

been selected, then a^i is simply inserted into R as the j^{th} row and processing goes to the next row of A . (ii) Otherwise, leading nonzero elements of a^i are annihilated one-by-one using Givens transformations on the appropriate rows of R and a^i until the first case holds or until all nonzeros of a^i have been annihilated. After all the rows of A have been processed in this manner, the system $Ax = b$ has been transformed to the form shown in Figure 6 below.

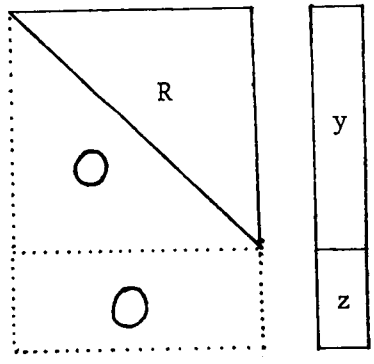


Figure 6. Result of applying Givens transformations to the system $Ax = b$.

An example will illustrate what is meant by intermediate fill-in which occurs during Givens reduction. Suppose R is currently given by the following triangular matrix, where an "X" indicates a nonzero element:

$$R = \begin{bmatrix} X & 0 & 0 & X & 0 & X & 0 & 0 & 0 \\ & X & X & 0 & X & 0 & 0 & 0 & 0 \\ & & & X & 0 & X & 0 & 0 & X \\ & & & & X & 0 & 0 & 0 & 0 \\ & & & & & X & X & 0 & 0 \\ & & & & & & X & X & 0 \end{bmatrix} .$$

Suppose the next row of A to be processed is

$$a^i = [\, 0 \;\; 0 \;\; X \;\; 0 \;\; 0 \;\; 0 \;\; X \;\; 0 \;\; 0 \,] .$$

Since the first nonzero element of a^i appears in the third column and the third row of R has not been selected, this row is inserted into R as its third row, and R becomes

$$R = \begin{bmatrix} X & 0 & 0 & X & 0 & X & 0 & 0 & 0 \\ & X & X & 0 & X & 0 & 0 & 0 & 0 \\ & & X & 0 & 0 & 0 & X & 0 & 0 \\ & & & X & 0 & X & 0 & 0 & X \\ & & & & X & 0 & 0 & 0 & 0 \\ & & & & & X & X & 0 & 0 \\ & & & & & & X & X & 0 \end{bmatrix} .$$

Let the next row of A to be processed be

$$a^{i+1} = [\, 0 \;\; X \;\; 0 \;\; 0 \;\; X \;\; X \;\; 0 \;\; 0 \;\; 0 \,] .$$

This row must have some of its nonzero elements eliminated by Givens rotations. The following picture indicates how the elimination process occurs:

$$R = \begin{bmatrix} X & 0 & 0 & X & 0 & X & 0 & 0 & 0 \\ & X & X & 0 & X & \oplus & 0 & 0 & 0 \\ & & X & 0 & \oplus & \oplus & X & 0 & 0 \\ & & & X & 0 & X & 0 & 0 & X \\ & & & & X & \oplus & \oplus & 0 & 0 \\ & & & & & X & X & 0 & 0 \\ & & & & & & X & X & 0 \end{bmatrix} .$$

$$a^{i+1} = [\ 0\ \otimes\ \oplus\ 0\ \otimes\ \otimes\ \oplus\ \oplus\ 0\ ]$$

An $\otimes$ means that a nonzero element of a^{i+1} was annihilated; $\oplus$ means that an element which was zero became nonzero during this step of the Givens process. Elements in a^{i+1} marked $\oplus$ reflect intermediate fill-in; these elements are ultimately annihilated to 0 again during this step, except for the one in column 8. That nonzero element becomes $r_{88} \in R$ when the transformed a^{i+1} becomes the eighth row of R. Elements in R marked $\oplus$ remain nonzero to the end of the Givens reduction process. In this step of Givens reduction, rows 2, 3, 5, 6, and 7 are successively rotated with a^{i+1}. Although it is not indicated, the same sequence of transformations is applied to the right-hand vector y. After a^{i+1} is incorporated into the triangular structure of R as the eighth row, R looks like

$$R = \begin{bmatrix} X & 0 & 0 & X & 0 & X & 0 & 0 & 0 \\ & X & X & 0 & X & X & 0 & 0 & 0 \\ & & X & 0 & X & X & X & 0 & 0 \\ & & & X & 0 & X & 0 & 0 & X \\ & & & & X & X & X & 0 & 0 \\ & & & & & X & X & 0 & 0 \\ & & & & & & X & X & 0 \\ & & & & & & & X & 0 \end{bmatrix} .$$

Processing would then continue with the next row of A.

The problem of intermediate fill-in may be described as follows. The zero-nonzero structure of R can be determined by a symbolic factorization of A^TA. (See George and Liu [1981, p. 142].) Thus, the necessary data structure in which to store R may be allocated in advance of computing R, and there is a location in that data structure for each nonzero element of R generated during the Givens reduction process. However, during each step of Givens reduction, many temporary nonzeros may be created in the current row of A being reduced (a^{i+1} in the previous example). That is, as leading nonzeros are annihilated, many other zero elements in a^{i+1} may be changed to nonzeros, requiring many additional computations to reduce them back to zeros. If this intermediate fill-in is severe, the effect will be to lose any advantage gained by attempting to exploit sparsity. The problem of intermediate fill-in can be confined to the current row of A being reduced when Givens transformations are employed, and intermediate fill-in may be lessened by permuting rows of A so that R remains sparse as it is built by Givens rotations.

Use of Givens Transformations in Sparse Least Squares Problems

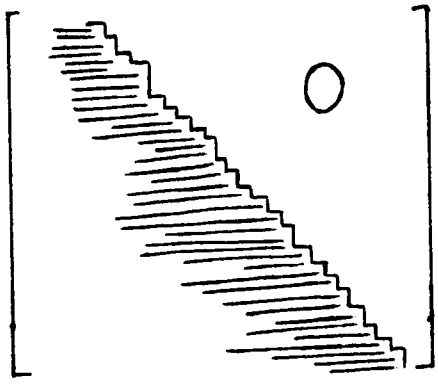
The problem of unacceptable intermediate fill-in has deterred the orthogonal factorization method from gaining popularity as a sparse least squares solver. George and Heath [1980] have developed a method for solving a large sparse least squares system $Ax = b$ which combines advantages of the normal equations approach (which uses Cholesky-based techniques) and the orthogonal factorization approach (using Givens transformations). The basic steps of this method are:

1) Determine the zero-nonzero structure of the matrix A^TA .
2) Find a good ordering P of A^TA so that P^TA^TAP has a sparse Cholesky factor R . The reordering of A^TA corresponds to a permutation AP of the columns of A .
3) Determine the zero-nonzero structure for R and allocate a row-oriented data structure for storing R .
4) Compute R using Givens transformations on the rows of A one at a time, as described in the preceding section.
5) Solve $Rz = y$ by back substitution (this involves accessing R by rows) and set $x = Pz$ (re-permute to the original column ordering of A).

Steps 1, 2, and 3 use well-developed techniques for solving sparse positive definite systems according to the normal equations approach. (See George and Liu [1981].) In particular, the method takes full advantage of ordering schemes for A^TA in these three steps. This method has further advantages: i) Since A^TA is not explicitly formed, potential numerical instability is avoided. ii) Storage requirements for a problem are the same as for the normal equations approach.

iii) A static data structure is used which allows efficiency in numerical calculations. iv) The use of row operations and row-oriented data structures allows efficient access of data from auxiliary storage. v) The rows of A may be reordered without regard to numerical stability or the ultimate sparsity of R ; thus, the rows of A may be reordered to reduce intermediate fill-in.

The George-Heath method is implemented as follows. The problem $Ax = b$ is stored by rows on an external file. Step 1 is accomplished by making a pass through the file and recording only the column subscripts of each nonzero element of each row of A ; from this information, the structure of A^TA is determined and stored in a graph. Step 2 (ordering A^TA) uses existing ordering routines available in the sparse matrix package SPARSPAK (George and Liu [1978]). The ordering routines work by relabeling nodes of the graph which represents the structure of A^TA ; any relabeling corresponds to a symmetric permutation of the rows and columns of A^TA . Step 3 is also accomplished by SPARSPAK routines. Prior to Step 4, the rows of A on the external file may be sorted according to a heuristic rule which attempts to minimize intermediate fill-in. For example, rows may be sorted in non-decreasing order with respect to maximum column subscript for a nonzero element in each row (column subscripts used are those of the permuted matrix AP); after this row sorting, AP looks like:

AP = 

The purpose of this row sort is to delay the amount of intermediate fill and reduce the number of operations required during factorization. In step 4, another pass is made through the external file to access the sorted rows of A in order to compute R. During this step, the floating point values of the nonzeros of A are read. R is stored in main memory as it is built up. A working vector of length n is utilized to help handle intermediate fill-in. After R is calculated, a SPARSPAK routine is used to calculate the solution x.

Minimum Degree Ordering

In the context of direct (versus iterative) methods for solving sparse systems $Ax = b$, where A is symmetric positive definite, the ordering step is of central importance for exploiting sparsity. We reduce storage and execution time requirements by finding P so that PAP^T has sparse Cholesky factor R. We shall look at three ordering schemes used in this project: minimum degree, nested dissection, and block trapezoidal ordering.

The motivation for the minimum degree ordering scheme comes from the Cholesky factorization process itself. If we are given an $n \times n$

symmetric positive definite matrix A, then according to the outer product form of Cholesky factorization we have

$$A = \begin{bmatrix} d_1 & v_1^T \\ v_1 & H_1 \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ \frac{v_1}{\sqrt{d_1}} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_1 - \frac{v_1 v_1^T}{d_1} \end{bmatrix} \begin{bmatrix} \sqrt{d_1} & \frac{v_1^T}{\sqrt{d_1}} \\ 0 & I_{n-1} \end{bmatrix},$$

or $A = R_1^T A_1 R_1$, where R_1 is upper triangular. The above factorization process may be repeated on A_1 (note that $H_1 - \frac{v_1 v_1^T}{d_1}$ is symmetric positive definite) and continued to the k^{th} step where we obtain

$$A_{k-1} = \begin{bmatrix} I_{k-1} & 0 & 0 \\ 0 & d_k & v_k^T \\ 0 & v_k & H_k \end{bmatrix}$$

$$= \begin{bmatrix} I_{k-1} & 0 & 0 \\ 0 & \sqrt{d_k} & 0 \\ 0 & \frac{v_k}{\sqrt{d_k}} & I_{n-k} \end{bmatrix} \begin{bmatrix} I_{k-1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & H_k - \frac{v_k v_k^T}{d_k} \end{bmatrix} \begin{bmatrix} I_{k-1} & 0 & 0 \\ 0 & \sqrt{d_k} & \frac{v_k^T}{\sqrt{d_k}} \\ 0 & 0 & I_{n-k} \end{bmatrix},$$

or $A_{k-1} = R_k^T A_k R_k$, where R_k is upper triangular. After n steps, the factorization $A = R_1^T R_2^T \cdot \cdots \cdot R_n^T \cdot R_n \cdot \cdots \cdot R_2 R_1 = R^T R$ is obtained. In view of the fact that $R = R_1 + R_2 + \cdots + R_n - (n-1)I_n$, we see that fill-in at the k^{th} step is confined to the last $n - k$ rows and columns of A_{k-1} and results from formation of the product $v_k v_k^T$.

At the beginning of the k^{th} step of Cholesky decomposition, A_{k-1} looks like

$$A_{k-1} = \left[\begin{array}{c|ccc} I_{k-1} & & 0 & \\ \hline & a_{kk} & \cdots & a_{kn} \\ 0 & \vdots & \bar{A}_{k-1} & \vdots \\ & a_{nk} & \cdots & a_{nn} \end{array}\right] .$$

In order to reduce the amount of fill-in which occurs in the k^{th} step, we find row j, $k \leq j \leq n$, with a minimum number of nonzeros and interchange rows and columns j and k. After this interchange, $\bar{A}_{k-1}$ looks like $\begin{bmatrix} d_k & v_k^T \\ v_k & H_k \end{bmatrix}$. Thus, we choose the diagonal element of a row with minimum number of nonzeros as the next pivot d_k in computing the Cholesky factor and expect the product $v_k v_k^T$ to generate the fewest number of nonzeros during the k^{th} step. The minimum degree algorithm thus provides local minimization of fill at each step of Cholesky factorization. Heuristically, the algorithm should greatly reduce fill-in in R, but it does not globally minimize the fill-in in R.

An example will serve to illustrate how the minimum degree algorithm works. Essentially, diagonal entries are picked in an order for which fill-in is minimized at each step. Two rows and columns are interchanged at each step. These interchanges may be done before or during Cholesky decomposition. In Figure 7 below, diagonal entries have been replaced by numbers which indicate their original row position in A. Entries marked "Φ" indicate fill-in which occurs during Cholesky reduction. The final ordering of the rows and columns is 4, 1, 5, 6, 2, 3, 7. These row and column permutations may be obtained

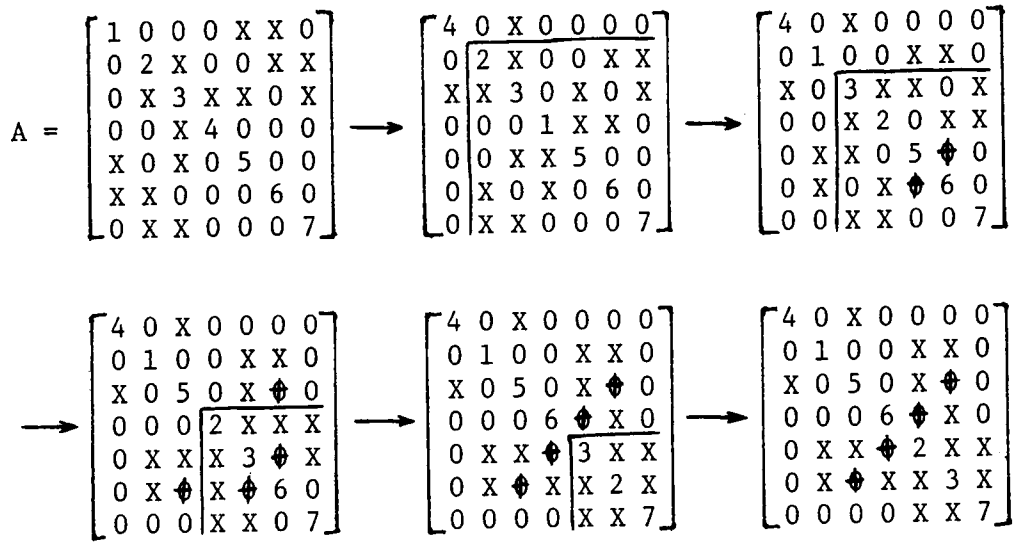
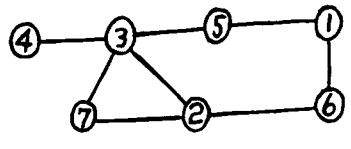


Figure 7. Example of minimum degree ordering.

by forming the product PAP^T where P is the permutation matrix

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} .$$

Minimum degree ordering may be implemented by means of a node elimination process performed on an adjacency graph G^A for symmetric matrix A. We illustrate with the above example. For 7×7 matrix A, there are seven nodes corresponding to the diagonal elements. If $a_{ij} \neq 0$, then there is an edge from node i to node j, and the two nodes are said to be adjacent. Thus, the graph of A is

$G^A =$ 

.

The degree of a node in G^A is the number of edges connected to that node. Above, the degree of node 3 is 4. The number of nonzero elements in row i of A is one greater than the degree of node i. Minimum degree ordering consists of successively picking node i in G^A of minimum degree and eliminating that node and its edges from the graph; this corresponds to picking the next pivot in Cholesky factorization. Although we minimize fill-in at this step, fill-in does occur whenever $a_{ij} \neq 0$, $a_{i\ell} \neq 0$, but $a_{j\ell} = 0$. (During the formation of $v_k v_k^T$, $a_{j\ell}$ will become nonzero.) In the context of G^A, this means that if nodes j and ℓ were connected to node i, then they must be connected directly by an edge after node i is eliminated and before one searches for what will currently be the next node of minimum degree. The order in which nodes are eliminated determines the same permutation P as before. The graph elimination process for the previous example is illustrated below in Figure 8. The final ordering of nodes is 4, 1, 5, 6, 2, 3, 7, the same as before.

Nested Dissection

The nested dissection scheme for ordering a large matrix A, where A is symmetric positive definite, attempts to determine a permutation P which breaks down the original problem $Ax = b$ into subproblems which can be solved separately and whose solutions can be used to solve the original problem. Additionally, since the subproblems have the same general structure as the original problem, they may be broken down recursively into even smaller subproblems until further subdivision of the original problem is no longer desirable. Once the

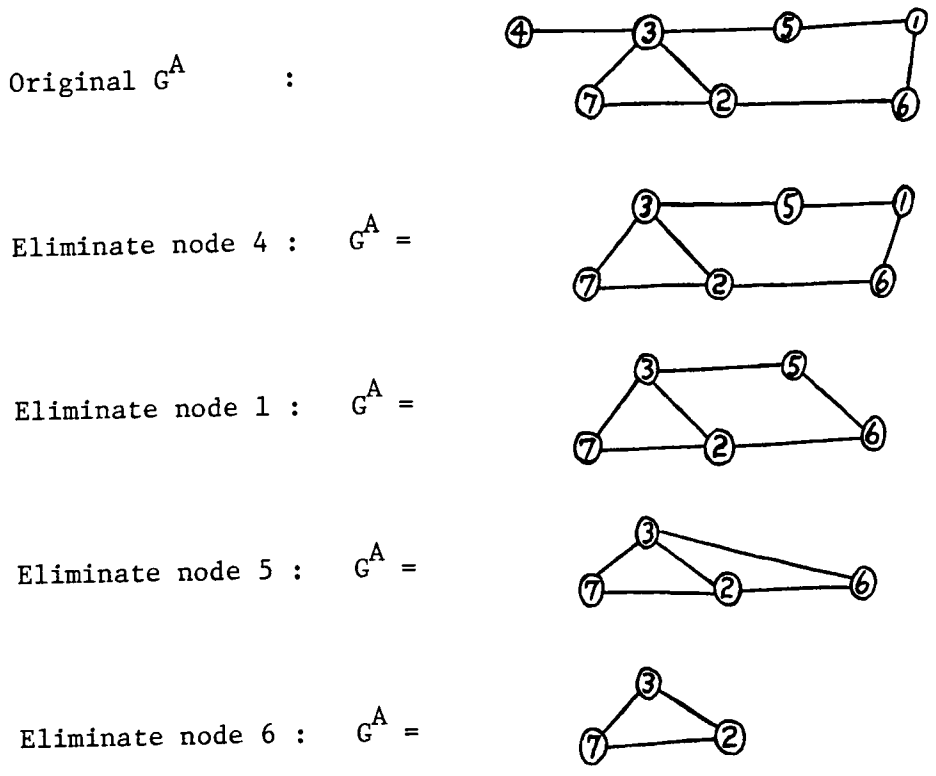
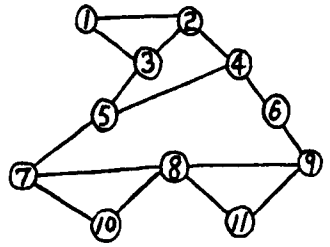


Figure 8. Graph elimination process to achieve minimum degree ordering.

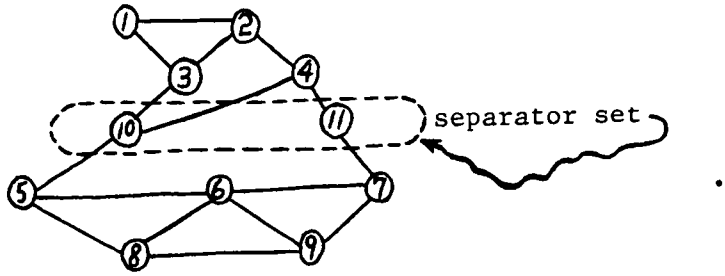
main problem is ordered, only part of the problem needs to be in main memory at one time, thus reducing storage required; this can be important in very large problems. The nested dissection method also seeks to minimize fill-in for the Cholesky factor R by creating large zero blocks in the permuted matrix PAP^T which are preserved as the matrix is factored; this reduces the number of operations required to compute R.

The key notion in the dissection process is the selection of a separator set of rows from A which divides the matrix into two

components. This choice provides the basis for breaking up the problem into smaller subproblems. Once again, the adjacency graph G^A associated with (symmetric positive definite) matrix A may be used to illustrate and implement the dissection process. In an example, we observe how a relabeling of G^A permutes the matrix A. Let G^A be given by

$G^A =$ 

Notice that the nodes of G^A (which correspond to columns of A) may be separated into two well-defined groups $A_1 = \{1, 2, 3, 4\}$ and $A_2 = \{7, 8, 9, 10, 11\}$ which are separated by the set of nodes $A_3 = \{5, 6\}$ in the sense that no node in A_1 is adjacent to any node in A_2. Now, reorder the nodes of G^A as follows: the nodes of A_1 first, the nodes of A_2 second, and the nodes of A_3 last. Thus, the nodes of G^A are numbered 1, 2, 3, 4, 7, 8, 9, 10, 11, 5, 6. The reordered graph looks like

$G^{PAP^T} =$ 

This reordering of nodes in G^{PAP^T} permutes A to PAP^T, where P is given below. Also shown below is the permuted matrix PAP^T:

$$P = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0 \end{bmatrix}$$

$$PAP^T = \left[\begin{array}{cccc|ccccc|cc} X&X&X&0&0&0&0&0&0&0&0 \\ X&X&X&X&0&0&0&0&0&0&0 \\ X&X&X&0&0&0&0&0&0&X&0 \\ 0&X&0&X&0&0&0&0&0&X&X \\ \hline 0&0&0&0&X&X&0&X&0&X&0 \\ 0&0&0&0&X&X&X&X&X&0&0 \\ 0&0&0&0&0&X&X&0&X&0&X \\ 0&0&0&0&X&X&0&X&X&0&0 \\ 0&0&0&0&0&X&X&X&X&0&0 \\ \hline 0&0&X&X&X&0&0&0&0&X&0 \\ 0&0&0&X&0&0&X&0&0&0&X \end{array}\right] .$$

The choice of a small separator set of nodes which divides the graph G^A into two pieces of approximately equal size is the important aspect of the dissection process. Notice that A has been permuted into the form

$$PAP^T = \begin{bmatrix} A_1 & 0 & A_{13} \\ 0 & A_2 & A_{23} \\ A_{13}^T & A_{23}^T & A_3 \end{bmatrix}$$

and that two large zero blocks have been created. The important consequence of this block partitioned form of A is that these zero blocks will remain zero during Cholesky factorization.

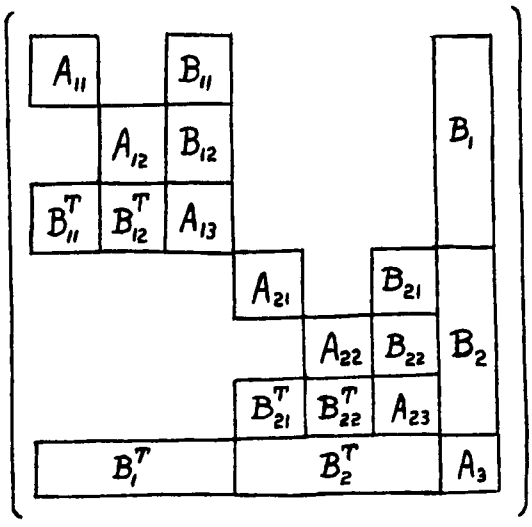
In order to obtain the Cholesky factor R of PAP^T, we must solve

$$\begin{bmatrix} A_1 & 0 & A_{13} \\ 0 & A_2 & A_{23} \\ A_{13}^T & A_{23}^T & A_3 \end{bmatrix} = \begin{bmatrix} R_1^T & 0 & 0 \\ 0 & R_2^T & 0 \\ R_{13}^T & R_{23}^T & R_3^T \end{bmatrix} \begin{bmatrix} R_1 & 0 & R_{13} \\ 0 & R_2 & R_{23} \\ 0 & 0 & R_3 \end{bmatrix}, \text{ or } PAP^T = R^T R .$$

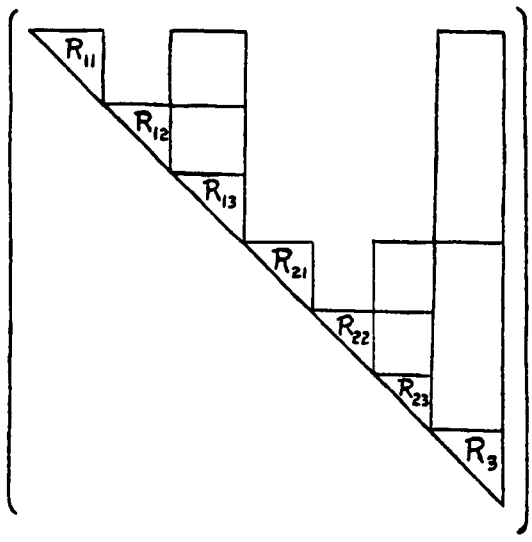
Note that PAP^T is positive definite and hence its principal submatrices A_1, A_2, and A_3 are positive definite. Thus, we may obtain

$A_1 = R_1^T R_1$ and $A_2 = R_2^T R_2$ by Cholesky factorization. Next, solve $R_1^T R_{13} = A_{13}$ and $R_2^T R_{23} = A_{23}$ to obtain R_{13} and R_{23}. Last, we need to solve $R_{13}^T R_{13} + R_{23}^T R_{23} + R_3^T R_3 = A_3$ for R_3; but, this can be done by obtaining R_3 as the Cholesky factor of the positive definite matrix $A_3 - R_{13}^T R_{13} - R_{23}^T R_{23}$. Thus, the zero block in the upper triangle of PAP^T is preserved in R.

The dissection process may be applied recursively. If dissection is applied to subblocks A_1 and A_2 in the above example, we achieve two levels of dissection and obtain a matrix block partitioned in the form



whose Cholesky factor has the form



George and Liu [1981, pp. 260-261] give a good illustration of four levels of nested dissection. As mentioned before, dissection may continue until further subdivision is no longer necessary or fruitful. Golub and Plemmons [1980] present an interesting account of dissection in terms of geographic blocking of stations involved in geodetic survey problems.

Block Trapezoidal Ordering

An $m \times n$ matrix A of rank n, where $m \geq n$ is said to be in block (upper) trapezoidal form if it has the following structure:

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

where each A_{ii}, $1 \leq i \leq k$, is a block of dimension $m_i \times n_i$. In the context of a least squares problem, if the problem can first be

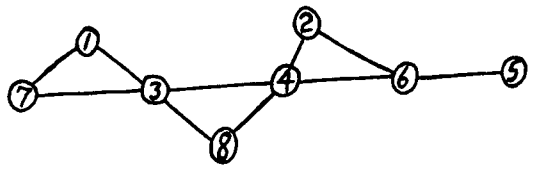
permuted to $Ax = b$ where A is in the block upper trapezoidal form given above, then orthogonal factorization can be applied separately to each row of blocks $A_{ii} \cdots A_{ik}$ in A, for $i = 1, 2, \cdots, k$ successively. This sequence of factorizations will result in the factorization $QA = \begin{bmatrix} R \\ 0 \end{bmatrix}$, where R is upper triangular, but the method requires that only certain blocks of A be in main memory at any one time, thus saving storage. George, Heath, and Plemmons [1981] describe how this method can be used in conjunction with auxiliary storage files to solve very large sparse least squares problems efficiently while using a relatively small amount of main storage.

Block trapezoidal form may be achieved by an incomplete nested dissection partitioning scheme. Given a sparse least squares system $Ax = b$, where A is $m \times n$, $m \geq n$, and of rank n, form the adjacency graph G for the matrix A^TA. Then, perform a nested dissection process on G to partition its nodes into several disjoint component sets; this partition of nodes of G corresponds to a partition of the columns of A. Now, relabel the nodes of G so that the nodes in each component set are numbered consecutively, and the component sets themselves are numbered in the reverse order from the order in which they were selected. This relabeling corresponds to a permutation AQ of the columns of A. Finally, the partitioning of the columns of A induces a partitioning of the rows of AQ so that if the rows of AQ are grouped according to successive components of this partition, a block upper trapezoidal form for A results. We illustrate this procedure in more detail with an example. (Further details may be found in George, Heath, and Plemmons [1981].)

Let 12×8 matrix A be given by

$$A = \begin{bmatrix} X & 0 & 0 & 0 & 0 & 0 & X & 0 \\ 0 & X & 0 & 0 & 0 & X & 0 & 0 \\ 0 & X & 0 & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & X & 0 \\ X & 0 & X & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & X & 0 & X & 0 & 0 \\ 0 & 0 & 0 & X & 0 & 0 & 0 & X \\ 0 & 0 & X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & 0 & X \\ 0 & 0 & 0 & 0 & X & 0 & 0 & X \end{bmatrix} .$$

Then, the graph of A^TA is given by

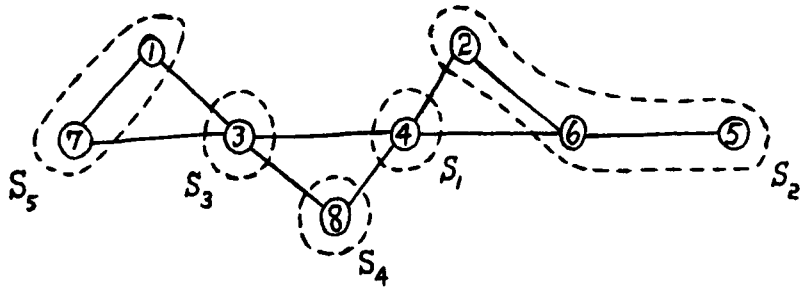
$G =$ 

.

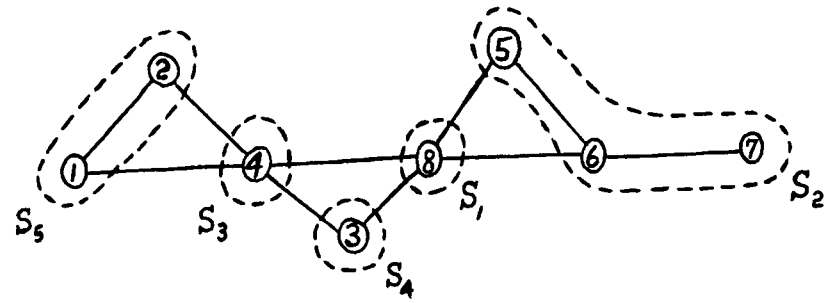
We now perform nested dissection on G in the following way:

1. Let V be the set of all nodes of G. Set $p = 1$.
2. If $V = \phi$, stop the dissection process. Otherwise, pick a connected component T of V. If the number of nodes in T is less than or equal to μ, let $S_p = T$ (select a final component of the dissection process); otherwise, let S_p be a minimal separator set of nodes for T which disconnects T into components of approximately equal size.
3. Set $V = V - S_p$, $p = p + 1$, and go to Step 2.

In our example, pick $\mu = 3$. The number μ controls the completeness (or incompleteness) of the dissection in that it specifies the maximal size of any final connected component. Whenever any connected component has more than 3 nodes, we dissect it further. Dissection in this case is shown below:



There are five component sets which result from this partition of the nodes of G. We now relabel nodes of G by taking the nodes from each of S_5, S_4, S_3, S_2, S_1 in that order; thus, we get the nodes in the order 7, 1, 8, 3, 2, 6, 5, 4. The relabeled graph is



In matrix A, this relabeling means that column 7 becomes column 1, column 1 becomes column 2, and so forth. Thus, the columns of A are permuted to obtain

$$AQ = \begin{bmatrix} X & X & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & 0 & 0 & X & 0 & 0 & X \\ 0 & 0 & 0 & X & 0 & 0 & 0 & X \\ X & 0 & 0 & X & 0 & 0 & 0 & 0 \\ 0 & X & 0 & X & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & X & 0 & X \\ 0 & 0 & X & 0 & 0 & 0 & 0 & X \\ 0 & 0 & 0 & X & 0 & 0 & 0 & X \\ 0 & 0 & 0 & 0 & 0 & X & X & 0 \\ 0 & 0 & X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 & X & 0 \end{bmatrix}.$$

Let $|S_p|$ denote the cardinality of S_p. Now, order the rows of AQ from top to bottom in the following way: all rows with nonzeros in the first $|S_5|$ columns, then all rows with nonzeros in the next $|S_4|$ columns, then the same for the next $|S_3|$ columns, and so forth until there are no more rows left. As we see below, the matrix A will have been permuted into block upper trapezoidal form after this row ordering: AQ becomes

$$PAQ = \begin{bmatrix} X & X & 0 & 0 & 0 & 0 & 0 & 0 \\ X & 0 & 0 & X & 0 & 0 & 0 & 0 \\ 0 & X & 0 & X & 0 & 0 & 0 & 0 \\ & & X & 0 & 0 & 0 & 0 & X \\ & & X & X & 0 & 0 & 0 & 0 \\ & & X & 0 & 0 & 0 & X & 0 \\ & & & X & 0 & 0 & 0 & X \\ & & & X & 0 & 0 & 0 & X \\ & & & & X & X & 0 & 0 \\ & & & & X & 0 & 0 & X \\ & & & & 0 & X & 0 & X \\ & & & & 0 & X & X & 0 \end{bmatrix} .$$

Alternative Method for Achieving Block Trapezoidal Form

An alternative method for permuting an $m \times n$ matrix A of rank n to block upper trapezoidal form is based upon obtaining a transversal of length n for the matrix. A transversal for a matrix is a set of nonzero elements of the matrix, no two of which are from the same row or column. The length of a transversal is the number of elements contained in it. A matrix of rank n has a transversal of length n.

This alternative method was developed by Duff [1978] and Duff and Reid [1978] and was extended and implemented by Litsey [1980] for the case $m \geq n$. The procedure for permuting A to block upper

trapezoidal form consists of three steps:

1) Pick a transversal of A of length n and permute the rows of A so that the transversal elements occupy the main diagonal entries of the first n rows of A. That is, find permutation matrix R so that

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

where $n \times n$ matrix A_1 has a zero-free main diagonal.

2) Permute the matrix A_1 to block upper triangular form. A matrix B is block upper triangular if it has the form

$$B = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1k} \\ 0 & B_{22} & \cdots & B_{2k} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & B_{kk} \end{bmatrix},$$

where each diagonal block is square. Since A_1 has a zero-free diagonal, we may find an $n \times n$ permutation matrix Q so that $Q^T A_1 Q$ is in block upper triangular form with indecomposable square diagonal blocks. A matrix C is <u>indecomposable</u> if there are no permutation matrices P_1 and P_2 so that

$$P_1 C P_2 = \begin{bmatrix} C_1 & D \\ 0 & C_2 \end{bmatrix},$$

where C_1 and C_2 are square. Thus, an indecomposable block cannot be broken down into smaller square blocks by further permutation.

Once Q above is found, we apply the following transformation to RA to obtain

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

where A_1' is block upper triangular with indecomposable diagonal blocks and A_2' results from a permutation of the columns of A_2.

3) The rows of A_2' are now permuted upward into the diagonal blocks of A_1'; thus, the diagonal blocks become rectangular blocks with no nonzero elements beneath them. For each row in A_2', the column of the first nonzero element is determined and that row is inserted into A_1' immediately above the row containing the transversal element corresponding to that column. When all the rows of A_2' are absorbed into A_1', the resulting matrix is in block upper trapezoidal form.

There is an additional optional step which may be applied in conjunction with or after step 1 above. Since A may have more than one transversal, the matrix A_1 obtained in step 1 is not unique. At the end of step 1, one may permute the rows of RA to obtain another A_1 which has a minimum number of nonzeros yet maintains a zero-free diagonal. This new A_1 with fewest nonzeros is desirable for the following heuristic reason: such an A_1 is obtained by a technique which produces few edges in the graph associated with the nodes of A_1; therefore, the graph tends to have smaller connected components. This fact, in turn, means that more and narrower square diagonal blocks will be produced in step 2 when A_1 is permuted to A_1'. Thus, there should

be more zeros below the diagonal blocks in A_1' . This alternate step was implemented by Litsey [1980].

An example shown below in Figure 9 illustrates how the alternative ordering scheme to achieve block upper trapezoidal form works. Transversal elements are circled in the matrix RA after step 1 .

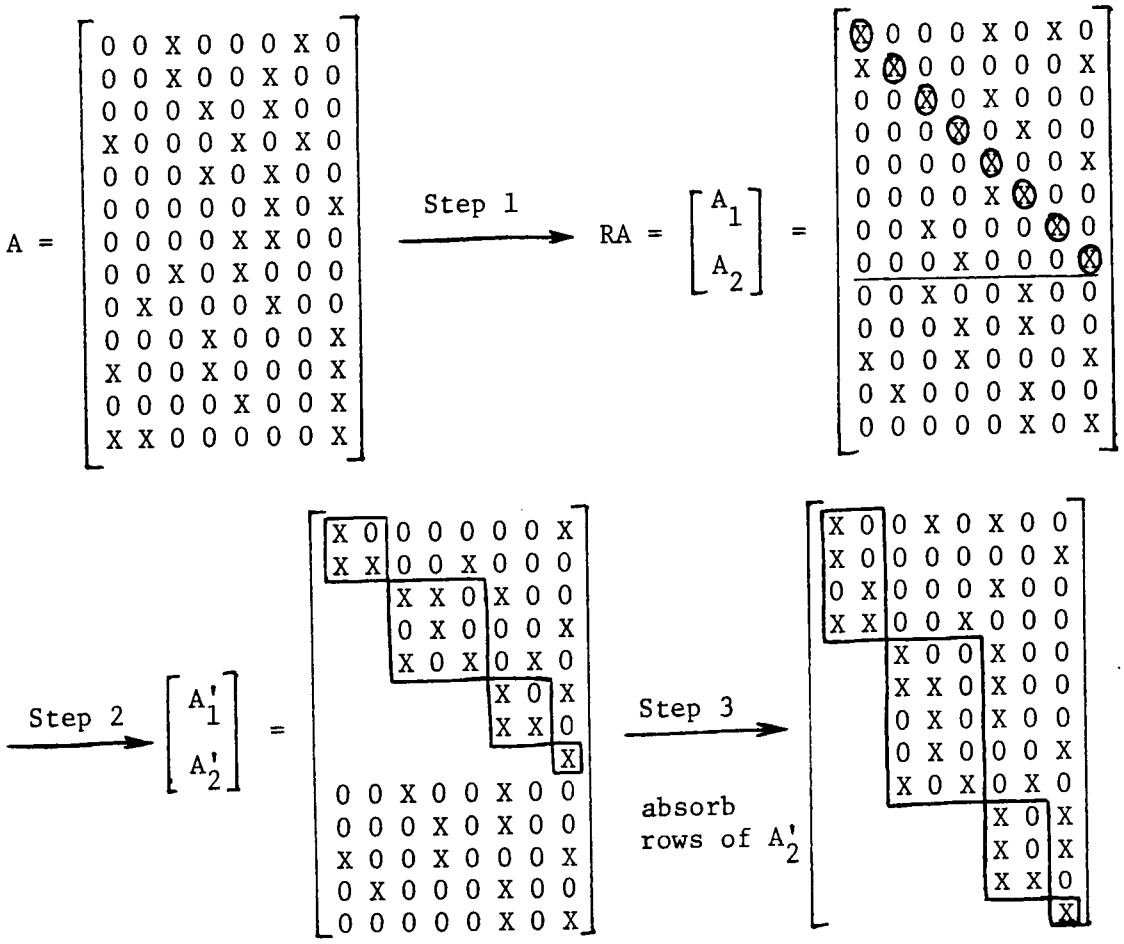


Figure 9. Example of block trapezoidal ordering scheme.

CHAPTER III

COMPUTER IMPLEMENTATION OF ORDERING OPTIONS

Software Used

The primary objective of this project was to compare performance of the minimum degree, nested dissection, and block trapezoidal ordering schemes in conjunction with Givens reduction to solve large sparse least squares problems. To this end, existing software from several sources was modified and combined to test the various ordering schemes on sample problems. Statistics for each method were recorded which provide a basis for comparing the effectiveness of the different schemes. In addition to the three ordering schemes listed above, the option of "no ordering" was tested in order to provide a benchmark against which to compare performance of the other ordering options. These four ordering options were compared on four sparse systems of equations, of sizes 75×50 , 100×75 , 150×100 , and 892×261 . The first three of these examples were constructed with known solutions in order to check the accuracy of the programs. The fourth problem consisted of actual geodetic network data obtained by the U. S. National Geodetic Survey as part of the 1983 North American Datum. All programs were written in IBM H Extended FORTRAN and were run on an IBM 370/3031 computer.

The basic software package used in this project was a double precision version of SPARSPAK, a sparse matrix package of subroutines written and documented by George and Liu [1978, 1981]. This package was modified by George and Heath [1980] to provide for solving a sparse least

squares problem using Givens rotations in conjunction with minimum degree ordering. Harwell subroutines MC21A and MC13D, developed by Duff [1978] and Duff and Reid [1978] to select a transversal of maximum length and to permute a square matrix to block upper triangular form, were adapted and used by Litsey [1980] in his implementation of block trapezoidal ordering for rectangular matrices. In this project, all of the above software was modified as necessary and combined to implement the four ordering options in conjunction with orthogonal factorization via Givens transformations to solve sparse least squares problems.

Data Structures for Sparse Matrices

A sparse matrix may be stored in one of several ways. Normally, only the nonzero elements of the matrix are specified; all others are assumed to be zero. Individual nonzero elements of a matrix can be entered as triples consisting of that entry's row index, column index, and value; this method is convenient if the number of nonzeros is not great. All of the nonzero elements in a given row of a sparse matrix can be specified, allowing the matrix to be read in row-by-row. SPARSPAK allows a single element, row, column, or submatrix to be entered at one time, providing several options for storing a sparse matrix. The 892×261 matrix of geodetic data used in this project was stored on a tape by rows according to a fixed format which allows up to five nonzero entries per row. In this instance, some input entries may be zero elements and must be separated out before the matrix is processed further.

Regardless of how a sparse matrix A is stored on input medium, it must be converted to either a Harwell or a SPARSPAK compatible storage scheme in order to be processed by the programs in this project. SPARSPAK routines, as adapted for Givens reduction, accept one row of a sparse matrix at a time, according to the format

(NSUBS,(SUBS(K),VALUES(K),K=1,NSUBS) ,

where NSUBS is the number of nonzero entries in the row, SUBS(K) is the column index of the kth nonzero entry in the row, and VALUES(K) is the corresponding value of that entry. This format allows a variable number of nonzeros for each row.

The Harwell-block trapezoidal ordering code requires a different scheme for storing a sparse matrix which is described below. The matrix is stored in four arrays: arrays ICN and VALUES contain the column indices and corresponding values of all nonzero elements of matrix A . Array element IP(I) points to the position in ICN where column indices for the Ith row begin, and array element LENR(I) gives the number of nonzero entries in the Ith row. Rows may be stored in any order, but all elements of a given row must be stored successively.

The following example illustrates the SPARSPAK and Harwell storage schemes for a sparse matrix. Let A be given by

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 6 & 0 & 5 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} .$$

Figure 10 shows how A would be stored according to the row-oriented SPARSPAK scheme or the Harwell scheme. Notice that, in the Harwell scheme, we have that row 4 of the matrix begins at the sixth entry in each of arrays ICN and VALUES, and row 4 has three nonzero elements.

SPARSPAK

2,(1,1.0,4,2.0)

2,(4,3.0,5,4.0)

1,(3,2.0)

3,(2,1.0,3,6.0,5,5.0)

1,(4,1.0)

Harwell

ICN	1	4	4	5	3	2	3	5	4
VALUES	1.0	2.0	3.0	4.0	2.0	1.0	6.0	5.0	1.0

IP	1	3	5	6	9
LENR	2	2	1	3	1

Figure 10. SPARSPAK and Harwell schemes for storing a sparse matrix.

In addition to providing the sparse matrix A for a problem $Ax = b$, one must input for each equation its right-hand side and a possible weighting factor for the equation. The weighting factor may be used in the context of least squares; for instance, in the case of geodetic adjustment, the weight may represent a confidence factor for a given observation.

General Organization of Computer Programs

The basic program layout in this project consists of two job steps, as indicated in Figure 11. In job step 1, data for a sparse system of equations is read according to its original format and converted to an appropriate format for the next stage (Step A in Figure 11).

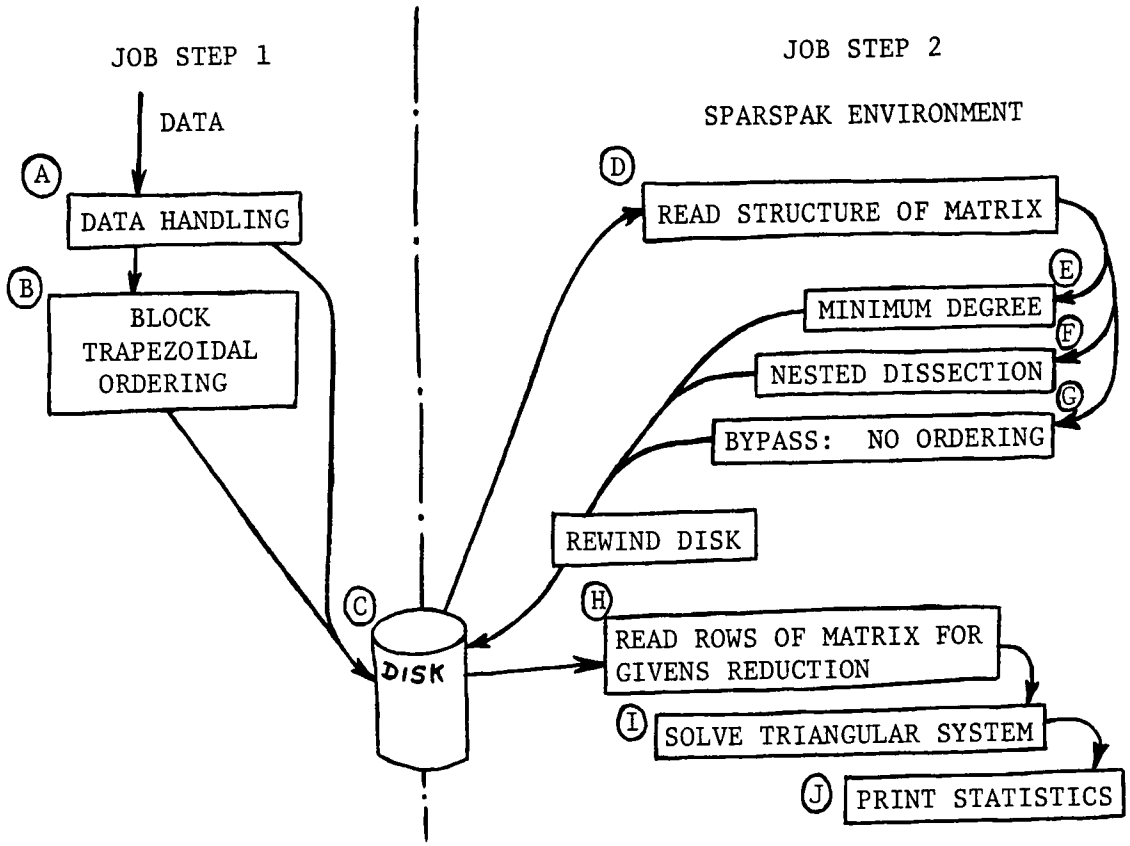


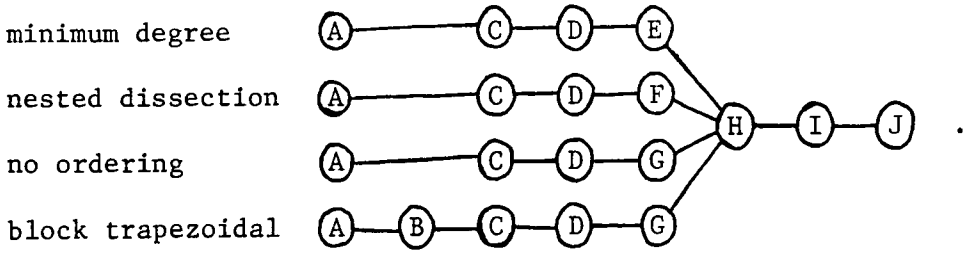
Figure 11. Schematic view of program.

For ordering options other than block trapezoidal ordering, the data is stored on disk according to SPARSPAK row-by-row format and passed to job step 2 (Step C). For the block trapezoidal ordering option, original data is converted to the Harwell storage scheme, ordering is performed (Step B), and the resulting reordered system is then converted to SPARSPAK format on the disk before being passed to step 2.

Job step 2 is executed in a SPARSPAK environment under control of a main driver routine. After the zero-nonzero structure of the coefficient matrix is read in (Step D), one of three ordering options is selected: the system is ordered by minimum degree ordering (Step E), nested dissection (Step F), or no ordering is performed (Step G). After

rewinding the data set on disk, numerical values for the system are read one row at a time, each row is processed by Givens transformations to form the Cholesky factor R (Step H), and the least squares solution to the system is computed (Step I). Finally, statistics provided by SPARSPAK are printed out (Step J).

The four sequences of execution steps listed below correspond to the four ordering options named:



In the next four sections, we shall discuss some of the programming considerations involved for each of the four execution sequences.

Implementation of Minimum Degree Ordering

As previously mentioned, the first job step in the execution sequence reads a sparse system of equations, converts it to SPARSPAK row format, and writes the system on disk as a temporary data set to be passed to the second job step. Each row (equation) consists of all non-zero elements on the left side of the equation, followed by the right-hand side of the equation and the weighting factor for that equation. An outline of the code for this job step appears below in Table 1. In this section of code, data for each row of the sparse system is assembled before being written as a record in a temporary data set on disk. This data set is named &&TEMP and is located on the disk associated with

TABLE 1

Outline of Code for First Job Step

```
//SPARSE JOB (__,__,__,__)
//STEP1 EXEC FORTQCLG
//FORT.SYSIN DD *
        REAL VALUES(10),RHS,WT
        INTEGER SUBS(10),NSUBS
        .
        .
        .
1       IF (end of data is reached) GO TO 2
        .
        .
        .  (read or generate data for next row
        .   of the system)
        .
        .
        WRITE (1) NSUBS,(SUBS(K),VALUES(K),K=1,NSUBS),RHS,WT
        GO TO 1
2       J=0
        WRITE (1) J,(SUBS(K),VALUES(K),K=1,J),RHS,WT
        STOP
        END
//GO.FT01F001 DD UNIT=SYSDA,DCB=(RECFM=VBS,LRECL=84,BLKSIZE=4204),
//   DSN=&&TEMP,DISP=(NEW,PASS),SPACE=(TRK,(10,1))
//GO.SYSIN DD DSN=NGSDAT,UNIT=TAPE,VOL=SER=012143,DISP=OLD
```

logical unit number 1, as shown in the first JCL DD statement. Data for each row is written as an unformatted record in order to save space. When the last actual row has been written, a dummy unformatted record is written to facilitate detection of the end of the data set. For a given row, the array SUBS holds column subscripts for nonzero coefficients on the left side, while array VALUES holds the numerical values of those coefficients. The variable NSUBS is the number of coefficients on the left side, RHS the right-hand side, and WT the weighting factor for the row. The last line in Table 1 shows the JCL card for reading the sparse

system from a tape whose name is "NGSDAT." If the system were to be read from cards, this DD statement would be: //GO.SYSIN DD * .

The second job step executes in a SPARSPAK environment under the control of a main driver program which calls SPARSPAK routines and controls data manipulation. An outline of the driver routine code appears in Table 2 below. Since a double precision version of SPARSPAK is used, the working storage array S is declared to be double precision; this array holds information which is passed to various SPARSPAK subroutines. In Table 2, code from the statement CALL IJBEGN to CALL IJEND causes the structure of the matrix A to be read in by rows (notice that only the column subscripts are recorded), from which the zero-nonzero structure of A^TA is determined. The variable M counts the number of rows that are read; rows are read from the data set passed from the first job step.

Subroutine ORDLB5 performs minimum degree ordering on A^TA so that $P(A^TA)P^T$ has sparse Cholesky factor R and then allocates a data structure for storing R which is based upon a symbolic factorization of $P(A^TA)P^T$ to determine the zero-nonzero structure of R . After rewinding the data set, the rows of the system are read again (Do Loop number 4) in order to compute the Cholesky factor R using Givens transformations. Notice that during this pass through the data, column subscripts and numerical values are read. Although the numerical values obtained as observations are stored in single precision on the data set, they are converted to double precision for the Givens computation. Subroutine LSBACK solves the resulting triangular system by back substitution. The solution is contained in the first NEQNS positions of the

TABLE 2

Main Driver Program

```
//STEP2 EXEC FORTQCLG
//FORT.SYSIN DD *
        COMMON /SPKUSR/ MSGLVL,IERR,MAXS,NEQNS
        DOUBLE PRECISION S(25000),A(10),B,W
        REAL VALUES(10),RHS,WT,DUMMY
        INTEGER SUBS(10),NSUBS
        MAXS = 25000
        CALL SPRSPK
        M = 0
        CALL IJBEGN
        DO 1 I = 1,MAXS
            READ (1) NSUBS,(SUBS(J),DUMMY,J=1,NSUBS)
            IF (NSUBS .EQ. 0) GO TO 2
            M = M+1
            CALL INCLQ (NSUBS,SUBS,S)
1       CONTINUE
2       CALL IJEND (S)
        CALL ORDLB5 (S)
        INPUT = 1
        REWIND INPUT
        DO 4 I = 1,M
            READ (1) NSUBS,(SUBS(J),VALUES(J),J=1,NSUBS),RHS,WT
            IF (NSUBS .EQ. 0) GO TO 5
            DO 3 J = 1,NSUBS
3           A(J) = VALUES(J)
            B = RHS
            W = WT
            CALL ROTAB (NSUBS,SUBS,A,B,W,S)
4       CONTINUE
5       CALL LSBACK (S)
        WRITE (6,6) (J,S(J),J = 1,NEQNS)
6       FORMAT (5(I4,1PD14.3))
        CALL PSTATS
        STOP
        END
//LKED.SYSLIB DD DSN=J1410.SPARSPAK,DISP=OLD
//GO.FT01F001 DD DSN=&&TEMP,DISP=(OLD,PASS)
```

array S and is printed out. Subroutine PSTATS prints statistics that are compiled by SPARSPAK.

Several of the subroutines called by the driver program (ORDLB5, ROTAB, and others) are not present in the SPARSPAK set of subroutines, but are modifications and additions to SPARSPAK made by George and Heath [1980]. These subroutines must be listed after the driver program and before the DD statements at the bottom of Table 2. Other SPARSPAK subroutines are called as necessary.

Implementation of Nested Dissection Ordering

The code in this case is essentially the same as that for minimum degree ordering. Once again, the first job step handles data entry and passes a data set to step 2 under control of the main driver program. (See Table 1, p. 45 and Table 2, p. 47.) In the driver program, one change is required: replace the statement CALL ORDLB5(S) by the statement CALL ORDRA5(S). Subroutine ORDRA5 orders the matrix A^TA by nested idssection; ORDRA5 is present in the original SPARSPAK set of subroutines.

Implementation of No Ordering

The central feature of this ordering option is a by-pass routine. Within SPARSPAK, there is a subroutine GENND (General Nested Dissection), called by ORDRA5, which determines a permutation of the nodes of the adjacency graph of A^TA to effect nested dissection ordering. In order to by-pass this ordering, GENND is replaced by a new GENND subroutine shown in Table 3 below. In the replacement subroutine, the identity

TABLE 3

By-Pass Subroutine for No Ordering

```
      SUBROUTINE GENND (NEQNS,XADJ,ADJNCY,MASK,
     1                  PERM,XLS,LS)
      INTEGER XADJ(1),ADJNCY(1),MASK(1),PERM(1)
     1        XLS(1),LS(1),ROOT,NUM,NSEP
      DO 500 I = 1,NEQNS
      PERM(I) = I
500   CONTINUE
      RETURN
      END
```

permutation is defined which results in no ordering being done. In order to implement the no ordering option, the main driver program still contains the statement CALL ORDRA5 (S), but the new GENND subroutine is inserted immediately behind the driver program.

Implementation of Block Trapezoidal Ordering

In order to implement this ordering option, the main driver program of job step 2 is set in the same way as for the no ordering option described in the previous section. That is, the driver program contains a call to ORDRA5, and the by-pass subroutine GENND is included. Data conversion and ordering are both handled in the first job step. In Table 4 below, the first job step in the case of block trapezoidal ordering is outlined with respect to data handling.

For example, in Table 4, suppose that the system $Ax = b$ to be ordered is such that A is 892×261 and is read in according to SPARSPAK row format. (Refer to p. 41 for definitions of the arrays ICN, IP, and LENR.) Do Loop number 4 reads in one row at a time. Do Loop

TABLE 4

Block Trapezoidal Ordering Code

```
//SPARSE JOB (__,__,__,__)
//STEP1 EXEC FORTQCLG
//FORT.SYSIN DD *
        INTEGER ICN(5000),IP(900),LENR(900),SUBS(10),NSUBS
        REAL VALUES(5000),RHS(900),WT(900),V(10),B,W
        K = 1
        M = 892
        N = 261
        DO 4 I = 1,M
        READ (5,1) NSUBS,B,W,(SUBS(K),V(K),K=1,NSUBS)
1       FORMAT (I2,2F7.2,10(I3,F7.2))
        DO 3 K2 = 1,NSUBS
        IF (K2 .NE. 1) GO TO 2
            IP(I) = K
            LENR(I) = NSUBS
            RHS(I) = B
            WT(I) = W
2       ICN(K) = SUBS(K2)
        VALUES(K) = V(K2)
        K = K+1
3       CONTINUE
4       CONTINUE
        LICN = K-1
        CALL  MC21A (M,N,ICN,LICN,IP,LENR,RHS,WT,...)
        CALL  MC13D (M,N,ICN,LICN,IP,LENR,RHS,WT,...)
        CALL PRMBLK (M,N,ICN,LICN,IP,LENR,RHS,WT,...)
        CALL ABSORB (M,N,ICN,LICN,IP,LENR,RHS,WT,...)
        DO 5 L = 1,M
        K1 = IP(L)
        K2 = K1 + LENR(L) - 1
        WRITE (1) LENR(L),(ICN(K),VALUES(K),K=K1,K2),RHS(L),WT(L)
5       CONTINUE
        J = 0
        WRITE (1) J,(SUBS(K),VALUES(K),K=1,J),RHS(1),WT(1)
        STOP
        END
```

number 3 inserts column subscripts and values into the ICN and VALUES arrays, defines a pointer IP(I) to that row, and records the number LENR(I) of nonzeros in that row, all according to the Harwell scheme for storing a sparse matrix. The variable LICN gives the total number of nonzeros in the matrix A . After the block trapezoidal ordering process is accomplished by four subroutine calls (See Litsey [1980] for details of this process.), Do Loop number 5 takes the reordered system and converts it from the Harwell storage scheme to SPARSPAK row format. Each row is written as an unformatted record on the data set that is passed to the second job step. After listing the subroutines MC21A, MC13D, PRMBLK, and ABSORB, DD statements as shown in the last three lines of Table 1, p. 45, are needed to end the job step shown in Table 4.

CHAPTER IV

RESULTS

Basis for Comparison of Ordering Methods

In this project, the relative performances of three methods for ordering a sparse least squares system $Ax = b$ were compared. This comparison is based upon statistics provided by the SPARSPAK package. The effectiveness of a solution method is gauged here by the amount of storage and execution time required to solve a problem. An ordering method seeks to take advantage of the sparsity of a system in order to reduce storage and execution time requirements. Statistics provided by SPARSPAK are discussed in the following paragraphs.

Execution times (in seconds) are reported for each of four steps: ordering (find permutation matrix P to obtain $P(A^TA)P^T$), allocation (set up data structure in which to store Cholesky factor R of $P(A^TA)P^T$), factorization (compute numerical values for R using Givens reductions), and triangular solution (solve $Rz = y$ and set $x = P^Tz$). In addition, operation counts are given for the factorization and solution steps. An operation is defined to be a multiplication or division since most arithmetic operations in these two steps occur as a multiply-add pair. The factorization step operation count reflects the amount of computation necessary to perform Givens reduction and is influenced by intermediate fill-in that occurs during that process. The operation count for the solution step gives the number of multiplications needed to solve $Rz = y$.

Storage requirements are reported for the ordering, allocation, and factorization-solution steps. In addition to storage required for nonzero floating-point values in R, the right-hand side, and the solution vector x, these figures include "overhead storage" for the adjacency graph G^{A^TA}, permutation P, structure of R, and other arrays needed to maintain data structures used by the solution method. Thus, these figures represent practical rather than minimal storage requirements. (See George and Liu [1981, pp. 281-283] for further discussion of storage requirements.)

Although one pays a price by introducing overhead storage needed to maintain specialized data structures which take advantage of sparsity, the hope is that storage required for ordering, allocation, and solving is less than that for storing all of A^TA in full symmetric mode. For example, a problem where A is 890×260 would result in A^TA of size 260×260; if sparsity is not exploited, one would expect storage to be about 34,000 floating-point locations (A^TA is symmetric). On the other hand, if a method is used which exploits sparsity in solving the system, a total storage requirement of about 9000 locations results, much of which involves integer data.

A major objective of ordering the system is to reduce fill-in so that computational time and storage are reduced. Thus, fill-in is a basic criterion for the effectiveness of a solution method. Recall that (final) fill-in occurs whenever a zero element in the upper triangle of $P(A^TA)P^T$ becomes a nonzero element in R. The amount of fill-in may be calculated as follows. The number k of off-diagonal nonzeros in $P(A^TA)P^T$ is reported by SPARSPAK. Thus, $k_1 = k/2$ is the number

of nonzeros in the upper triangle of $P(A^TA)P^T$. The number r of off-diagonal nonzero elements of R may be obtained by subtracting the number of diagonal elements of R from the number of operations required to solve $Rz = y$ since one multiplication is required for each nonzero of R during solution of $Rz = y$. Since each nonzero element of the upper triangle of $P(A^TA)P^T$ remains as a nonzero in R , the amount of fill-in is $r - k_1$.

Test Results

SPARSPAK statistics obtained on four test problems are given below in Tables 5, 6, and 7. Statistics are given for minimum degree, nested dissection, and block trapezoidal ordering and also for the no ordering option. Table 5 gives storage requirements. Note that in the block trapezoidal case, storage for ordering is not available from SPARSPAK. Storage for solving includes that for both the factorization and solution steps. Operation counts for the factorization and solution steps and computed fill-in are given in Table 6. Table 7 gives execution times for the ordering and allocation steps (combined) and the factorization and solution steps (combined). Ordering time for the block trapezoidal case is taken as the reported CPU time for the GO step in which ordering is performed and is not provided by SPARSPAK.

A series of examples given in Appendices 1 and 2 illustrates the result of ordering a 75×50 sparse matrix A according to the minimum degree, nested dissection, and block trapezoidal schemes. In Appendix 1, the zero-nonzero structure is given for A in its original form and for the matrices obtained after selection of a transversal,

TABLE 5

Storage Requirements for Four Test Problems

	75 × 50	100 × 75	150 × 100	892 × 261
No Ordering				
Storage for				
allocation	624	843	1361	5472
solution	1362	1756	3791	24,285
Minimum Degree				
Storage for				
ordering	774	1034	1342	6227
allocation	626	838	1177	4762
solution	980	1313	2329	7531
Nested Dissection				
Storage for				
ordering	401	536	697	3179
allocation	620	848	1240	4539
solution	1121	1476	2909	8442
Block Trapezoidal				
Storage for				
ordering	900*	1350*	1800*	12,000*
allocation	630	845	1240	4645
solution	1272	1711	2909	8143

*These figures are estimates and are not provided by SPARSPAK.

TABLE 6

Fill-in and Operation Counts for Four Test Problems

	75 × 50	100 × 75	150 × 100	892 × 261
No Ordering				
Fill-in	669	764	2231	18,292
Operations for				
factorization	100,314	106,186	387,445	37,211,056
solution	963	1185	2776	21,077
Minimum Degree				
Fill-in	305	326	953	2248
Operations for				
factorization	46,162	49,288	185,588	3,128,844
solution	579	747	1498	5033
Nested Dissection				
Fill-in	402	479	1470	3382
Operations for				
factorization	65,063	63,597	287,323	3,056,212
solution	726	900	2015	6167
Block Trapezoidal				
Fill-in	543	717	1470	2977
Operations for				
factorization	60,032	79,446	284,555	4,386,519
solution	867	1138	2015	5762

TABLE 7

Execution Times for Four Test Problems

	75 × 50	100 × 75	150 × 100	892 × 261
No Ordering				
Time for allocation	.326	.415	.532	1.301
Time for factorization and solution	1.084	1.223	3.274	246.728
Minimum Degree				
Time for ordering and allocation	.565	.669	1.112	6.593
Time for factorization and solution	.676	.778	1.930	27.187
Nested Dissection				
Time for ordering and allocation	.359	.491	.608	1.459
Time for factorization and solution	.824	.881	2.680	26.303
Block Trapezoidal				
Time for ordering and allocation	1.85	4.33	3.15	219.83
Time for factorization and solution	.776	.975	2.651	37.720

Note: times in seconds

conversion to block upper triangular form, and conversion to block upper trapezoidal form. Appendix 2 shows the zero-nonzero structure of $P(A^TA)P^T$ after minimum degree, nested dissection, and block trapezoidal ordering have been performed.

Discussion of Test Results

The primary purpose of this project was to investigate the relative performance of minimum degree, nested dissection, and block trapezoidal ordering of a sparse system $Ax = b$ before solution by Givens reduction. In particular, the performance of block trapezoidal ordering was of interest.

Storage requirements (see Table 5) were reduced by all three ordering methods in the factorization-solution step by an amount ranging from 3% to 69% less than the corresponding storage needed if no ordering occurred. In the 892×261 problem, storage for the factorization-solution step was reduced 69%, 65%, and 67% by the three ordering methods. In all problems, minimum degree required the least storage in the factorization-solution step. In terms of storage required for the ordering step, nested dissection required about 55% as much as minimum degree ordering. In all problems, storage for ordering was less (23% to 82%) than that for factorization-solution in the case of minimum degree and nested dissection. SPARSPAK statistics giving storage required for block trapezoidal ordering were not available since the ordering was not performed in a SPARSPAK environment. However, estimates based upon the arrays used in block trapezoidal ordering yield the figures marked by an asterisk in Table 5. Storage required for block

trapezoidal ordering runs from 16% to 93% more than that required for minimum degree ordering. In the 892 × 261 problem, storage for block trapezoidal ordering exceeds that for the subsequent factorization-solution step.

The number of operations required for factorization (see Table 6) was reduced 92%, 92%, and 88% by the three ordering methods in the 892 × 261 problem and was reduced 25% to 54% in the smaller problems, when compared to no ordering. Minimum degree ordering produced the lowest operation count during factorization on the three small problems, while nested dissection was lowest on the 892 × 261 problem. Block trapezoidal ordering produced a lower count than nested dissection on two problems. Operation count during factorization seems to depend upon the problem as well as the ordering method used.

The amount of fill-in (see Table 6) incurred in solving the 892 × 261 problem was reduced by 88%, 82%, and 84% by the three ordering methods when compared with no ordering. Minimum degree was best here and block trapezoidal second best. In the smaller problems, fill-in was reduced by 7% to 58% by the three ordering methods. Minimum degree ordering produced the least fill-in in all problems. Fill-in produced by minimum degree was 25% to 40% less than that produced by the other two ordering methods.

Time required for factorization and solution was reduced 89%, 89%, and 85% by the three ordering methods in the 892 × 261 problem when compared to no ordering. Nested dissection provided the lowest time on the 892 × 261 problem while minimum degree was lowest on the other

three problems. In the 892×261 problem, block trapezoidal ordering time was very high--high enough to offset the reduction in factorization-solution time.

Results given in this project did not utilize row sorting prior to Givens reduction. It is interesting to note results obtained by George and Heath [1980] in this regard. They tested the 892×261 problem using minimum degree ordering in conjunction with the standard normal equations approach and the orthogonal factorization approach using Givens reduction. They found that the Givens approach increased execution time by a factor of 5 and number of operations during factorization-solution by a factor of 4. However, when row-sorting was employed, the Givens process increased execution time and number of operations by factors of only 2.5 and 2 respectively when compared to the standard normal equations approach.

Conclusions

All three ordering methods resulted in great reductions in storage, time, and number of operations required in the factorization-solution step. Also, fill-in was greatly reduced. From a mathematical point of view, all three methods are highly effective in reducing (final) fill-in and intermediate fill-in associated with Givens reduction. The beneficial effects of these ordering methods which exploit sparsity are much more pronounced in the larger 892×261 problem than in the smaller problems. Minimum degree ordering generally produced the best results, but results produced by nested dissection and block trapezoidal ordering were comparable in terms of reducing fill-in and operation count, storage, and time required during factorization-solution.

The block trapezoidal ordering step incurred high execution times and increasingly high storage requirements for larger problems. This relatively poor performance may be due to the fact that some of the software for the block trapezoidal ordering algorithm has not been optimized with respect to taking advantage of sparsity considerations. This is particularly true of data structures used in that code. In contrast, the code for minimum degree and nested dissection ordering is highly refined with respect to sparsity. In order for block trapezoidal ordering to compete with minimum degree and nested dissection ordering on very large problems, its software implementation must be improved. This is true in spite of the fact that block trapezoidal ordering yields excellent results in the factorization-solution step.

LIST OF REFERENCES

LIST OF REFERENCES

1. Duff, I. S. [1978], "On algorithms for obtaining a maximum transversal," AERE Harwell Report CSS49, Harwell, England.

2. Duff, I. S. and Reid, J. K. [1978], "An implementation of Tarjan's algorithm for the block triangularization of a matrix," ACM Transactions on Mathematical Software, Vol. 4, pp. 137-147.

3. Gentleman, W. M. [1973], "Least squares computations by Givens transformations without square roots," Journal of the Institute of Mathematics and its Applications, Vol. 12, pp. 329-336.

4. George, A. and Heath, M. T. [1980], "Solution of sparse linear least squares problems using Givens rotations," Linear Algebra and Its Applications, Vol. 34, pp. 69-84.

5. George, A., Heath, M. T., and Plemmons, R. J. [1981], "Solution of large-scale sparse least squares problems using auxiliary storage," SIAM Journal of Scientific and Statistical Computing, to appear.

6. George, A. and Liu, J. W-H. [1978], "User's guide for SPARSPAK--Waterloo sparse linear equations package," Report CS-78-30, Department of Computer Science, University of Waterloo, Canada.

7. George, A. and Liu, J. W-H. [1981], Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, N.J.

8. Golub, G. H. and Plemmons, R. J. [1980], "Large scale geodetic least squares adjustments by dissection and orthogonal decomposition," Linear Algebra and Its Applications, Vol. 34, pp. 3-28.

9. Litsey, J. [1980], "Finding a block upper trapezoidal form of a rectangular matrix," M.S. Degree Thesis, Department of Mathematics, The University of Tennessee, Knoxville, Tennessee.

10. Stewart, G. W. [1973], Introduction to Matrix Computations, Academic Press, New York, N.Y.

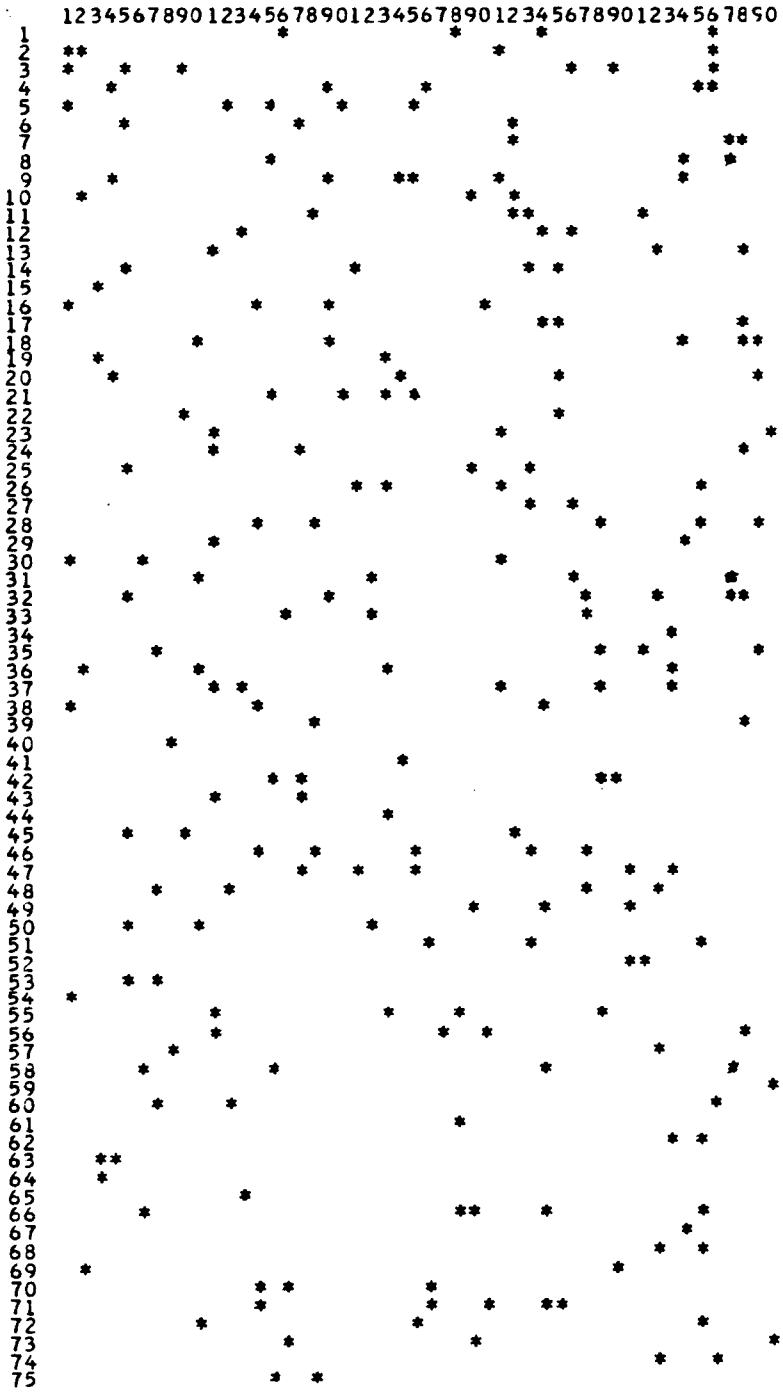
11. U.S. National Oceanic and Atmospheric Administration [1978], Proceedings of the Second International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Washington, D.C.

APPENDICES

APPENDIX 1

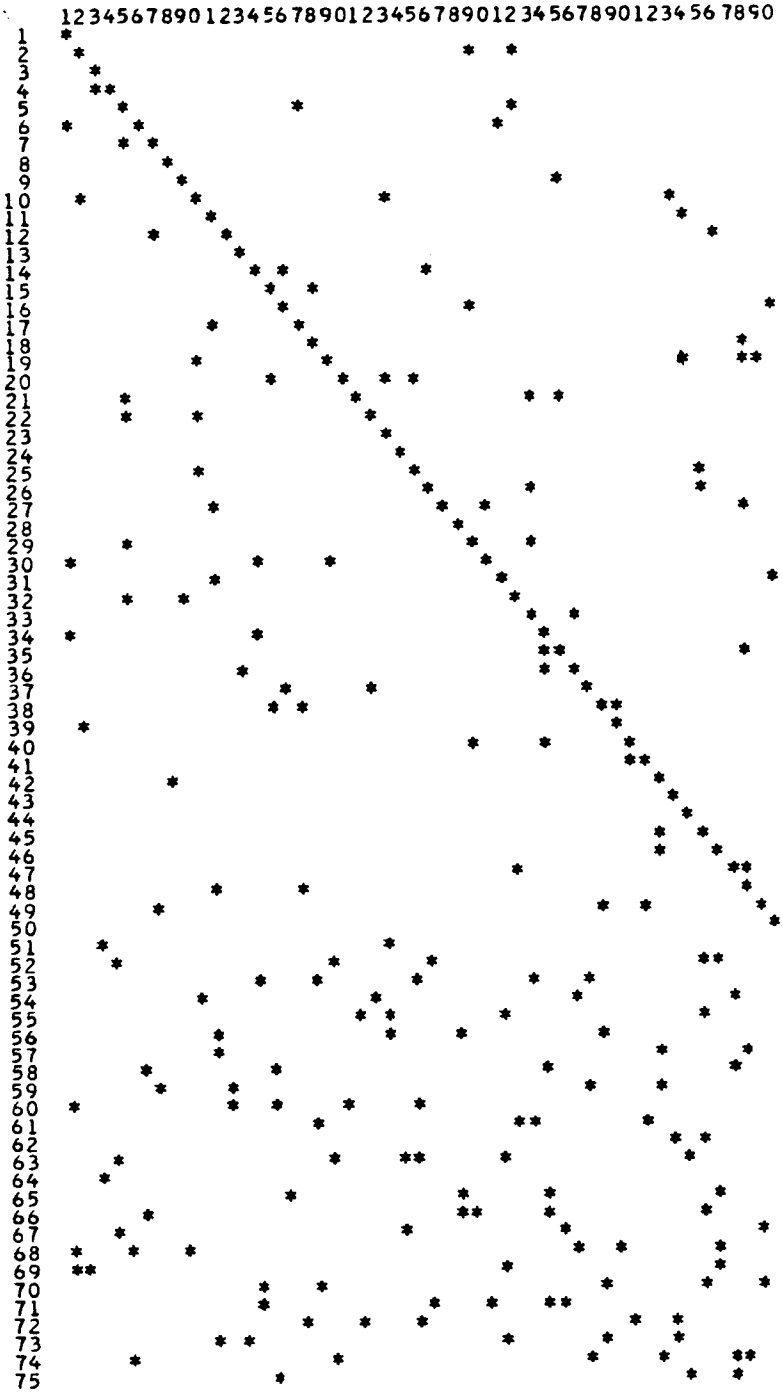
STAGES OF BLOCK TRAPEZOIDAL ORDERING

Original 75 × 50 sparse matrix A

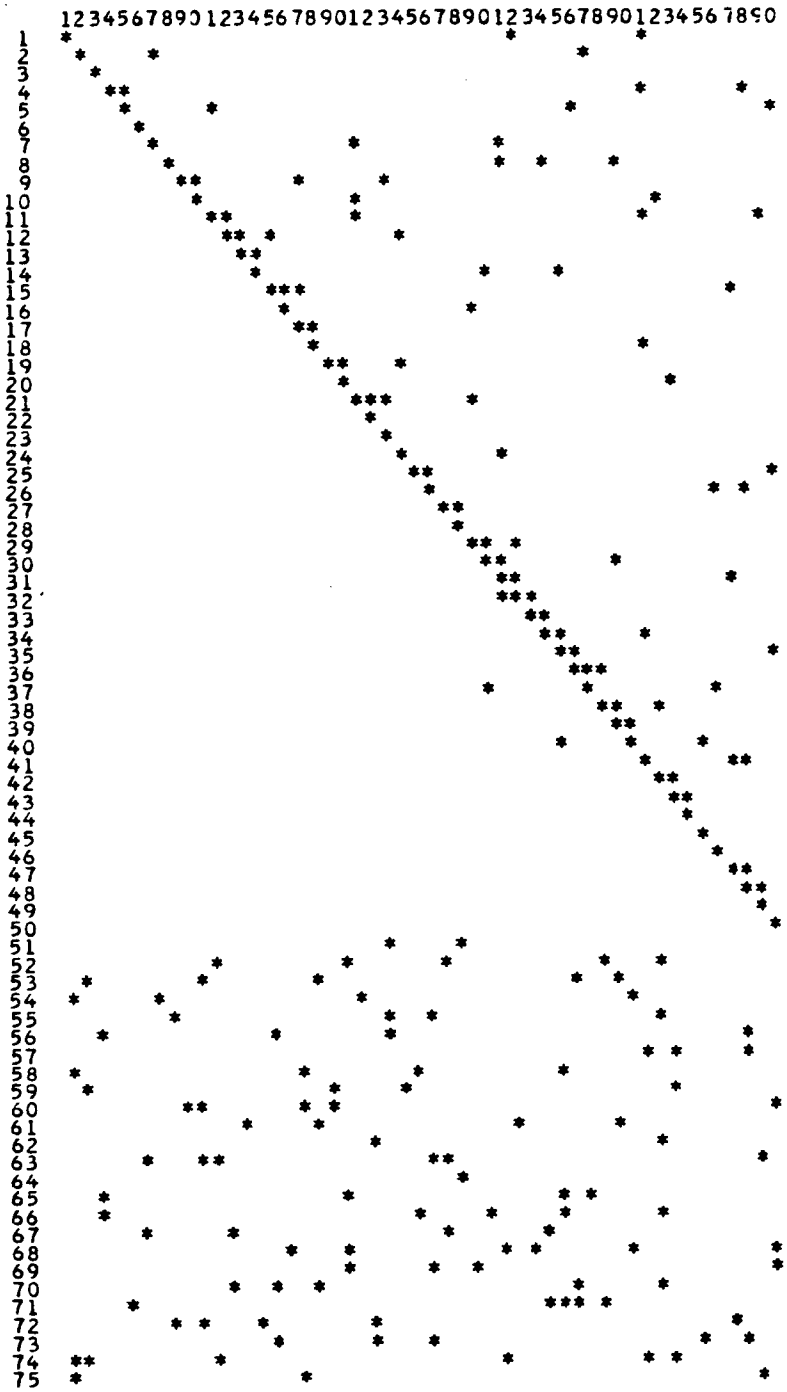


Note: An asterisk indicates a nonzero element of A .

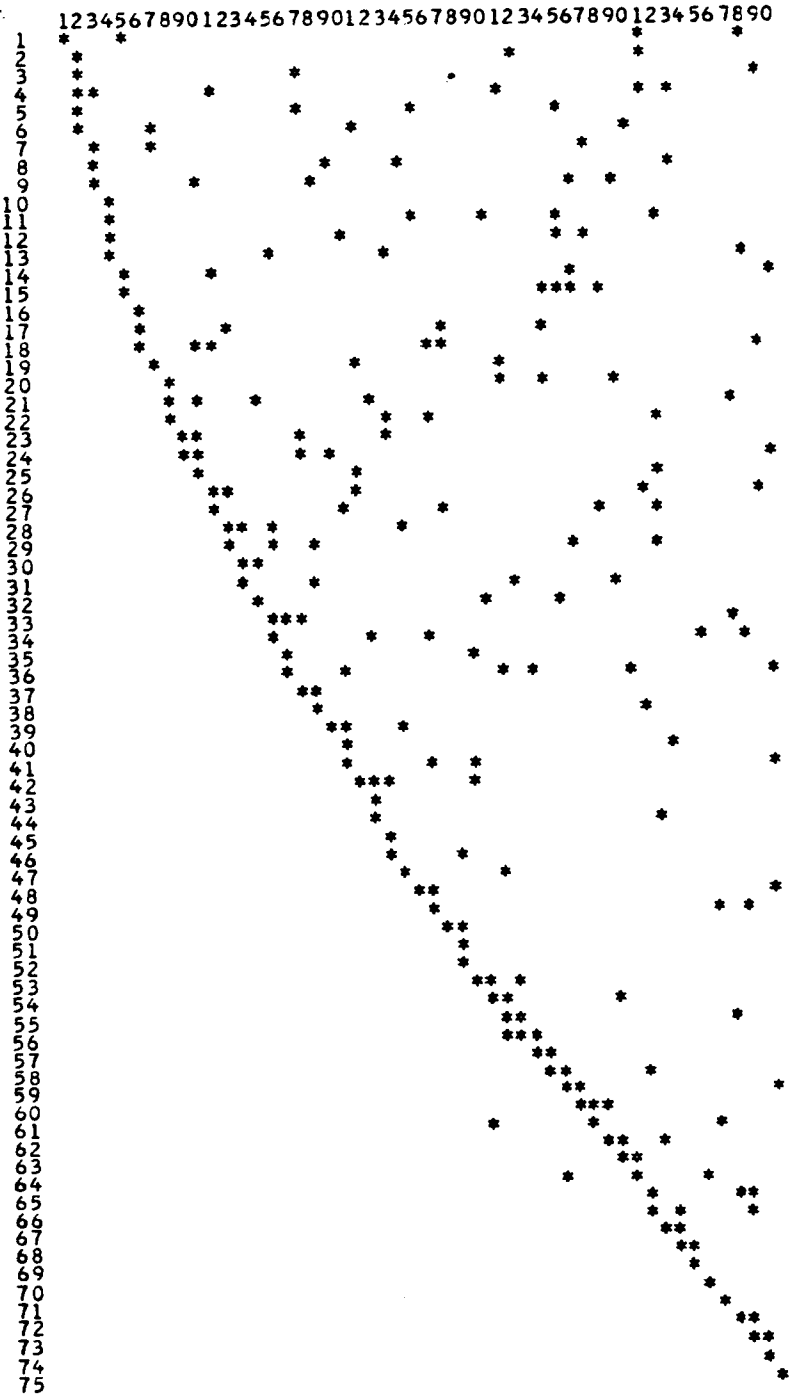
Matrix A after selection of transversal



Matrix A in block upper triangular form



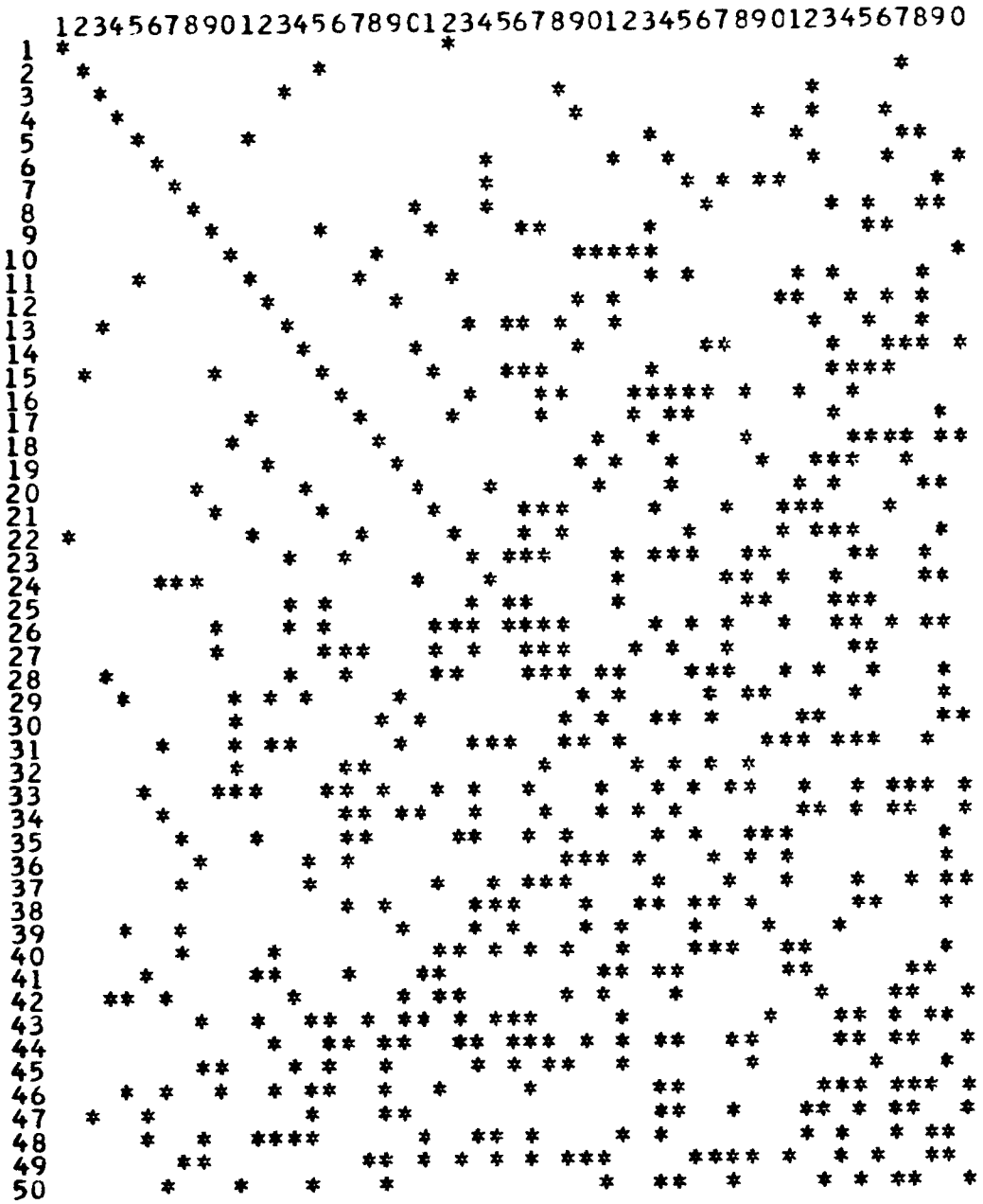
Matrix A in block upper trapezoidal form



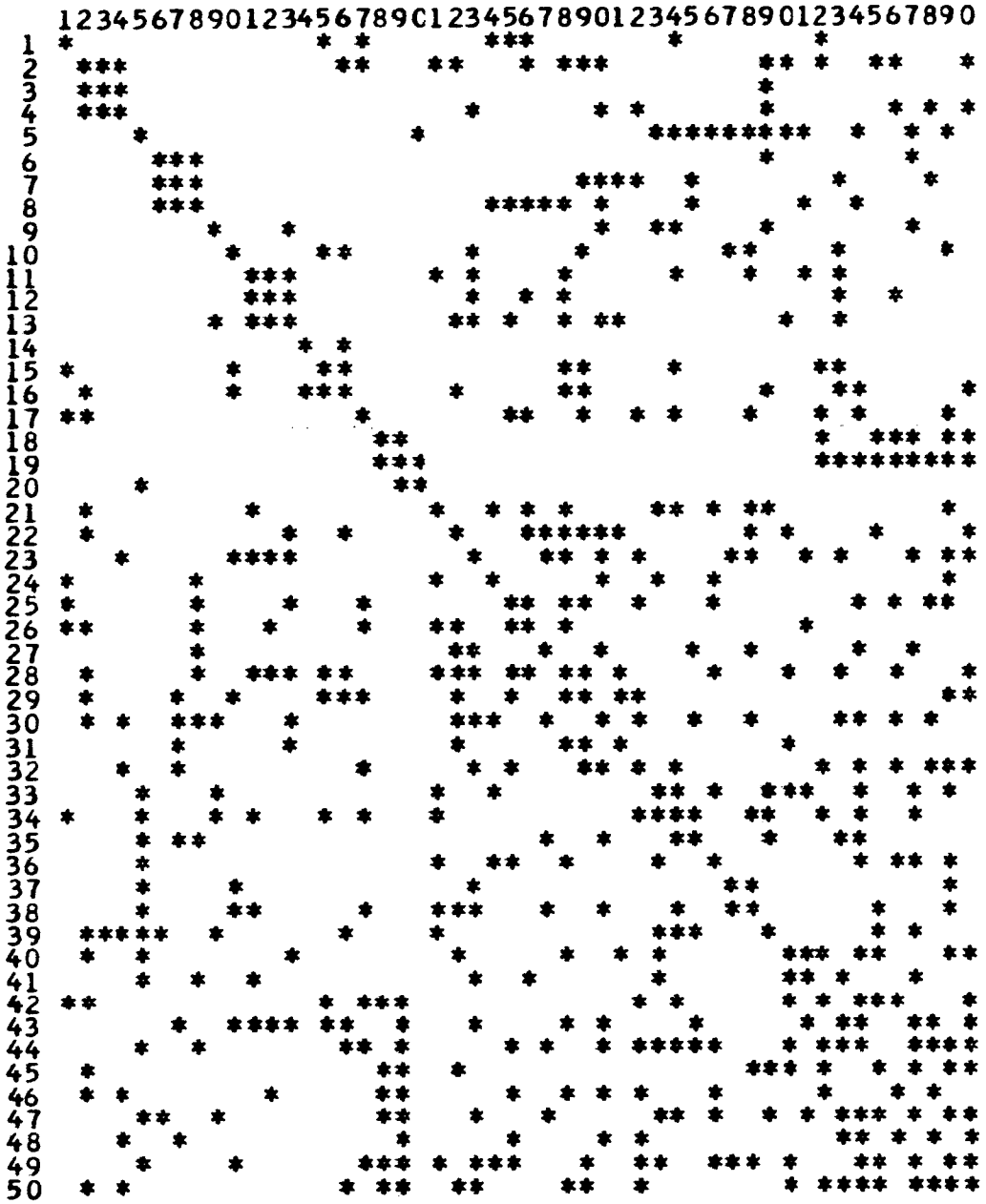
APPENDIX 2

STRUCTURE OF $P(A^TA)P^T$ AFTER THREE ORDERING METHODS

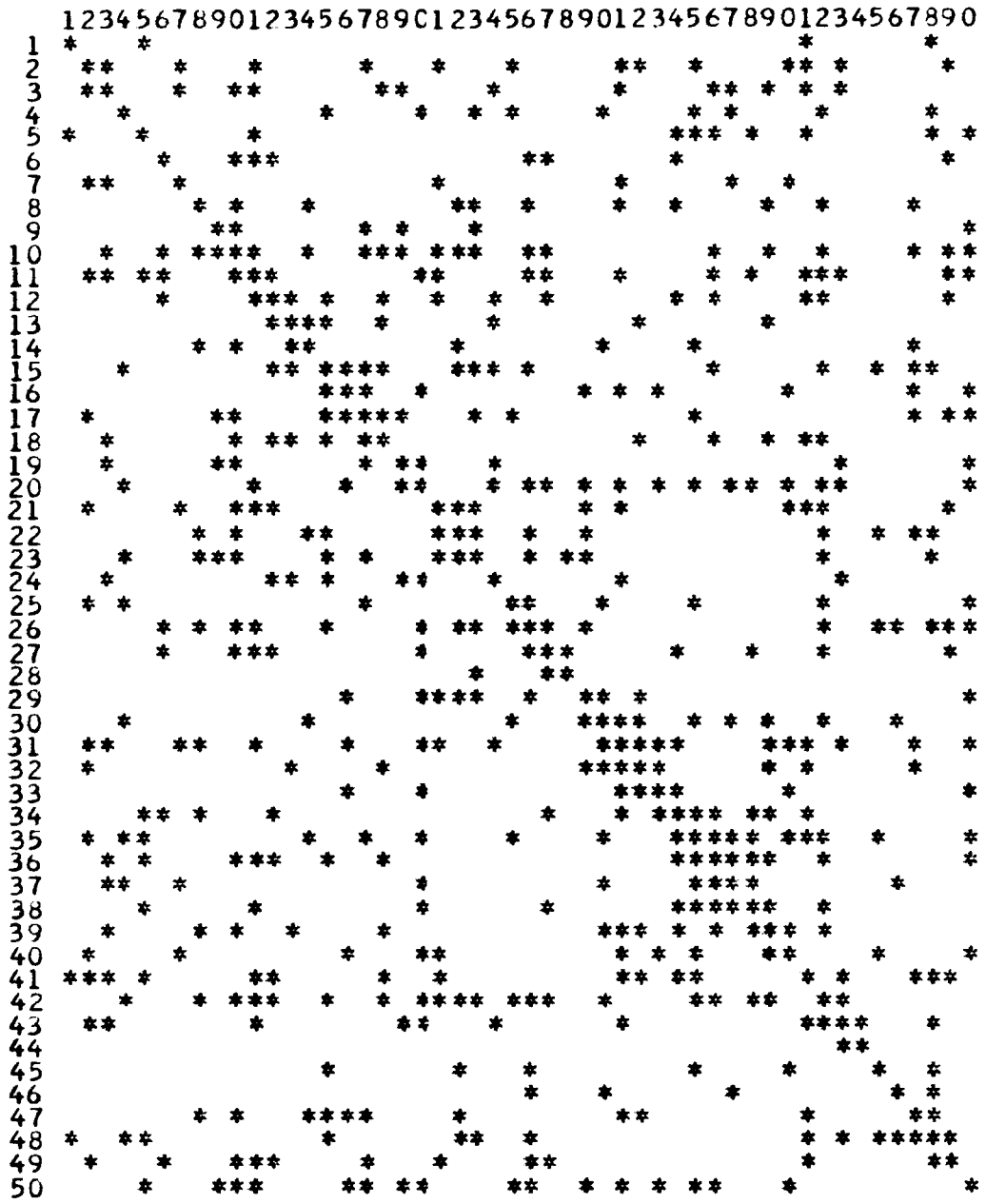
Zero-nonzero structure of $P(A^TA)P^T$ after minimum degree ordering



Zero-nonzero structure of $P(A^TA)P^T$ after nested dissection ordering



Zero-nonzero structure of $P(A^TA)P^T$ after block trapezoidal ordering



VITA

David Hume was born in Knoxville, Tennessee, on January 18, 1945. He attended public schools in that city and graduated from West High School in 1962. He entered The University of Tennessee that year and received a Bachelor of Science degree with a major in Mathematics in June, 1966. He attended the University of California at Berkeley from 1966 to 1968, taking courses in pure mathematics. From 1968 to 1973, he was employed as Instructor of Mathematics at Western Carolina University. In March, 1973, he received a Master of Arts degree with a major in Mathematics from The University of Tennessee. From 1973 to 1979, he served as graduate teaching assistant in the Department of Mathematics at The University of Tennessee and completed a Doctor of Education degree in Mathematics Education at that time. He was employed as Assistant Professor of Mathematics and Computer Science at Tennessee Technological University for the year 1979-1980. He completed a Master of Science degree with a major in Computer Science from The University of Tennessee during the 1980-1981 academic year. He will return to Tennessee Tech for the year 1981-1982.