Large sparse least squares computations

by

George Ostrouchov

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfilment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Approved:

In Charge of Major Work

For the Major Department

For the Graduate College

Iowa State University
Ames, Iowa

1984

TABLE OF CONTENTS

1. INTRODUCTION

Consider the standard linear model $y = X\beta + e$, where y is a vector of observed values of the dependent variable of length n, X is a fixed known $n \times p$ matrix, β is a length n vector of parameters to be estimated, and e is a length n vector of random errors. The most popular method of estimating the parameters is the method of least squares, which can be stated as

$$\min_b \|y - Xb\|_2,$$

where b is the estimator of β. This least squares problem is considered sparse, if the matrix X contains relatively few nonzeros. Typically this means less than 10% nonzeros. The problems considered here are large, because some overhead is incurred by taking advantage of sparsity , so that a reduction of storage and computation becomes evident only on large problems.

There are a number of application areas, where large sparse least squares problems arise. Perhaps some of the largest problems arise in geodetic network adjustment. One of the largest least squares problems attempted is the adjustment of the North American Datum, see Kolata (1978). This is a network of some 200,000 reference points on the North American continent, whose positions are adjusted by solving iteratively a least squares problem with approximately 6,000,000 observations and 400,000 parameters. Some other areas, where such problems arise, include photogrammetry, econometric models, analysis of seismological data, and finite element structural analysis. Many of these problems are so large, that storage needed for their solution by standard techniques exceeds the virtual address space of the largest computers.

Before describing how one can take advantage of sparsity, some direct methods for solving the standard least squares problem are briefly described. For a more complete description of these methods see Kennedy and Gentle (1980).

A solution to the least squares problem is given by a solution to the normal equations

$$X'Xb = X'y.$$

Two of the most popular methods, which solve the least squares problem through the normal equations, use the sweep operator or Cholesky decomposition on $X'X$. The generalized sweep operator produces a pseudo inverse $(X'X)^-$ and then a solution b is given by $(X'X)^-X'y$. Cholesky decomposition can be used when X has full rank, and it produces a lower triangular matrix L, such that $X'X=LL'$. Then, the unique solution is obtained by backsolving two triangular systems $Lc=X'y$ and $L'b=c$.

Although the above two methods are computationally very efficient, they may perform poorly on an ill-conditioned problem. Also, precision can be lost by forming $X'X$. Methods which deal directly with X avoid forming $X'X$, and are numerically more stable. These include the Peters and Wilkinson (1970) decomposition, and orthogonal decompositions. The former gives a decomposition of the form $X=LU$, where L is a unit lower trapezoidal matrix, and U is an upper trapezoidal matrix. This leads to equations $L'LUb=L'y$, which can be solved using methods of the preceding paragraph, but which are better conditioned than the normal equations. Assuming that the leading columns of X are linearly independent, the orthogonal decompositions are of the form

$$X=Q\begin{bmatrix} R & T \\ O & O \end{bmatrix}$$

where R is upper triangular of order $r=rank(X)$, and Q is orthogonal of order n. The matrix Q can be a product of Householder transformation matrices, or a product of Givens transformation matrices, or it can be produced by the Gram-Schmidt orthogonalization process. A solution b is then obtained by backsolving the triangular system $Ru=Q'y$, and setting $b=\begin{bmatrix} u \\ 0 \end{bmatrix}$.

Some of the above methods have been considered by various authors in the case when X is sparse. Of course storage methods, where only or nearly only the nonzeros are stored, are used. However, there is more that can be done in the above methods to take full advantage of the sparsity. Many methods are developed for specific application areas, where the X matrix is assumed to have a particular structure. There are also several general methods, which make little or no assumptions on the structure of X. A survey of direct methods for sparse linear systems, applicable here through the normal equations, is given by Duff (1983), and a survey of iterative methods for the same is given by Eisenstat (1983). Heath (1983) gives a comprehensive survey of methods particularly applicable to sparse least squares problems. He focuses mainly on developments since an earlier survey by Björck (1976). Three of the most widely applicable direct methods are briefly described below.

The first is the method of Cholesky factorization, which is discussed by George and Liu (1981) in their book on the solution of sparse positive definite linear systems. When Cholesky factorization is applied to $X'X$, the matrix L usually suffers fill-in. That is, some entries which are zero in the lower triangular part of $X'X$ become nonzero in L. When a symmetric row and column permutation is applied to $X'X$, the resulting Cholesky factor may have a different amount of fill-in. A symmetric row and column permutation amounts to reordering the normal equations, and relabelling the parameters. The amount of fill-in produced will have a direct effect on the amount of storage and computer time required to solve the least squares problem. Thus, in the sparsity context, the solution takes two steps. First, a "good" symmetric permutation with a sparse Cholesky factor is found, and then the permuted problem is solved. The two steps can be performed simultaneously, but it is advantageous to perform a symbolic step first to find a good permutation, and determine the nonzero structure of the matrix L, and then perform the numerical factorization in a fixed data structure. This is because otherwise the data structure has to be dynamic to accommodate the fill-in, and this can be very inefficient. Finding an optimal permutation is a computationally nearly impossible task for any but the smallest problems, where full matrix methods can be used anyway, so heurisrtic algorithms are used to find a "good" permutation. A number of these heuristic algorithms are described by George and Liu.

Björck and Duff (1980) discuss a method based on the Peters and Wilkinson LU decomposition of the X matrix. Pivot choice during the factorization is used to preserve the sparsity of L and U, as well as to enhance the conditioning of L. Björck and Duff modify the Peters-Wilkinson scheme, by observing that if the least squares problem is nearly consistent, an adequate solution can be obtained directly from the decomposition without solving $L'LUb=L'y$. If the problem is not nearly consistent, then only a correction is computed using $L'L$. This has the advantage that any ill-conditioning in L affects only the correction. Sparsity preservation is needed here at two stages. During the LU decomposition, which consists of Gaussian elimination, sparsity is preserved by choosing pivots according to the Markowitz (1957) scheme. Then during the "correction" phase, a positive definite system is solved, for which the methods described by George and Liu can be used.

Another method by George and Heath (1980) is based on the fact, that the upper triangular R factor from orthogonal decomposition of X is mathematically equivalent to L', the transpose of the factor from Cholesky decomposition of $X'X$. This means that sparsity preservation methods for Cholesky decomposition of positive definite matrices can be used in a symbolic phase, to produce a data structure for orthogonal decomposition of X. George and Heath use Givens rotations for the decomposition, since these allow X to be processed by rows. Their method thus requires no more storage than the normal equations method, since X can be read in by rows from auxiliary storage.

Obtaining information on the variance covariance structure of the model parameters is quite easy in the normal equations and the Givens algorithms, since this is given by $(R'R)^{-1}=R^{-1}(R^{-1})'$. This information is not so readily available from the Peters-Wilkinson algorithm. Of the above three methods, considering stability, flexibility, and efficiency, the George-Heath method using the Givens algorithm seems to show the most promise for solving general problems and obtaining statistical information about the estimates. For this reason, the improvement or extensions of the Givens algorithm was chosen as a topic of this research. A recent comparison of the above three methods by George, Heath, and Ng (1983) shows the normal equations method as the most efficient. However, it often fails on ill-conditioned problems. Of the two more stable methods, the Givens algorithm uses less storage, but which method executed faster was problem dependent.

The numerical phase of the George-Heath method operates directly on the X matrix without forming $X'X$. Its symbolic phase, however, forms $X'X$ rather than operate directly on X. There are some disadvantages in this, as will be discussed in Chapter 2. In Chapter 2, some results on row ordering of X are obtained, and then based on these results a symbolic phase is developed for Givens orthogonal decomposition, which operates directly on the nonzero structure of X.

When X contains some relatively dense rows, severe fill-in can result in the R factor. George and Heath (1980) propose to leave out these rows from the initial factorization, and then update only the solution, not the R factor, by these rows. The updating algorithm, also described in Heath (1982), assumes that X has full rank. With a very large problem, it may not be possible to make this assumption. Chapter 3 extends the updating algorithm to rank defficient problems.

An area, which has not received any attention from sparse matrix technology, is the computation in fitting a large analysis of variance model. The model matrix associated with a large model is quite sparse. Only the unbalanced case is of interest here, since very efficient algorithms exist for the balanced case. Chapter 4 discusses what can be done to improve efficiency in these computations with sparse matrix technology.

Finally, Chapter 5 contains computer testing and implementation of some of the methods developed in this research.

2. SYMBOLIC GIVENS FACTORIZATION OF A SPARSE MATRIX

Givens factorization of an $n \times p$ matrix X of rank r is of the form

$$X = Q\begin{bmatrix} R & T \\ O & O \end{bmatrix} \tag{2.1}$$

where R is upper triangular of order r, T is $r \times (p-r)$, and Q is a product of orthogonal Givens rotation matrices of order n. For simplicity of presentation, the first r columns of X are assumed linearly independent. Rank is a property of the numerical values of X, and the nonzero structure of X contains only partial information on the numerical values, namely whether they are zero or not. A symbolic factorization, thus, should obtain an R of order q, where $r \leqslant q \leqslant p$, and q is called the structural rank of X. The symbolic factorization is then of the form (2.1), where R is upper triangular of order q, and T is $q \times (p-q)$. When the $q \times q$ matrix R is computed numerically using exact arithmetic, it will contain $q-r$ zero rows, as was shown by Heath (1982) for the case when $q=p$.

George and Heath (1980) have observed that the factor R is mathematically equivalent to the Cholesky factor of $X'X$. They use this fact and symbolic Cholesky factorization of $X'X$ to obtain the nonzero structure of R. This approach assumes that $X'X$ is a sparse matrix, and always produces an R with $q=p$. The presence of a single full row in X makes $X'X$ a full matrix. Heath (1982) proposes to leave out relatively dense rows from the initial factorization, and then update only the solution with these rows. He also notes that there may be other, less obvious, rows which cause $X'X$ to be relatively full. These "problem" rows, when present, always cause fill in $X'X$, but there are cases when R is again a sparse matrix. For example, see Figure 2.1 (a) and (b). The example in (a), due to Björck (1976), shows that X and R can be sparse while $X'X$ is full. The example in (b) shows in addition that the structural rank of X can be less than the structural rank of $X'X$.

$$X = \begin{bmatrix} * & * & * & * \\ & * & & \\ & & * & \\ & & & * \\ & & & * \\ & & & * \end{bmatrix} \qquad X'X = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \qquad R = \begin{bmatrix} * & * & * & * \\ & * & & \\ & & * & \\ & & & * \end{bmatrix}$$

(a)

$$X = \begin{bmatrix} * & * & * & * \\ & * & & \\ & & * & \\ & & * & \\ & & * & \\ & & * & \end{bmatrix} \qquad X'X = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \qquad R = \begin{bmatrix} * & * & * \\ & * & \\ & & * \end{bmatrix} \quad T = \begin{bmatrix} * \\ \\ \end{bmatrix}$$

(b)

Figure 2.1 Both X and R are sparse in (a), but $X'X$ is full. Same holds in (b), and also structural rank of X and R is three, while structural rank of $X'X$ is four

Clearly, some sparsity information is lost by forming $X'X$. In particular, the information on the nonzero structure of individual rows is lost. This information can be retained by operating directly on the nonzero structure of X. This chapter discusses a symbolic Givens factorization algorithm, which operates on a bipartite graph representation of the nonzero structure of X. Section 2.1 presents basic notation of graph theory and its use in the study of sparse matrices. Most of the results of this section can be found in George and Liu (1981), and in Tewarson (1973).

Section 2.2 then presents a bipartite graph model of Givens reduction, and Sections 2.3 and 2.4 discuss row orderings and column orderings respectively. Finally in Section 2.5 an algorithm is presented, which implements symbolic Givens reduction, and which is based on the results of the preceding four sections.

2.1 The Use of Graph Theory in the Study of Sparse Matrices

The notation of graph theory is useful in the study of the nonzero structure of sparse matrices. Here, some basic notions are introduced, which are used throughout Chapter 2.

Definition 2.1.1 A graph $G = (C;E)$ consists of a finite set of nodes C together with a set E of edges, which are unordered pairs of nodes.

Definition 2.1.2 An ordering or labelling α of G is the mapping of $\{1, 2, \ldots, n\}$ onto C, where n is the number of nodes in C.

A graph $G = (C;E)$ labelled by α will be denoted by $G^\alpha = (C^\alpha;E)$.

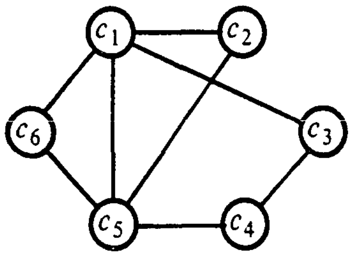
Definition 2.1.3 The labelled graph associated with a $p \times p$ symmetric matrix A, is denoted by $G^A = (C^A;E)$, and consists of p nodes labelled c_1 to c_p, and edges $(c_i,c_j) \in E$ iff $a_{ij} = a_{ji} = 0$, $i \neq j$.

See Figure 2.2 for an example of a sparse symmetric matrix and its associated labelled graph, where the ith diagonal element of the matrix is denoted by i, as it corresponds to node c_i of the graph, and off diagonal nonzeros are denoted by "*".

The unlabelled graphs of PAP', where P is a permutation matrix of order p, are the same, but the associated labellings are different. So, applying a symmetric row and column permutation to A is the same as relabelling the graph associated with A. Figure 2.3 gives an

$$\begin{bmatrix} 1 & * & * & & * & * \\ * & 2 & & & * & \\ * & & 3 & * & & \\ & & * & 4 & * & \\ * & * & & * & 5 & * \\ * & & & & * & 6 \end{bmatrix}$$

Matrix A



Graph G^A

Figure 2.2 A symmetric matrix and its associated labelled graph

example of PAP' and $G^{PAP'}$ for a permutation matrix P. Note that the structure of the graphs in Figures 2.2 and 2.3 is the same, only the labellings are different.

Definition 2.1.4 Nodes x and y in G are adjacent if $(x,y) \in E$.

Definition 2.1.5 The adjacent set of $Y \subset C$ in graph $G = (C;E)$ is

$$Adj(Y,G) = \{ \; x \in C-Y \mid (x,y) \in E \text{ for some } y \in Y \; \}.$$

$$\begin{bmatrix} 1 & & * & * & & \\ & 2 & * & * & & \\ * & * & 3 & * & & * \\ * & * & * & 4 & * & \\ & & & * & 5 & * \\ & & * & & * & 6 \end{bmatrix}$$

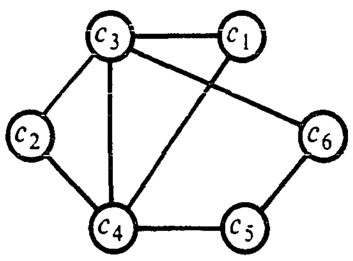


Figure 2.3 Graph of Figure 2.2 with a different labelling, and the corresponding permuted matrix PAP'

When Y contains a single node y, the adjacent set of Y is simply denoted by $Adj(y,G)$. For example, in Figure 2.3 $Adj(c_6,G^{PAP'}) = \{c_3,c_5\}$, and $Adj(\{c_2,c_4\},G^{PAP'}) = \{c_1,c_3,c_5\}$.

Definition 2.1.6 A path of length $\lambda \geqslant 1$ from node x to node y in graph G is an ordered set of $\lambda+1$ nodes, $(c_1,c_2,\ldots,c_{\lambda+1})$, such that $c_{i+1} \in Adj(c_i,G)$ for $i = 1, 2, \ldots, \lambda$, with $c_1 = x$ and $c_{\lambda+1} = y$.

For example, in Figure 2.3, a path of length 4 from c_3 to c_2 is (c_3,c_6,c_5,c_4,c_2).

Definition 2.1.7 A bipartite graph, or a bigraph, $B = (R,C;E)$ is a graph whose nodes are partitioned into two sets R and C, and each edge has one node from R and one node from C.

It should be clear from the context whether R refers to the matrix factor, or the set of nodes as above. Note that $Adj(R,B) = C$, and $Adj(C,B) = R$. The knowledge of either $Adj(r,B)\ \forall r \in R$, or $Adj(c,B)\ \forall c \in C$ completely defines the bigraph.

$$\begin{bmatrix} * & & * & & \\ & * & & & * \\ & & * & & * \\ * & & & & \\ & * & & * & \\ & * & & * & * \end{bmatrix}$$

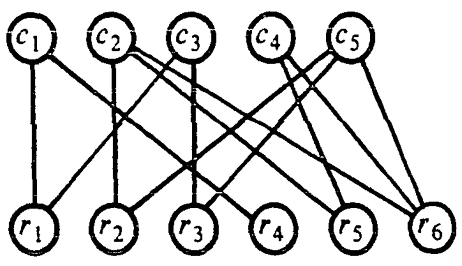


Figure 2.4 A matrix and its associated labelled bigraph

Definition 2.1.8 The ordered bipartite graph associated with an $n \times p$ matrix X is denoted by $B^X = (R^X, C^X; E)$. R^X consists of n nodes labelled r_1 to r_n corresponding to rows of X. C^X consists of p nodes labelled c_1 to c_p corresponding to columns of X. $(r_i, c_j) \in E$ iff $x_{ij} \neq 0$.

Whenever a graph represents a matrix, a labelling is implied. When the associated matrix X is clear from the context, B^X will simply be denoted by B. An example of a labelled bigraph is in Figure 2.4. The Adj operator can be used on a bigraph to obtain the set of rows or columns with a nonzero in a given column or row respectively. For example in Figure 2.4, $Adj(c_3, B) = \{r_1, r_3\}$ is the set of rows with a nonzero in column 3.

A row permutation of X is equivalent to a relabelling of nodes associated with rows in B^X, and a column permutation of X is equivalent to a relabelling of nodes associated with columns in B^X. Thus for all $n \times n$ permutation matrices P_r, and all $p \times p$ permutation matrices P_c, the unlabelled bigraphs of $P_r X P_c$ are identical, but the associated labellings change. Bipartite graphs thus provide a convenient tool for the study of row and column permutations of sparse matrices. Figure 2.5 gives an example of $P_r X P_c$ and the associated bigraph for permutation matrices P_r and P_c. Note that the structure of the bigraphs in Figure 2.4 and Figure 2.5 is the same, only the labellings have changed.

$$\begin{bmatrix} * & & & * & \\ * & & & & * \\ & * & * & & \\ & * & & * & \\ & * & * & * & \\ & & & & * \end{bmatrix}$$

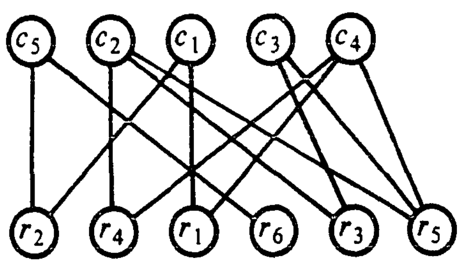


Figure 2.5 The bigraph of Figure 2.4 with a different labelling, and the corresponding matrix $P_r X P_c$

2.2 Givens Reduction of a Sparse Matrix and its Effect on the Associated Bipartite Graph

Each Givens transformation involves only two rows. If rows i and j of X are the two rows involved, and the first element of row j is to be annihilated, then the transformation takes the form

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \\ x_{j1} & x_{j2} & \dots & x_{jp} \end{bmatrix} = \begin{bmatrix} cx_{i1}+sx_{j1} & cx_{i2}+sx_{j2} & \dots & cx_{ip}+sx_{jp} \\ 0 & -sx_{i2}+cx_{j2} & \dots & -sx_{ip}+cx_{jp} \end{bmatrix}, \quad (2.2)$$

where
$$c = \frac{x_{i1}}{S},$$
$$s = \frac{x_{j1}}{S},$$
and
$$S = (x_{i1}^2+x_{j1}^2)^{1/2} .$$

In this transformation, row i is called the pivot row, and the element x_{i1} is called the pivot element.

Givens reduction of an $n \times p$ matrix X into the upper trapezoidal form (2.1) can be performed either by rows or by columns. In the following, the processing of an entire row or an entire column of X shall be referred to as a major step of the reduction, and the annihilation of a single nonzero will be referred to as a minor step of the reduction.

When processing by rows, the pivots used in each minor step are fixed, as is their order. That is, once a row is selected, its elimination sequence is determined, since any other sequence may result in filling previously annihilated positions. This sequence is illustrated in Figure 2.6(a). An advantage of processing by rows is that each row can be read from auxiliary storage, and only the partially formed factor R needs to be accessed during reduction.

Processing by columns allows much more flexibility within each major step. Each minor step can use any eligible row as a pivot, and rows can be processed in any order. During reduction, the completed rows of matrix R do not need to be accessed, however the entire unprocessed portion of X needs to be accessed. Processing by columns is equivalent to processing by rows, in terms of operations performed, when a particular order is taken within each major step. Each major step must use a single pivot row, and rows must be processed in the same order within each column. Figure 2.6(b) illustrates this order. Only nonzeros on main diagonal of R are shown by "*". When a subdiagonal element is zero, the corresponding minor step is omitted. Processing by columns is thus more flexible than processing by rows, and in fact is equivalent to processing by rows in a special case. For this reason the following will discuss only processing by columns.

$$
\underset{\text{(a)}}{\begin{bmatrix}
* & & & & \\
1 & * & & & \\
2 & 3 & * & & \\
4 & 5 & 6 & * & \\
7 & 8 & 9 & 10 & * \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20
\end{bmatrix}}
\qquad\qquad
\underset{\text{(b)}}{\begin{bmatrix}
* & & & & \\
1 & * & & & \\
2 & 7 & * & & \\
3 & 8 & 12 & * & \\
4 & 9 & 13 & 16 & * \\
5 & 10 & 14 & 17 & 19 \\
6 & 11 & 15 & 18 & 20
\end{bmatrix}}
$$

Figure 2.6 Elimination order in Givens reduction by rows (a), and an equivalent elimination order by columns (b)

Assuming no cancellation in (2.2), the nonzero structure of each of the two rows involved in a single Givens transformation becomes the union of their nonzero structures before the transformation, excluding the annihilated element in the pivot column. Figure 2.7 gives an example.

Let $B = (R,C;E)$ be the bigraph associated with a matrix X. Suppose a Givens transformation is applied to rows i and j of X, with x_{ik} as the pivot element and x_{jk} as the element to be annihilated. Both x_{ik} and x_{jk} must be nonzero. In terms of the

$$\begin{bmatrix} \circledast & * & & & * & * & \\ * & & & * & * & & * \end{bmatrix} \qquad \begin{bmatrix} \circledast & * & & * & * & * & * \\ & * & & * & * & * & * \end{bmatrix}$$

before transformation after transformation

Figure 2.7 An example of the nonzero structure of two rows before and after a Givens transformation. The pivot element is circled

bigraph B it is said that a Givens transformation is applied to nodes r_i and r_j of B, with (r_i,c_k) as the pivot edge and (r_j,c_k) as the edge to be annihilated. If B' is the bigraph after the Givens transformation, then the structure of B' is given by adjacency sets

$$\begin{aligned} Adj(r_\lambda,B') &= Adj(r_\lambda,B) \quad \forall\ r_\lambda \in R,\ \lambda \neq i,\ \lambda \neq j \\ Adj(r_i,B') &= Adj(r_i,B) \bigcup Adj(r_j,B) \\ Adj(r_j,B') &= Adj(r_i,B) \bigcup Adj(r_j,B) \ - \ c_k. \end{aligned} \tag{2.3}$$

These sets, of course, completely describe B'. Figure 2.8 gives the bigraph equivalent of Figure 2.7.

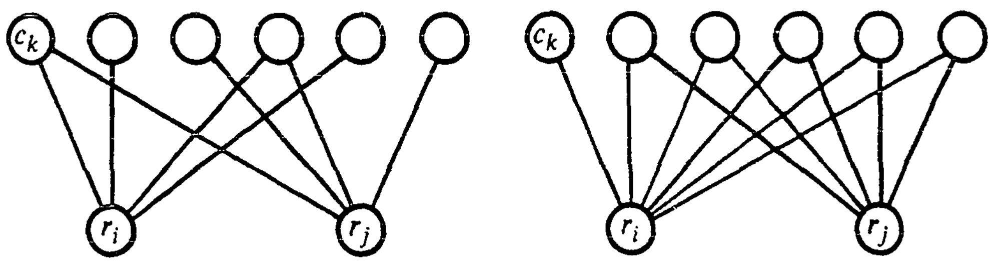


Figure 2.8 Bigraph representation of a Givens transformation of the two rows of Figure 2.7

Consider now a complete major step sequence of Givens transformations with a fixed pivot row. Let $B_j = (R,C;E_j)$ be the bigraph of only the unprocessed portion of X after the j^{th} minor step within a given major step. After each major step, the pivot row becomes part of the R factor, so the unprocessed portion consists of those rows, which were not used as pivots in previous major steps. Suppose column node c_α is being processed, and $Adj(c_\alpha,B_0) = \{r_{j_1}, r_{j_2}, \ldots, r_{j_k}\}$. Let r_{j_1} be the pivot row, and let $j_2, j_3, \ldots, j_k$ be the order in which rows are processed. After the first minor step, the structure of B_1 is, as in (2.3), given by

$$\begin{aligned} Adj(r_\lambda,B_1) &= Adj(r_\lambda,B_0) \quad \forall\ r_\lambda \in R,\ \lambda \neq j_1,\ \lambda \neq j_2 \\ Adj(r_{i_1},B_1) &= Adj(r_{i_1},B_0) \bigcup Adj(r_{i_2},B_0) \\ Adj(r_{i_2},B_1) &= Adj(r_{i_1},B_0) \bigcup Adj(r_{i_2},B_0) - c_\alpha. \end{aligned}$$

After the second minor step, the structure of B_2 is given by

$$\begin{aligned} Adj(r_\lambda,B_2) &= Adj(r_\lambda,B_1) \quad \forall\ r_\lambda \in R,\ \lambda \neq j_1,\ \lambda \neq j_3 \\ Adj(r_{i_1},B_2) &= Adj(r_{i_1},B_1) \bigcup Adj(r_{i_3},B_1) \\ Adj(r_{i_3},B_2) &= Adj(r_{i_1},B_1) \bigcup Adj(r_{i_3},B_1) - c_\alpha. \end{aligned}$$

So in terms of B_0

$$\begin{aligned} Adj(r_\lambda,B_2) &= Adj(r_\lambda,B_0) \quad \forall\ r_\lambda \in R,\ \lambda \neq j_1,\ \lambda \neq j_2,\ \lambda \neq j_3 \\ Adj(r_{i_1},B_2) &= Adj(r_{i_1},B_0) \bigcup Adj(r_{i_2},B_0) \bigcup Adj(r_{i_3},B_0) \\ Adj(r_{i_2},B_2) &= Adj(r_{i_1},B_0) \bigcup Adj(r_{i_2},B_0) - c_\alpha. \\ Adj(r_{i_3},B_2) &= Adj(r_{i_1},B_0) \bigcup Adj(r_{i_2},B_0) \bigcup Adj(r_{i_3},B_0) - c_\alpha. \end{aligned}$$

Finally, after completing the $k-1$ minor steps, thus completing the major step, the structure of B_{k-1} is given by Lemma 2.2.1.

Lemma 2.2.1

$$Adj(r_{i_\lambda},B_{k-1}) = \begin{cases} \bigcup_{s=1}^{k} Adj(r_{i_s},B_0) & \text{for } \lambda = 1 \\ \bigcup_{s=1}^{\lambda} Adj(r_{i_s},B_0) - \{c_\alpha\} & \text{for } \lambda = 2,3,\ldots,k \end{cases}$$

Proof: Preceding discussion. □

After completion of the major step, row r_{i_1} becomes the next row of the matrix R, and only the remaining rows stay for further processing. The preceding lemma thus gives the means of updating the bigraph for each major step of the reduction. As it stands, however, a dynamic data structure is needed to represent the bigraph. This is because the adjacency sets are growing, as we form new unions in each major step. Section 2.5, with the aid of results of this section and Section 2.3, develops a more efficient representation.

2.3 Row Ordering

Sparsity of the matrix R depends only on column ordering, and does not depend on the row ordering. However, the intermediate fill of the unreduced portion of X can vary substantially with both row order and column order, and thus affect the number of operations or Givens rotations needed to produce R. The comparison of two row orderings is meaningful, only if the same column order is used for both. The column order, therefore, is assumed fixed in this section.

When processing by rows, the row ordering is simply a linear ordering of the n rows. When processing by columns, however, the situation is much more complex. Each minor step is free to choose both rows from the set of rows with a nonzero in the current column at the current stage of the reduction.

Definition 2.3.1 When processing by columns, a row ordering α is a sequence of ordered pairs (s,t), where each ordered pair corresponds to a minor step of the reduction. Each ordered pair (s,t) specifies the two rows involved, where s is the pivot row.

There are two important restricted row orderings that need to be considered. The first restriction is when only a single pivot row is used within each major step of the reduction. An example of such row ordering is given in Figure 2.9.

Definition 2.3.2 A single pivot row ordering is a row ordering, where the pivot row entry of each ordered pair is constant within each major step.

$$\begin{bmatrix} 2 & 8 & * & & \\ 4 & 10 & 14 & 17 & 20 \\ * & & & & \\ 3 & 9 & 13 & * & \\ 6 & 11 & 15 & 18 & 19 \\ 1 & 7 & 12 & 16 & * \\ 5 & * & & & \end{bmatrix}$$

Figure 2.9 A matrix with a specified elimination order. Pivot elements are denoted by "*". The corresponding single pivot row ordering is {(3,6), (3,1), (3,4), (3,2), (3,7), (3,5), (7,6), (7,1), (7,4), (7,2), (7,5), (1,6), (1,4), (1,2), (1,5), (4,6), (4,2), (4,5), (6,5), (6,2), (5,2)}

Each major step of a single pivot row ordering induces a partial ordering on the rows of X.

Definition 2.3.3 Let $\{(s,t_1),(s,t_2),\ldots,(s,t_k)\}$ be a subsequence of a single pivot row ordering corresponding to a major step. The partial ordering induced by this subsequence on the set of n rows is $s,\ t_1,\ t_2,\ \ldots,\ t_k$.

The second restriction requires that the partial orderings of the n rows, induced within each major step of a single pivot row ordering, do not disagree. For example, the single pivot row ordering of Figure 2.9 is not of this type, since rows 2 and 5 are taken in different order in major steps 4 and 5.

Definition 2.3.4 A compatible row ordering is a single pivot row ordering, where the partial orderings induced within each major step on rows of the matrix are compatible.

A compatible row ordering corresponds to the elimination order of Figure 2.6(b). With this type of row ordering, a linear order of the n rows is produced, and so processing by columns is equivalent to processing by rows.

Each of the successive definitions puts more restrictions on the row ordering. Thus, the class of all row orderings contains the class of single pivot row orderings, which contains the class of compatible row orderings. This section contains results on the two latter classes of row orderings.

Suppose Givens reduction by columns with a single pivot row ordering is performed on an $n \times p$ matrix X. Let $B_j^i = (R^i, C^i; E_j^i)$ be the bigraph associated with the unreduced portion of X after the j^{th} minor step following the i^{th} major step. Thus B_0^0 is associated with the original matrix X. For notational convenience define Θ_α^i to be the ordered set of row nodes involved in major step i under single pivot row ordering α where c_i is the pivot column node. That is, $\Theta_\alpha^i = Adj(c_i, B_0^{i-1})$ under row ordering α. After the completion of i^{th} major step, which involved k_i rows, the structure of $B_{k_i-1}^{i-1}$ in terms of the structure of B_0^{i-1} is obtained by applying Lemma 2.2.1:

$$Adj(\{\Theta_\alpha^i\}_\lambda, B_{k_i-1}^{i-1}) = \begin{cases} \bigcup\limits_{s=1}^{k_i} Adj(\{\Theta_\alpha^i\}_s, B_0^{i-1}) & \text{for } \lambda = 1 \\ \bigcup\limits_{s=1}^{\lambda} Adj(\{\Theta_\alpha^i\}_s, B_0^{i-1}) & \text{for } \lambda = 2,3,\ldots,k_i. \end{cases} \tag{2.4}$$

The bigraph of the unreduced portion of X, $B_0^i = (R^i, C^i; E_0^i)$, is then obtained by removing the two nodes of the pivot edge of major step i. That is,

$$\begin{aligned} R^i &= R^{i-1} - \{r_i\} \\ C^i &= C^{i-1} - \{c_i\} \\ E_0^i &= E_{k_i-1}^{i-1} - \{ (r_i, c) \mid c \in C^i \} , \end{aligned} \tag{2.5}$$

where (r_i, c_i) is the pivot edge of major step a_i. A direct result of Lemma 2.2.1 is the following theorem.

Theorem 2.3.1 $Adj(\{\theta_\alpha^i\}_2, B_0^i) \subseteq Adj(\{\theta_\alpha^i\}_3, B_0^i) \subseteq \cdots \subseteq Adj(\{\theta_\alpha^i\}_{k_i}, B_0^i)$.

Proof: Lemma 2.2.1 gives

$$Adj(\{\theta_\alpha^i\}_\lambda, B_{k_i-1}^{i-1}) = \bigcup_{s=1}^{\lambda} Adj(\{\theta_\alpha^i\}_s, B_0^{i-1}) - \{c_i\} \qquad \text{for } \lambda = 2,3,\ldots,k_i \ .$$

But $Adj(\{\theta_\alpha^i\}_\lambda, B_0^i) = Adj(\{\theta_\alpha^i\}_\lambda, B_{k_i-1}^{i-1})$ for $\lambda = 2,3,\ldots,k_i$. □

Now consider two row nodes r_s and r_t involved in major step i, which satisfy $Adj(r_s, B_0^{i-1}) \subset Adj(r_t, B_0^{i-1})$, where the inclusion is proper. It is natural to process row node r_s before r_t to avoid possible unnecessary local fill-in in this major step.

Definition 2.3.5 A single pivot row ordering α is locally acceptable, if whenever $r_s, r_t \in \theta_\alpha^i$ and $Adj(r_s, B_0^{i-1}) \subset Adj(r_t, B_0^{i-1})$, with proper inclusion, r_s is ordered before r_t in θ_α^i.

The following lemma is useful in proving a theorem about locally acceptable single pivot row orderings. The lemma essentially states, that if a previously processed row is involved in a subsequent major step, all rows which followed it in the previous major step are also involved.

Lemma 2.3.1 Let m be the smallest $j>i$ such that $\Theta_\alpha^j \cap \Theta_\alpha^i \neq \varnothing$. Then,

$\{\Theta_\alpha^i\}_\lambda \notin \Theta_\alpha^m, \quad \lambda = 2, ..., u-1,$ and $\{\Theta_\alpha^i\}_\lambda \in \Theta_\alpha^m, \quad \lambda = u, ..., k_i$

for some $2 \leqslant u \leqslant k_i$.

Proof: $Adj(r,B_0^{m-1}) = Adj(r,B_0^i) \quad \forall \quad r \in \Theta_\alpha^i,$ since these row nodes were not altered between major steps i and m. Let u be the smallest λ such that $c_m \in Adj(\{\Theta_\alpha^i\}_\lambda,B_0^i)$, where c_m is the pivot column node of major step m. Then by Theorem 2.3.1 $c_m \in Adj(\{\Theta_\alpha^i\}_\lambda,B_0^i)$ for $\lambda = u, ..., k_i$, and by definition of u, $c_m \notin Adj(\{\Theta_\alpha^i\}_\lambda,B_0^i)$ for $\lambda = 2, ..., u-1$. □

Theorem 2.3.2 A locally acceptable single pivot row ordering is compatible up to the order of rows with identical nonzero structure.

Proof: Let α be a locally acceptable single pivot row ordering, and consider Θ_α^i and Θ_α^m of Lemma 2.3.1. The row nodes in Θ_α^i satisfy the relationship of Theorem 2.3.1 after completion of major step i. So, because α is locally acceptable, the order of row nodes common to major steps i and m must be the same in Θ_α^m as in Θ_α^i except possibly row nodes with identical adjacency structure. This holds for any major step i, so α must be compatible up to the order of rows with identical nonzero structure. □

If two rows have an identical nonzero structure, reversing their order will have no effect on the fill-in created during the reduction. Thus, any locally acceptable single pivot row ordering can be made completely compatible without changing the fill-in created. So a locally acceptable single pivot row ordering is essentially compatible. Compatibility is a good property, as it allows processing by rows during the numerical phase of the reduction. Locally acceptable single pivot row orderings are a subset of the class of all single pivot row orderings. What is lost by considering only locally acceptable single pivot row orderings? Theorem 2.3.3 will show that nothing is lost.

Let $Adj_\alpha(r, B_0^i)$ be the adjacency structure of row node r after major step i under row ordering α. Definition 2.3.6 gives a means of comparing some row orderings.

Definition 2.3.6 A row ordering β is at least as good as row ordering α, if $Adj_\beta(r, B_0^i) \subseteq Adj_\alpha(r, B_0^i) \quad \forall\ r \in R^i$, for $i = 1, 2, \ldots, p$.

The use of this definition is not in finding a good row ordering, because it cannot compare just any two row orderings. But it is sufficient to obtain a result about locally acceptable row orderings, without assuming a specific criterion, such as number of operations, or number of Givens transformations required for the complete reduction. The criterion of Definition 2.3.6 is more conservative than more specific criteria. A statement of "at least as good as" in terms of this definition implies "at least as good as" in terms of many reasonable specific criteria, such as the two named above.

Theorem 2.3.3 For every single pivot row ordering α there exists a locally acceptable single pivot row ordering β, which is at least as good.

Proof: Let γ and δ be two single pivot row orderings, which are identical up to major step i. That is, $\Theta_\gamma^h = \Theta_\delta^h$ for $h = 1, 2, \cdots, i-1$, so that $Adj_\gamma(r, B_0^h) = Adj_\delta(r, B_0^h) \ \forall\ r \in R^h$ for $h = 1, 2, \cdots, i-1$. Suppose $Adj_\gamma(r_a, B_0^{i-1}) \subset Adj_\gamma(r_b, B_0^{i-1})$, with proper inclusion, and same holds for ordering δ. Within major step i let γ take row node r_a before row node r_b, and let δ take them in the reverse order. That is, $\{\Theta_\gamma^i\}_h = \{\Theta_\delta^i\}_h$ for all h except $\{\Theta_\gamma^i\}_s = \{\Theta_\delta^i\}_t = r_a$ and $\{\Theta_\gamma^i\}_t = \{\Theta_\delta^i\}_s = r_b$, $s<t$. Then, using Lemma 2.2.1, for $h = \max(2,s), \cdots, t-1$

$$\begin{aligned}
Adj_\gamma(\{\Theta_\gamma^i\}_h, B_0^i) &= \bigcup_{m<h} Adj_\gamma(\{\Theta_\gamma^i\}_m, B_0^{i-1}) - \{c_i\} \\
&= \bigcup_{\substack{m<h \\ m\neq s}} Adj_\gamma(\{\Theta_\gamma^i\}_m, B_0^{i-1}) \cup Adj_\delta(r_a, B_0^{i-1}) - \{c_i\} \\
&\subseteq \bigcup_{\substack{m<h \\ m\neq s}} Adj_\gamma(\{\Theta_\gamma^i\}_m, B_0^{i-1}) \cup Adj_\delta(r_b, B_0^{i-1}) - \{c_i\} \\
&= \bigcup_{m<h} Adj_\delta(\{\Theta_\delta^i\}_m, B_0^{i-1}) - \{c_i\} \\
&= Adj_\delta(\{\Theta_\delta^i\}_h, B_0^i),
\end{aligned}$$

and for $h = 1, \cdots, \max(2,s)-1, t, \cdots, k_i$

$$Adj_\gamma(\{\Theta_\gamma^i\}_h, B_0^i) = Adj_\delta(\{\Theta_\delta^i\}_h, B_0^i).$$

So that $Adj_\gamma(r, B_0^i) \subseteq Adj_\delta(r, B_0^i) \quad \forall\ r \in R^i$. Suppose γ and δ are also identical after major step i, except for rows which are omitted in γ due to the switch in major step i. Since only unions are taken to form new adjacency sets of row nodes, $Adj_\gamma(r, B_0^h) \subseteq Adj_\delta(r, B_0^h) \quad \forall\ r \in R^h$ for $h = i+1, \cdots, p$. Thus, γ is at least as good as δ. Given any single pivot row ordering, pairwise row interchanges within major steps, such as the change from δ to γ, can produce a locally acceptable single pivot row ordering. Each interchange produces a row ordering, which is at least as good. So the final locally acceptable single pivot row ordering will be at least as good as the original row ordering. □

Corollary 2.3.1 For every single pivot row ordering α there exists a locally acceptable compatible row ordering β, which is at least as good.

Proof: By Theorem 2.3.2 a locally acceptable single pivot row ordering is compatible up to the order of rows with identical nonzero structure. But the order of these rows can be altered without affecting the nonzero structure. □

The set of locally acceptable compatible row orderings thus contains row orderings which are at least as good as any given single pivot row ordering. Single pivot row orderings in general do not allow processing by rows. So locally acceptable compatible row orderings are attractive, since they do allow processing by rows, and yet do not restrict opportunities for good orderings.

A locally acceptable compatible row ordering can be constructed during the symbolic factorization discussed in Section 2.2. In fact, local acceptability is defined in terms of the nonzero structure of a partially factored matrix. If the rows involved in each major step are ordered according to the local acceptability principle, the resulting row ordering will be locally acceptable. Theorem 2.3.2 assures compatibility of this ordering except for rows with identical nonzero structure. If the nonzero structure of two rows becomes identical in any major step, it will remain identical in subsequent major steps. By letting the first occurrence of these two rows determine their order in subsequent major steps, complete compatibility is ensured.

Duff (1974) tested three row ordering strategies. Two of the strategies satisfy the local acceptability criterion, when ties are handled properly, and their performance on the test matrices used was uniformly better than the third strategy. Duff used the number of Givens transformations as the criterion of comparison. The two strategies are given below. Strategy 2.3.1 is referred to as the minimum pivotal row fill strategy, and Strategy 2.3.2 is referred to as the local minimum fill strategy.

Strategy 2.3.1 Within each major step take the sparsest row as the pivot row, and then for each minor step process the row which causes least fill in the pivot row.

Strategy 2.3.2 Within each major step take the sparsest row as the pivot row, and then for each minor step process the row which causes the least fill in all rows remaining in the current major step.

Note that Strategy 2.3.2 does not count the fill created in the pivot row directly. Rather, it counts the fill distributed by the pivot row to remaining rows within the current major step. Duff calls this the corrected fill-in count.

Another strategy, which produces a locally acceptable row ordering, simply considers the number of nonzeros in each row locally within each major step. This will be called the minimum local row count strategy.

Strategy 2.3.3 Within each major step take the sparsest row as the pivot row, and then for each minor step process the row with the least number of nonzeros.

It is sometimes the case, that a relatively full row will cause severe fill-in in the matrix R. As was pointed out in the beginning of this chapter, it is also possible that some less obvious rows will cause this. Leaving out these rows from the initial factorization, and then updating the solution with these rows, may be advantageous. Chapter 3 deals with the question of updating. Since the bigraph contains the information on row structure during Givens reduction, it may be used to decide which rows should be left out. Particularly the amount of fill-in a row causes in a minor step can be used to make this decision. The last part of Section 2.5 addresses this topic again.

Only the two restricted classes of row orderings, as defined at the outset of this section, were discussed so far. Theoretically, only the class of row orderings allowing variable pivots possibly contains better row orderings, than the class of locally acceptable compatible row orderings. Duff (1974) has compared a variable pivot row ordering strategy with Strategy 2.3.1. On the test matrices considered, there was little to choose between the two strategies tested. There is, however, a rather special case, where the matrix structure clearly warrants using a variable pivot row ordering strategy. Matrices with this special structure are discussed in Chapter 4.

2.4 Column Ordering

In contrast to the definition of a row ordering, the definition of a column ordering is the same whether the matrix is processed by rows or by columns.

Definition 2.4.1 A column ordering is simply the linear order in which columns are processed.

The column order affects both the sparsity of the R factor, and the number of operations needed to obtain it. A number of the most popular strategies is discussed in George and Liu (1981). Some are also discussed by Duff (1974), and Duff and Reid (1976). Rose (1972) gives a good graph-theoretic study of the ordering problem for a positive definite matrix. The strategies can be divided into two classes. Strategies in one class use only the initial nonzero structure of X or $X'X$ to determine the column order. These include band and envelope methods, and dissection methods. Strategies in the other class make local decisions, during numerical or symbolic factorization, about which column to choose next. These include the minimum degree algorithm and other variations or generalizations of the Markowitz (1957) scheme. The strategies discussed here are in the latter class.

It is advantageous to perform the factorization symbolically, in order to obtain a data structure for the factor R. This speeds up the numerical factorization, as it can be done in a fixed rather than dynamic data structure. This chapter develops a symbolic Givens factorization algorithm for this purpose. George and Heath (1980) perform symbolic Cholesky factorization of $X'X$ to obtain the nonzero structure of R. As was pointed out at the beginning of this chapter, this is done because the Cholesky factor and the Givens factor are mathematically equivalent. However, they are equivalent only in the numerical phase. When only the positions of nonzeros are considered without the information on their values, as is done in symbolic factorization, in general the two are no longer equivalent. This is illustrated by the examples in Figure 2.1. The symbolic

Cholesky factor provides, in most cases very good, upper bound on the symbolic Givens factor in the sense of positions of nonzeros. It would be useful to know how and when exactly do the two symbolic factorizations differ. This discussion is included in this section, as it is only the column order that affects the nonzero structure during Cholesky factorization. First, the relationship between a bipartite graph associated with X and a graph associated with $A = X'X$ will be discussed.

We start by defining the *Bireach* operator, which gives the set of nodes in a graph reachable by a path of length two from a given node.

Definition 2.4.2 $Bireach(c,B) = \bigcup_{r \in Adj(c,B)} Adj(r,B) - \{c\}$, where $B = (R,C;E)$ is a bipartite graph.

This operator can then be used to construct a graph associated with $X'X$ from a bigraph associated with X, as can be seen in the following theorem.

Theorem 2.4.1 Let $B = (R,C;E)$ be a bipartite graph associated with X, and $G = (C;F)$ be a graph associated with $X'X$. Then $Adj(c,G) = Bireach(c,B)$ $\forall\ c \in C$.

Proof: Let $c_i \in Adj(c_j,G)$, so that $a_{ij} \neq 0$. But $a_{ij} = \sum_{m=1}^{n} x_{mi}x_{mj}$, so columns i and j must have at least one nonzero in a common row. This means that $Adj(c_i,B) \cap Adj(c_j,B) \neq \emptyset$, thus $c_i \in Bireach(c_j,B)$. With the assumption of no cancellation in $\sum_{m=1}^{n} x_{mi}x_{mj}$, the reverse of the above argument holds, so that $c_i \in Bireach(c_j,B)$ implies $c_i \in Adj(c_j,G)$. □

One step of Cholesky factorization of A using the outer product form, as described in George and Liu (1981), takes the form

$$A = A_0 = H_0 = \begin{bmatrix} d_1 & \nu_1' \\ \nu_1 & \bar{H}_1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{d_1} & 0 \\ \dfrac{\nu_1}{\sqrt{d_1}} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \begin{bmatrix} \sqrt{d_1} & \dfrac{\nu_1'}{\sqrt{d_1}} \\ 0 & I_{n-1} \end{bmatrix}$$

$$= L_1 A_1 L_1',$$

where $H_1 = \bar{H}_1 - \dfrac{\nu_1 \nu_1'}{d_1}$. This step is applied recursively to H_1, H_2, ..., H_{p-1}, and finally $R' = L_1 L_2 \cdot \cdot \cdot L_p$. One major step of Givens reduction by columns takes the form

$$X = X_0 = Q_1 \begin{bmatrix} f_1 & \rho_1 \\ 0 & X_1 \end{bmatrix},$$

where Q_1 is the product of orthogonal Givens rotation matrices, f_1 is a scalar, and other matrices conform. Then,

$$A = X'X = \begin{bmatrix} f_1^2 & f_1 \rho_1 \\ f_1 \rho_1' & \rho_1' \rho_1 + X_1' X_1 \end{bmatrix},$$

so that if X_i is the unreduced portion of X in Givens reduction by columns after major step i, then $H_i = X_i' X_i$. One step in Cholesky factorization is thus equivalent to one major step of Givens reduction by columns. Let $G^i = (C^i; F^i)$ be the graph of H_i, and c_i be the pivot column node of major step i. The following algorithm for producing G^i from G^{i-1} is adapted from Parter (1961).

Algorithm 2.4.1

1. Add edges to G^{i-1} so that $Adj(c_i, G^{i-1})$ are pairwise adjacent. That is,

   $Adj(c, G^i) = Adj(c, G^{i-1}) \cup Adj(c_i, G^{i-1}) \quad \forall \ c \in Adj(c_i, G^{i-1})$.

2. Delete node c_i and all edges incident to c_i.

As in Section 2.3, let $B_0^i = (R^i, C^i; E_0^i)$ be the bigraph associated with X_i. Algorithm 2.4.1 is used to update the unfactored portion of $X'X$ in symbolic Cholesky factorization, and (2.4) together with (2.5) is used to update the unreduced portion of X in symbolic Givens factorization. If G^i represents H_i and B_0^i represents X_i, then by Theorem 2.4.1 $Adj(c,G^i) = Bireach(c,B_0^i) \;\forall\; c \in C^i$. This is clearly the case with G^0 and B_0^0. Theorem 2.4.2 shows when $Adj(c,G^i) = Bireach(c,B_0^i) \;\forall\; c \in C^i$ holds for $i > 0$. When this relationship holds for a given i, the i^{th} symbolic Cholesky step is equivalent to the i^{th} symbolic Givens major step.

Theorem 2.4.2 Let $Adj(c,G^{i-1}) = Bireach(c,B^{i-1}) \;\forall\; c \in C^{i-1}$. Then, $Adj(c,G^i) = Bireach(c,B^i) \;\forall\; c \in C^i$, except when $Adj(c_i,B_0^{i-1}) = \{r_\alpha\}$ for some row node r_α, and there exist two other column nodes $c_j, c_m \in Adj(r_\alpha,B_0^{i-1})$ such that $Adj(c_j,B_0^{i-1}) \cap Adj(c_m,B_0^{i-1}) = \{r_\alpha\}$. That is, except when only a single row, r_α, has a nonzero in the pivot column, and at least one pair of other columns has a nonzero inner product only due to row r_α.

Proof: Let $\Theta^i = Adj(c_i,B_0^{i-1})$ be the set of k_i row nodes involved in major step i. By definition, for $c \in C^i$,

$$Bireach(c,B_0^i) = \bigcup_{r \in Adj(c,B_0^i)} Adj(r,B_0^i) - \{c\}$$

$$= \left[\bigcup_{r \in Adj(c,B_0^i) - \Theta^i} Adj(r,B_0^i) \right] \cup \left[\bigcup_{r \in Adj(c,B_0^i) \cap \Theta^i} Adj(r,B_0^i) \right] - \{c\}.$$

The row nodes in the first term are not altered in major step i, so B_0^i can be replaced by B_0^{i-1} giving

$$\left[\bigcup_{r \in Adj(c,B_0^{i-1}) - \theta^i} Adj(r,B_0^{i-1}) \right] \bigcup \left[\bigcup_{r \in Adj(c,B_0^i) \cap \theta^i} Adj(r,B_0^i) \right] - \{c\}. \tag{2.6}$$

If $c \notin Adj(c_i,G^{i-1}) = Bireach(c_i,B_0^{i-1}) = Adj(\theta^i,B_0^{i-1})$, then $Adj(c,B_0^i) \cap \theta^i = \varnothing$, and so the second term of (2.6) is null. Also $Adj(c,B_0^{i-1}) - \theta^i = Adj(c,B_0^{i-1})$, so that (2.6) equals $Bireach(c,B_0^{i-1}) = Adj(c,G^{i-1}) = Adj(c,G^i)$.

If $c \in Adj(c_i,G^{i-1})$, then two cases must be considered. First, suppose $k_i = 1$. Then $\theta^i = \{r_\alpha\}$ for some row node r_α, and by (2.5) $r_\alpha \notin R^i$, so $Adj(c,B_0^i) \cap \theta^i = \varnothing$. So (2.6) becomes

$$\bigcup_{r \in Adj(c,B_0^{i-1}) - \theta^i} Adj(r,B_0^{i-1}) - \{c\}. \tag{2.7}$$

Note that in this case $Adj(c_i,G^{i-1}) = Adj(r_\alpha,B_0^{i-1})$, so that $r_\alpha \in Adj(c,B_0^{i-1})$. This gives $Bireach(c,B_0^{i-1}) \supseteq Adj(r_\alpha,B_0^{i-1}) = Bireach(c_i,B_0^{i-1})$, so that $Adj(c,G^{i-1}) \supseteq Adj(c_i,G^{i-1})$, and $Adj(c,G^i) = Adj(c,G^{i-1})$.

Now if there exists $c_m \neq c_i$ such that $Adj(c_m,B_0^{i-1}) \cap Adj(c,B_0^{i-1}) = \{r_\alpha\}$, as specified in the "except" clause of the theorem, then $Adj(c_m,B_0^i) \cap Adj(c,B_0^i) = \varnothing$. This means that $c_m \in Bireach(c,B_0^{i-1})$, and $c_m \notin Bireach(c,B_0^i)$. So $Bireach(c,B_0^{i-1}) \neq Bireach(c,B_0^i)$, and $Adj(c,G^i) \neq Bireach(c,B_0^i)$ (in fact $Adj(c,G^i) \supset Bireach(c,B_0^i)$), thus giving the "except" clause of the theorem.

Otherwise, if such c_m does not exist, (2.7) equals

$$\begin{aligned} &\bigcup_{r \in Adj(c,B_0^{i-1})} Adj(r,B_0^{i-1}) - \{c\} \\ &= Bireach(c,B_0^{i-1}) \\ &= Adj(c,G^{i-1}) \\ &= Adj(c,G^i). \end{aligned}$$

Now suppose $k_i > 1$. By Lemma 2.2.1 $Adj(\Theta^i, B_0^{i-1}) = Adj(\{\Theta^i\}_{k_i}, B_0^i)$, and $\{\Theta^i\}_{k_i} \in Adj(c, B_0^i)$, so Θ^i can be omitted from the first term of (2.6). (2.6) now becomes

$$\begin{aligned} & Bireach(c,B_0^{i-1}) \bigcup Adj(\{\Theta^i\}_{k_i},B_0^i) \\ &= Bireach(c,B_0^{i-1}) \bigcup Adj(\Theta^i,B_0^{i-1}) \\ &= Bireach(c,B_0^{i-1}) \bigcup Bireach(c_i,B_0^{i-1}) \\ &= Adj(c,G^{i-1}) \bigcup Adj(c_i,G^{i-1}) \\ &= Adj(c,G^i). \qquad \square \end{aligned}$$

This is a rather tedious proof. The general idea is, that if more than one row is involved in a major step, the nonzero patterns of these rows are copied into the last row involved, which stays in the bigraph. If only a single row is involved, it leaves the bigraph, and if its effect on the *Bireach* pattern is not duplicated in other rows, the pattern changes. The graph representation does not see that only one row was involved, and fails to record any change in the adjacency pattern.

In terms of numerical Cholesky factorization, the above situation amounts to a special case of numerical cancellation. To illustrate this, consider the outer product form of Cholesky factorization, as described earlier in this section. The cancellation occurs when forming

$H_1 = \bar{H}_1 - \frac{\nu_1 \nu_1'}{d_1}$. If h_{ij}^k is the ij^{th} entry of H_k, then

$$h_{ij}^1 = h_{ij}^0 - \frac{h_{i1}^0 h_{1j}^0}{h_{11}^0} .$$

But

$$h_{ij}^0 = \sum_{m=1}^{n} x_{mi} x_{mj},$$

so that

$$h_{ij}^1 = \sum_{m=1}^{n} x_{mi}x_{mj} - \frac{\sum_{m=1}^{n} x_{mi}x_{m1} \sum_{m=1}^{n} x_{m1}x_{mj}}{\sum_{m=1}^{n} x_{mj}^2} .$$

Now if column 1 of X has a single nonzero in row α, and nonzeros of columns d and e of X coincide only in row α, then the de entry of H_1 reduces to

$$h_{de}^1 = x_{\alpha d}x_{\alpha e} - \frac{x_{\alpha d}x_{\alpha 1}x_{\alpha 1}x_{\alpha e}}{x_{\alpha 1}^2} = 0 .$$

After an occurrence of such cancellation, the two types of symbolic factorization no longer agree, since the symbolic Cholesky factorization does not detect the cancellation.

The most popular algorithm, based on local decisions during Cholesky factorization, is the minimum degree algorithm due to Tinney (1969), which is a symmetric matrix variant of the Markowitz (1957) scheme.

Definition 2.4.3 The degree of node c in graph $G = (C;F)$ is the number of edges incident to c. That is, $Deg(c,G) = |Adj(c,G)|$.

At each step of symbolic Cholesky factorization, the node with minimum degree is processed next. This minimizes locally the number of nonzeros in the next row added to the matrix factor R. In fact, the minimum degree is the number of nonzeros in the row added to R. Using Theorem 2.4.1, this algorithm can be adapted to symbolic Givens reduction. The degree of each column node c after major step i is given by $|Bireach(c,B_0^i)|$. The adapted algorithm operating on the bigraph B associated with X should perform better in some cases than the algorithm for graph G associated with $X'X$, since it accounts for the numerical cancellation discussed above.

In a large sparse matrix it is usually the case, that many columns have the same degree. When operating on the graph G, there is no information which can be meaningfully used to break the minimum degree ties. The bigraph B, however, has information on row nonzero structure, which can be used for meaningful tiebreaking in the adapted minimum degree algorithm. An example of a simple tiebreaking strategy is to take the tied minimum degree column which has the least number of nonzeros. This will have the effect of processing columns which involve fewer rows early. The number of operations needed to produce R should decrease, but the number of nonzeros in R will probably not be greatly affected. Also note that when a column with a single nonzero is processed, no fill is produced, since the row simply becomes part of R. Without such tiebreaking, columns with a single nonzero may be lost by being filled-in. Both examples of Figure 2.1 illustrate this. All columns are tied with degree three, but processing first any other column than column one will produce fill. Column one is ordered first with the simple tiebreaking strategy.

Other column ordering strategies for Givens reduction include those given by Duff (1974), where he uses them directly during the numerical phase of the reduction without performing a symbolic phase. Some of these are: taking the column with minimum nonzero count; taking the column with minimum nonzero count in the row with minimum nonzero count; taking the column which contains the minimum product of row and column counts (Markowitz (1957)); and taking the column which contains the minimum product of the row count and the square of the column count. Any of these strategies can be used in the symbolic Givens reduction described in this chapter to generate a data structure for the numerical phase.

2.5 Implementation of Symbolic Givens Reduction

The successive bigraphs of the unreduced portion of the matrix after each major step can be generated using (2.4) and (2.5). Since we are taking unions, the number of edges in the bigraph can grow. As a result of Corollary 2.3.1, we are only interested in locally acceptable compatible row orderings, when the numerical factorization is to be done by rows. Using Theorem 2.3.1, an efficient representation can be developed for the successive bigraphs, when a compatible row ordering is used. First, a few more definitions are needed. Most of the definitions can be found in George and Liu (1981).

Definition 2.5.1 A connected graph is a graph in which there exists a path between all pairs of nodes.

Definition 2.5.2 A tree $T = (X;E)$ is a connected graph, where
$|X| = |E| + 1$.

It is easily shown, that every pair of nodes in a tree is connected by exactly one path.

Definition 2.5.3 A rooted tree is an ordered pair (r,T), where r is a distinguished node of T called the root.

The path from r to a node $x \in X$ is unique. If the path passes through $y \in X$, then y is an *ancestor* of x, and x is a *descendant* of y. If in addition $(x,y) \in E$, then y is the *parent* of x, and x is a *child* of y. Another way to characterize a rooted tree is that every node has a single parent except the root, which has no parent. A node y together with its descendants and associated edges is a *subtree* of T, and y is the root of this subtree. A rooted tree can be used to impose a partial ordering on its nodes.

Definition 2.5.4 If node x is a descendant of node y, then x is ordered before y.

The ordering works, because only a single path exists between every pair of nodes, and thus there can be no conflicts.

Definition 2.5.5 A forest is a collection of rooted trees.

Note that every forest can impose a partial ordering on its nodes using Definition 2.5.4, but not every partial ordering can be represented by a forest. For example, $b<a,\ c<a,\ d<b,\ d<c$ is a partial ordering of $\{a,\ b,\ c,\ d\}$, which does not have a forest representation, since two paths would exist between d and a.

Consider a forest of single root nodes, corresponding to the row nodes of a bigraph before symbolic Givens reduction begins. This forest imposes the null partial ordering on its nodes. Each major step of the reduction, when using a single pivot row ordering α, can be viewed as imposing a partial ordering on the row nodes of the bigraph. Definition 2.3.3 gives this ordering. The set of nodes ordered, and their order, in major step i is given by Θ^i_α. Given the sets $\Theta^i_\alpha,\ i = 1,\ 2,\ ...,\ p,$ Algorithm 2.5.1 can be used to update a forest representation of the accumulated partial ordering after each major step i.

Algorithm 2.5.1

```
1. for $m = 1$ to $k_i - 1$ do
2.     remove edge from $\{\Theta^i_\alpha\}_m$ to its parent, if present
3.     add edge to make $\{\Theta^i_\alpha\}_m$ a child of $\{\Theta^i_\alpha\}_{m+1}$
4. endfor
```

Theorem 2.5.1 If α is a compatible row ordering, then the accumulated partial ordering after each major step can be represented by a forest generated by Algorithm 2.5.1.

Proof: Suppose we have a forest representing the accumulated partial ordering after major step $i-1$. First it will be shown that the algorithm produces a forest after major step i, and then that the forest represents the accumulated partial ordering. After major step 0, that is before major step 1, we have trivially a forest representing the null partial ordering, so by induction the theorem holds.

Only steps 2 and 3 of the algorithm affect the forest structure. If $\{\theta_\alpha^i\}_m$ has a parent, then $\{\theta_\alpha^i\}_m$ together with its descendants form a subtree, say $T_m = (X_m;E_m)$. Removing the edge to its parent creates two connected components, one of which, T_m, is a tree. The other component must also be a rooted tree, since the original structure was a rooted tree. So now $\{\theta_\alpha^i\}_m$ is the root of T_m. The node $\{\theta_\alpha^i\}_{m+1} \notin X_m$, since it cannot be a descendant of $\{\theta_\alpha^i\}_m$, because α is compatible. Ancestors or descendants of $\{\theta_\alpha^i\}_{m+1}$ also do not belong to X_m, since T_m is connected. Thus $\{\theta_\alpha^i\}_m$ and $\{\theta_\alpha^i\}_{m+1}$ belong to two disjoint rooted trees, and $\{\theta_\alpha^i\}_m$ is a root. Making $\{\theta_\alpha^i\}_m$ a child of $\{\theta_\alpha^i\}_{m+1}$ creates a single rooted tree from the two disjoint rooted trees. Thus, the algorithm preserves a forest structure.

To show that the forest represents the accumulated partial ordering, first we show that if $r \in \theta_\alpha^i$ then all ancestors of r must also belong to θ_α^i and moreover must be ordered after r. It is sufficient to show this for the parent of r. If r is a root, then it is trivially true. If r has a parent, say s, then there exists $j < i$ such that $r = \{\theta_\alpha^j\}_h$, $s = \{\theta_\alpha^j\}_{h+1}$, $2 \leq h < k_j - 1$ (Note that $h \geq 2$, since $h = 1$ gives the pivot row of major step j, which is no longer present in B_0^{i-1}.). By Theorem 2.3.1 $Adj(r,B_0^{i-1}) \subseteq Adj(s,B_0^{i-1})$, so $s \in \theta_\alpha^i$. Since α is compatible, s is ordered after r in θ_α^i. Hence if r has an ancestor r' before adjusting the forest for θ_α^i, it still has the ancestor after adjusting for θ_α^i. So, any previous partial ordering information is not altered, and clearly any new partial ordering information is recorded by step 3 of the algorithm. Thus the forest produced after processing θ_α^i represents the accumulated partial ordering after major step i. □

Let $T^i = (X^i;H^i)$ be the sequence of forests generated by Algorithm 2.5.1. This sequence of forests together with B_0^0, the initial bigraph, can generate the sequence B_0^i of bigraphs. This is stated in a theorem that follows, which is the main result of this section and forms the basis of the implementation of symbolic Givens reduction.

Definition 2.5.6 $Fam(r,T) = \{s \in X \mid s$ is a descendant of r$\} \bigcup \{r\}$, where $T = (X;H)$ is a forest. That is, $Fam(r,T)$ is the "family" of r in T consisting of all the nodes in the subtree of T rooted at r.

Theorem 2.5.2 $Adj(r,B_0^i) = Adj(Fam(r,T^i),B_0^0) \cap C^i \quad \forall \quad r \in R^i$.

Proof: For $\lambda = 2, 3, ..., k_i$, (2.4) can be written as

$$Adj(\{\theta_\alpha^i\}_\lambda,B_{k_i-1}^{i-1}) = \bigcup_{s=1}^{\lambda} Adj(\{\theta_\alpha^i\}_s,B_0^{i-1}) - \{c_i\}, \tag{2.8}$$

and note that $\{\theta_\alpha^i\}_s \in Fam(\{\theta_\alpha^i\}_\lambda,T^i)$, for $s = 1, 2, ..., \lambda$.
Suppose $c \in Adj(r,B_0^i)$, and $r \in \theta_\alpha^i$, then $c \in C^i$, and by (2.8) $c \in Adj(Fam(r,T^i),B_0^{i-1})$. If $r \notin \theta_\alpha^i$, then $Adj(r,B_0^i) = Adj(r,B_0^{i-1})$, and trivially $c \in Adj(Fam(r,T^i),B_0^{i-1})$. Now, there exists $r' \in Fam(r,T^i)$ such that $c \in Adj(r',B_0^{i-1})$, so by the same argument as above $c \in Adj(Fam(r,T^{i-1}),B_0^{i-2})$. But $Fam(r',T^{i-1}) \subset Fam(r,T^i)$, so $c \in Adj(Fam(r,T^i),B_0^{i-2})$. This can be repeated until finally $c \in Adj(Fam(r,T^i),B_0^0)$, and so
$Adj(r,B_0^i) \subseteq Adj(Fam(r,T^i),B_0^0) \cap C^i$.

Suppose now $c \in Adj(Fam(r,T^i),B_0^0) \cap C^i$, where $r \in R^i$. There exists an $r' \in Fam(r,T^i)$ such that $c \in Adj(r',B_0^0)$. Since the nodes in $Fam(r,T^i)$ form a tree rooted at r, there exists a unique path from r' to r. Let this path be $(r_1, r_2, ..., r_m)$, where $r_1 = r'$ and $r_m = r$. Each r_s is a child of r_{s+1}, $s = 1, 2, ..., m-1$. Since r_s is a child of r_{s+1}, then by construction of T^i there exists $j \leq i$ such that r_{s+1} follows r_s in θ_α^j. So if $d \in Adj(r_s,B_0^{j-1})$, then

$d \in Adj(r_{s+1}, B_0^{j'})$. Therefore, there exists $j' \leq i$ such that $c \in Adj(r, B_0^{j'})$, so that $c \in Adj(r, B_0^{i})$, and we have $Adj(Fam(r, T^i), B_0^0) \subseteq Adj(r, B_0^i)$. □

The implication of this theorem is that the symbolic Givens reduction can be performed in a fixed row oriented data structure. The initial bigraph B_0^0 need not be modified, only each successive forest structure T^i needs to be updated after each major step. The construction of T^i from T^{i-1} requires Θ_α^i, which is the ordered set $Adj(c_i, B_0^{i-1})$, where c_i is the pivot column node of major step i. Because a row oriented data structure is used, $Adj(c_i, B_0^{i-1})$ is not available directly, and must be computed. Note that $r \in Adj(c_i, B_0^{i-1})$ iff $c_i \in Adj(r, B_0^{i-1})$, so Θ_α^i can be constructed by checking if $c_i \in Adj(r, B_0^{i-1}) = Adj(Fam(r, T^{i-1}), B_0^0) \quad \forall \; r \in R^i$. The computational effort can be greatly reduced by using the information in T^{i-1} about $Adj(r, B_0^{i-1})$. Particularly, if r_1 is a descendant of r_2 in T^{i-1}, then $Adj(r_1, B_0^{i-1}) \subseteq Adj(r_2, B_0^{i-1})$. So that if $r_1 \in \Theta_\alpha^i$, then all ancestors of r_1 in T^{i-1} belong to Θ_α^i. Furthermore, if the bigraph is modified after each major step by deleting redundant edges, information about descendants can also be used in reducing the computational effort. Consider generating a sequence of bigraphs $\bar{B}_j^i$ by Algorithm 2.5.2, defining $\bar{B}_0^0 = B_0^0$. In major step i, $\{\Theta_\alpha^i\}_{j-1}$ becomes a descendant of $\{\Theta_\alpha^i\}_j$, so any column nodes in $Adj(Fam(\{\Theta_\alpha^i\}_{j-1}, T^{i-1}), \bar{B}_{j-1}^{i-1})$ can be removed from $Adj(\{\Theta_\alpha^i\}_j, \bar{B}_{j-1}^{i-1})$, while maintaining the structure of $Adj(Fam(\{\Theta_\alpha^i\}_j, T^{i-1}), \bar{B}_{j-1}^{i-1})$ unchanged. The array P_j is initialized to all zeros, and keeps track of pivot rows, since these become part of the matrix factor R. The pivot row nodes are not removed, since they still contribute to the structure of their ancestor nodes through the Fam operator. However, in order to have a forest in which every node is "available" (has a zero entry in P), a supernode is formed from each pivot row node and its parent (which has a zero entry in P), and the parent is the representative of this supernode. Elements of each supernode are represented by a linked list starting with each supernode representative. An exception is when $k_i = 1$ in a major step. In this case, the pivot row supernode no longer contributes to the structure of any other rows, and the corresponding entries in P are set to 2 to indicate this.

Algorithm 2.5.2

```
 1. if $k_i = 1$ then set $P$ to 2 for each member of supernode $\{\Theta_\alpha^i\}_1$
 2. else
 3.     for $j = 2$ to $k_i$ do
 4.         for each $r$ in supernode $\{\Theta_\alpha^i\}_j$ do
 5.             $Adj(r,\bar{B}_{j-1}^{i-1}) \leftarrow Adj(r,\bar{B}_{j-2}^{i-1}) - Adj(Fam(\{\Theta_\alpha^i\}_{j-1},T^{i-1}),\bar{B}_{j-2}^{i-1})$
 6.         endfor
 7.     endfor
 8.     link supernode $\{\Theta_\alpha^i\}_1$ to supernode $\{\Theta_\alpha^i\}_2$
 9.     for each $r$ in supernode $\{\Theta_\alpha^i\}_1$ do
10.         $\Phi \leftarrow Fam(\{\Theta_\alpha^i\}_2,T^{i-1}) - Fam(\{\Theta_\alpha^i\}_1,T^{i-1})$
11.         $Adj(r,\bar{B}_0^i) \leftarrow Adj(r,\bar{B}_{k_i-1}^{i-1}) - Adj(\Phi,\bar{B}_{k_i-1}^{i-1}) - \{c_i\}$
12.     endfor
13.     $P_{\{\Theta_\alpha^i\}_1} \leftarrow 1$
```

Note that, as $\bar{B}_{j-1}^{i-1}$ is formed from $\bar{B}_{j-2}^{i-1}$ in the loop of step 5, only the adjacency structure of row nodes in supernode $\{\Theta_\alpha^i\}_j$ is modified. Step 8 is accomplished by linking the end of supernode $\{\Theta_\alpha^i\}_2$ list to node $\{\Theta_\alpha^i\}_1$. After this link is made, each r in supernode

$\{\theta_\alpha^i\}_1$ must satisfy $Adj(r,\bar{B}_0^i) \cap Adj(s,\bar{B}_0^i) = \varnothing$, for all $s \in Fam(\{\theta_\alpha^i\}_2,T^{i-1}) - Fam(\{\theta_\alpha^i\}_1,T^{i-1})$. The necessary adjustment is in the loop of step 10. A similar relationship, as in Theorem 2.5.2, holds for this new sequence of bigraphs.

Corollary 2.5.1 $Adj(r,B_0^i) = Adj(Fam(r,T^i),\bar{B}_0^i) \quad \forall \quad r \in R^i$.

Proof: It is enough to prove $Adj(Fam(r,T^i),B_0^0) \cap C^i = Adj(Fam(r,T^i),\bar{B}_0^i)$

$\forall \quad r \in R^i$, and use Theorem 2.5.2.

The equation holds for $i = 0$. Suppose it holds for $i = m$. If $r \notin \theta_\alpha^{m+1}$, then it clearly holds for $i = m+1$. If $r \in \theta_\alpha^{m+1}$, and say $r = \{\theta_\alpha^{m+1}\}_h$,

then note that $$Fam(r,T^{m+1}) = \bigcup_{s=1}^{h} Fam(\{\theta_\alpha^{m+1}\}_s,T^m),$$

so that
$$Adj(Fam(r,T^{m+1}),B_0^0) \cap C^i = Adj\left(\bigcup_{s=1}^{h} Fam(\{\theta_\alpha^{m+1}\}_s,T^m),B_0^0\right) \cap C^i$$

$$= \bigcup_{s=1}^{h} Adj(Fam(\{\theta_\alpha^{m+1}\}_s,T^m),B_0^0) \cap C^i$$

$$= \bigcup_{s=1}^{h} Adj(Fam(\{\theta_\alpha^{m+1}\}_s,T^m),\bar{B}_0^m) \cap C^i$$

$$= Adj(Fam(r,T^{m+1}),\bar{B}_0^m) \cap C^i$$

$$= Adj(Fam(r,T^{m+1}),\bar{B}_0^{m+1}).$$

Hence, by induction the relationship holds for all i. □

The sequence of bigraphs $\bar{B}_0^i$ has the nice property, that if $c_i \in Adj(r,\bar{B}_0^{i-1})$, then $c_i \notin Adj(\{\text{ancestors and descendants of r}\},\bar{B}_0^{i-1})$. That is, if r_1 is an ancestor of r_2, then $Adj(r_1,\bar{B}_0^{i-1}) \cap Adj(r_2,\bar{B}_0^{i-1}) = \varnothing$. This aids in the search for r such that $c_i \in Adj(r,B_0^{i-1})$. When we find an r such that $c_i \in Adj(r,\bar{B}_0^{i-1})$, then we immediately know that ancestors of r belong to Θ_α^i and descendants of r do not belong to Θ_α^i. Also note that when forming Θ_α^i, r must be ordered before its ancestors, so it is convenient to find it first. These advantages are at the expense of extra work done in generating the $\bar{B}_j^i$ sequence. However, the extra work done in a given major step is useful not only in the next major step, but in a number of subsequent major steps. Algorithm 2.5.3 performs symbolic Givens reduction incorporating the above ideas.

Algorithm 2.5.3

```
1.  $\bar{B}_0^0 \leftarrow B_0^0 = (R^0,C^0;E_0^0)$
2.  $X^0 \leftarrow R^0$;  $H^0 \leftarrow \varnothing$;  $T^0 = (X^0;H^0)$
3.  initialize array $P$ to zero
4.  for $i = 1$ to $p$ do
5.      initialize array $I$ to zero
6.      choose $c_i$
7.      for $j = 1$ to $n$ do
8.          if $I_j = 0$ and $P_j < 2$ then
9.              if $c_i \in Adj(r_j,\bar{B}_0^{i-1})$ then
10.                 if $P_j = 0$ then $r \leftarrow r_j$
```

```
11.                 else $r \leftarrow$ ancestor of $r_j$ with zero entry in $P$
12.                 $\Theta^i \leftarrow \Theta^i \bigcup r$
13.                 set $I_s \leftarrow 1 \ \forall \ s$ such that $r_s \in Fam(r,T^{i-1})$
14.                 set $I_s \leftarrow 1 \ \forall \ s$ such that $r_s$ ancestor of $r$ in $T^{i-1}$
15.             endif
16.         endif
17.     endfor
18.     order $\Theta^i$ and its ancestors in $T^{i-1}$ to form ordered $\Theta^i_\alpha$
19.     use Algorithm 2.5.1 to form $T^i$ from $T^{i-1}$ using $\Theta^i_\alpha$
20.     use Algorithm 2.5.2 to form $\bar{B}^i_0$ from $\bar{B}^{i-1}_0$ using $\Theta^i_\alpha$
21. endfor
```

The sequence of bigraphs $\bar{B}^i_j$ is represented in a row oriented data structure. Adjacency lists for each $r \in R^0$ are stored sequentially in a one dimensional array of length $|E^0_0|$, and pointers to the beginning of each adjacency list are stored in an array of length $n + 1$. Steps 5 and 10 of Algorithm 2.5.2 are accomplished by making the appropriate entries in the adjacency lists negative. The sequence of forests, T^i, requires $3n$ storage locations, and is stored in a triply linked tree form (see for example Knuth (1968)) to facilitate fast searching. Steps 6 and 18 in Algorithm 2.5.3 are dependent on the column ordering and row ordering strategies respectively.

The representation of symbolic redction in Algorithm 2.5.3 is not simple to explain, and an example is needed to aid the above explanations. The computer implementation of the algorithm is discussed in Section 5.1, where an example of the reduction process is also given. All of $\bar{B}_0^i$, B_0^i, and T^i are displayed for a few stages of the reduction on a sparse matrix.

Implementation of the minimum degree column ordering strategy requires additional p storage locations for the column degree. As outlined in Section 2.4, the degree of a column node c after major step i is $|Bireach(c,B_0^i)|$, which is given by Definition 2.4.2. Note that after each major step, only the degree of column nodes in $Adj(\{\theta_a^i\}_{k_i},B_0^i)$ needs to be updated. Algorithm 2.5.4 performs this update for a column node c.

Algorithm 2.5.4

```
1.  for $j = 1$ to $n$ set $I_j = 0$
2.  $\theta \leftarrow \emptyset$
3.  for $j = 1$ to $n$ do
4.      if $I_j = 0$ then
5.          if $c \in Adj(r_j,\bar{B}_0^i)$ then
6.              find root $r$ of tree containing $r_j$
7.              $\theta \leftarrow \theta \bigcup Adj(Fam(r,T^i),\bar{B}_0^i)$
8.              $I_s \leftarrow 1 \quad \forall \; s$ such that $r_s \in Fam(r,T^i)$
9.          endif
10.     endif
11. endfor
12. $|\theta|$ gives the degree of $c$
```

Some column ordering strategies, as well as the minimum degree tiebreaking strategy, require the number of nonzeros in a given column. This information is not available without some computation, since a row oriented data structure is used. Algorithm 2.5.5 obtains d the number of nonzeros in column c of B_0^i.

Algorithm 2.5.5

```
 1. for $j = 1$ to $n$ set $I_j = 0$
 2.  $d \leftarrow 0$
 3. for $j = 1$ to $n$ do
 4.     if $I_j = 0$ then
 5.         if $c \in Adj(r_j, \bar{B}_0^i)$ then
 6.             $m \leftarrow j$
 7.             while $I_m = 0$ do
 8.                 $I_m \leftarrow 1$
 9.                 $d \leftarrow d + 1$
10.                 if $r_m$ has a parent then $m \leftarrow$ index of parent
11.             endwhile
12.             $I_s \leftarrow 1$  $\forall$ $s$ such that $r_s \in Fam(r_j, T^i)$
13.         endif
14.     endif
15. endfor
```

Any row ordering strategy used has to be compatible, as the algorithms of this section are based on this assumption, and should be locally acceptable due to the results of Section 2.3. To ensure compatibility, Θ^i before step 18 in Algorithm 2.5.3 contains only those row nodes, which are competing for the first place under the compatibility restriction. Once the first node is selected, it is replaced in Θ^i by its ancestor, and Θ^i now contains the nodes competing for the second place. This continues until all ancestors are exhausted and Θ^i is empty. A row ordering strategy thus decides, at any one time, only between the row nodes in Θ^i. Algorithm 2.5.6 performs the ordering without explicitly specifying the strategy used in step 2.

Algorithm 2.5.6

```
1. while $\Theta^i \neq \emptyset$ do
2.     let $r$ be the next selected row from $\Theta^i$
3.     $\Theta^i \leftarrow \Theta^i - \{r\}$
4.     $\Theta^i \leftarrow \Theta^i \bigcup parent(r)$
5. endwhile
```

The question of which rows should be left out from the initial factorization can be addressed in step 5 of Algorithm 2.5.2. The size of the set $Adj(\{\Theta^i_\alpha\}_j, \bar{B}^{i-1}_{j-1})$ measures how "different" the row $\{\Theta^i_\alpha\}_j$ is from rows involved in previous rotations within the current major step. If this set is "large", especially for large j, the row $\{\Theta^i_\alpha\}_j$ should be left out from the initial factorization. Changes to B^0_0 were made only by making some entries negative, so the factorization can be restarted by taking absolute values of all entries of $\bar{B}^{i-1}_{j-1}$ and marking $\{\Theta^i_\alpha\}_j$ as unavailable in P. It seems that a partial restart should be possible, using the information in T^{i-1}, thus saving some of the previous computations. Further research is needed into this question.

As was mentioned at the outset of this chapter, and shown in Figure 2.1 (b), it is possible that the matrix X has less than full structural rank. The symbolic Givens factorization algorithm will find the structural rank of X, which is an upper bound on the actual rank of X. In step 6 of Algorithm 2.5.3, if every row node j with $P_j = 0$ has no edges left, the unreduced portion of the matrix X is a null matrix. The structural rank of X is given by the value of i at this point.

When using one of the row ordering strategies given in Section 2.3, the resulting row ordering is determined in two parts. First, the pivot rows are ordered in the order they are marked in P. Then, the remainder of the rows is ordered after the pivot rows, as determined by T^{p-m}, where m is the structural rank deficiency. The column order is, of course, determined by step 4 of Algorithm 2.5.5, and the structure of each row of the matrix factor R is given by

$Adj(\{\Theta_{\alpha}^{i}\}_{k_i}, B_0^i) \cup \{c_i\}, \quad i = 1, 2, ..., p-m.$

3. UPDATING A LEAST SQUARES SOLUTION

It is often the case that a few rows of the X matrix are the cause of much fill-in in the R factor. To avoid the fill-in, these rows should be left out in the initial factorization, and then used to update the solution. Chapter 2 suggests a method for deciding which rows to leave out. Equality constraints often consist of a few very dense rows, so they may likely be among the rows left out. For example, if all parameters must sum to a constant, then we have a completely full row. The constraints can be treated as additional observations, but they have to be satisfied exactly rather than in the least squares sense.

Normally, when operating with full matrices, adding observations is no problem. The R factor can be modified by additional Givens transformations, and a new solution computed. However, when dealing with a sparse matrix, this modification will produce unacceptable fill in the R factor, since this is why these rows were left out from the initial factorization. The methods discussed in this chapter are special, in the sense that they only modify the solution, not the R factor, while using a minimal amount of additional storage.

Heath (1982) gives a method for updating a solution for the Givens algorithm using the computed R factor. His method allows for equality constraints, but assumes that X, and hence R, is full rank. Björck and Duff (1980) give an updating method in the context of a different basic algorithm (Peters-Wilkinson LU factorization), which also allows for equality constraints. Their method is more general, as it makes no assumptions about the rank of X. In this chapter, Section 3.1 extends the updating method of Heath to rank deficient problems, and Section 3.2 discusses the inclusion of equality constraints.

During research on the methods of this chapter, a result was obtained on the nonzero structure of the inverse of a triangular matrix. This result was not used in the final version of these methods. However, the result is interesting by itself, and so it was put into Appendix B.

3.1 Updating with Additional Observations

The problem considered here is a least squares solution of

$$\begin{bmatrix} X \\ E \end{bmatrix} b \doteq \begin{bmatrix} y \\ z \end{bmatrix}, \tag{3.1}$$

where X is an $n \times p$ sparse matrix of rank $r \leq min(n,p)$, E is a $q \times p$ matrix, b is a $p \times 1$ vector, y is a $n \times 1$ vector, and z is a $q \times 1$ vector. Initially, a least squares solution of $X\tilde{b} \doteq y$ is produced using Heath's (1982) extension of the Givens algorithm. Then $\tilde{b}$ is updated by the additional rows in E to produce b.

For simplicity of presentation, assume that the first r columns of X are linearly independent. First, X is factored using Givens rotations as

$$X = Q \begin{bmatrix} R & T \\ O & O \end{bmatrix}, \tag{3.2}$$

where Q is a product of orthogonal Givens rotation matrices of order n, and R is upper triangular of order r. Partition Q as $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$, where Q_1 is $n{\times}r$. Then,

$$X = Q_1 \begin{bmatrix} R & T \end{bmatrix}.$$

Applying the same transformations to the right hand side, we obtain

$$y = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}. \tag{3.3}$$

A solution $\tilde{b} = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{bmatrix}$ is then obtained by solving $R\tilde{b}_1 = c$, and setting $\tilde{b}_2 = 0$. Let $r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} X \\ E \end{bmatrix} b$ be the residual for the updated solution, and let $\tilde{r} = \begin{bmatrix} \tilde{r}_1 \\ \tilde{r}_2 \end{bmatrix}$ be the corresponding residual for solution $\tilde{b}$.

Then,
$$\begin{aligned}\tilde{r}_1 &= y - X\tilde{b} \\ &= Q_1c + Q_2d - Q_1\begin{bmatrix} R & T \end{bmatrix}\begin{bmatrix} \tilde{b}_1 \\ 0 \end{bmatrix} \\ &= Q_1c + Q_2d - Q_1R\tilde{b}_1 \\ &= Q_2d, \end{aligned} \tag{3.4}$$

and
$$\begin{aligned}\tilde{r}_2 &= z - E\tilde{b} \\ &= z - \begin{bmatrix} E_1 & E_2 \end{bmatrix}\begin{bmatrix} \tilde{b}_1 \\ 0 \end{bmatrix} \\ &= z - E_1\tilde{b}_1, \end{aligned} \tag{3.5}$$

where $\begin{bmatrix} E_1 & E_2 \end{bmatrix}$ is a conforming partition of E.

Now define K and M by $R'K = E_1'$ and $RM = T$ respectively. Then, K is an $r \times q$ matrix, and M is an $r \times p-r$ matrix. Also, let $H = E_2 - K'RM$. Note that the rank of H is the increase in the rank of the solution due to the update. To see this, note that

$$\begin{aligned} rank\begin{bmatrix} X \\ E \end{bmatrix} &= rank\begin{bmatrix} R & T \\ E_1 & E_2 \end{bmatrix} \\ &= rank\begin{bmatrix} R & RM \\ K'R & E_2 \end{bmatrix} \\ &= rank\left\{\begin{bmatrix} I_r & O \\ K' & I_q \end{bmatrix}\begin{bmatrix} R & RM \\ O & E_2 - K'RM \end{bmatrix}\right\} \\ &= rank\begin{bmatrix} R & RM \\ O & E_2 - K'RM \end{bmatrix} \\ &= rank(R) + rank(E_2 - K'RM) \\ &= r + rank(H). \end{aligned}$$

Let $b = \bar{b} + \delta$, and partition $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$, $\bar{b} = \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \end{bmatrix}$, and $\delta = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$, so that all partitions conform.

Then
$$\begin{aligned} r_1 &= \bar{r}_1 - X\delta \\ &= Q_2 d - Q_1 \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \\ &= Q_2 d - Q_1 \begin{bmatrix} R\delta_1 + RM\delta_2 \end{bmatrix} \\ &= Q_2 d - Q_1 Rf, \end{aligned} \tag{3.6}$$

and
$$\begin{aligned} r_2 &= \bar{r}_2 - E\delta \\ &= \bar{r}_2 - E_1\delta_1 - E_2\delta_2 \\ &= \bar{r}_2 - K'R\delta_1 - H\delta_2 - K'RM\delta_2 \\ &= \bar{r}_2 - K'Rf - H\delta_2, \end{aligned} \tag{3.7}$$

where $f = \delta_1 + M\delta_2$.

Since Q is orthogonal, $\|r_1\|_2 = \|Q_2 d\|_2 + \|Q_1 Rf\|_2$

$$= \|d\|_2 + \|Rf\|_2.$$

So the least squares solution of (3.1) is given by

$$\min_{b} \left\| \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \right\|_2 = \min_{f,\delta_2} \left\| \begin{bmatrix} Rf \\ \bar{r}_2 - K'Rf - H\delta_2 \end{bmatrix} \right\|_2 . \tag{3.8}$$

Substituting $u = Rf$, (3.8) becomes

$$\min_{u,\delta_2} \left\| \begin{bmatrix} u \\ \bar{r}_2 - K'u - H\delta_2 \end{bmatrix} \right\|_2 . \tag{3.9}$$

For a fixed δ_2, (3.9) can be written as

$$\min \left\| \begin{bmatrix} u \\ v \end{bmatrix} \right\|_2 \quad \text{subject to} \quad K'u + v = \bar{r}_2 - H\delta_2, \tag{3.10}$$

and the solution to this is given by the minimum norm solution to

$$\begin{bmatrix} K' & I_q \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \bar{r}_2 - H\delta_2. \tag{3.11}$$

Note that when $r = p$, then $H = 0$ and (3.11) reduces to an expression obtained by Heath (1982) for the full rank updating problem. This can be solved by orthogonal factorization

$$U' \begin{bmatrix} K \\ I_q \end{bmatrix} = \begin{bmatrix} L' \\ O \end{bmatrix}, \quad \text{and}$$

$$U' \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s \\ t \end{bmatrix}, \tag{3.12}$$

where U is an orthogonal matrix of order $q + r$, and L is a lower triangular matrix of order q. The minimum norm problem now becomes

$$\begin{bmatrix} L & O \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \bar{r}_2 - H\delta_2,$$

which is solved by setting $t = 0$ and solving the triangular system $Ls = \bar{r}_2 - H\delta_2$. In terms of s and t (3.10) now becomes

$$\min \left\| \begin{bmatrix} s \\ t \end{bmatrix} \right\|_2 \quad \text{subject to} \quad Ls = \bar{r}_2 - H\delta_2,$$

So now (3.9) is solved for u in terms of δ_2, and becomes

$$\min_{\delta_2} \|L^{-1}\bar{r}_2 - L^{-1}H\delta_2\|_2 . \tag{3.13}$$

But this is a least squares problem, which can be solved by another orthogonal factorization. Let $S = L^{-1}H$, and $w = L^{-1}\bar{r}_2$, so that

$$F'S = \begin{bmatrix} B & Z \\ O & O \end{bmatrix},$$

$$F'w = \begin{bmatrix} g \\ h \end{bmatrix}, \tag{3.14}$$

where F is an orthogonal matrix of order q, B is an upper triangular matrix of order $k = rank(H)$, g is a vector of length k, and remaining matrices conform. Partition δ_2 into $\begin{bmatrix} \delta_{21} \\ \delta_{22} \end{bmatrix}$, where δ_{21} is a vector of length k. Then, δ_2 minimizing (3.13) is obtained by setting $\delta_{22} = 0$, and solving the triangular system

$$B\delta_{21} = g \ . \tag{3.15}$$

The solution to original problem (3.1) is then obtained by solving for δ_1 and setting $b = \bar{b} + \delta$. The preceding development of the updating problem gives the following algorithm.

Algorithm 3.1.1

1. Obtain R, T, and c as defined in (3.2) and (3.3) using the Heath (1982) algorithm.
2. Solve $R\bar{\delta}_1 = c$.
3. $\bar{r}_2 = z - E_1\bar{\delta}_1$.
4. Solve $R'K = E_1'$.
5. $H = E_2 - K'T$.
6. Compute orthogonal factorization $U'\begin{bmatrix} K \\ I_q \end{bmatrix} = \begin{bmatrix} L' \\ O \end{bmatrix}$.
7. Solve $LS = H$ and $Lw = \bar{r}_2$.
8. Compute orthogonal factorization $F'\begin{bmatrix} S & w \end{bmatrix} = \begin{bmatrix} B & Z & g \\ O & O & h \end{bmatrix}$.
9. Solve $B\delta_{21} = g$.

10. Solve $Ls = \bar{r}_2 - H\begin{bmatrix} \delta_{21} \\ 0 \end{bmatrix}$.

11. Compute u in $\begin{bmatrix} u \\ v \end{bmatrix} = U\begin{bmatrix} s \\ 0 \end{bmatrix}$

12. Solve $Rf = u$.

13. Solve $RM = T$.

14. $\delta_1 = f - M\begin{bmatrix} \delta_{21} \\ 0 \end{bmatrix}$.

15. $b = \begin{bmatrix} \bar{b}_1 + \delta_1 \\ \delta_{21} \\ 0 \end{bmatrix}$.

Note that in step 6 of the algorithm, only the first q rows of the matrix U are needed for the calculation of step 11, so only the q rows need to be stored. Any matrix requiring additional storage, over that needed by R and T, in this algorithm has dimensions at most $q \times p$. Assuming that q is small, the calculations can be performed in full storage mode, with the exception of those involving R and T, which are stored in sparse storage mode.

The development leading to the above algorithm assumes that exact arithmetic is used in all calculations. With finite precision arithmetic of a computer, the rank of X and that of H is estimated. Heath (1982) discusses the problem of estimating the rank of X in his algorithm, and concludes, based on a number of test cases, that the algorithm performs well. In Algorithm 3.1.1, step 8 estimates the increase in rank due to the update. The orthogonal factorization of the $q \times p-r$ matrix S can be done by Householder transformations with pivoting for stability, so no problem should arise here. However H itself is computed by taking a difference in step 5, and potentially some cancellation could occur here due to finite precision arithmetic. Some testing of this algorithm will be discussed in Chapter 5.

3.2 Updating with Equality Constraints

Here we consider the least squares solution of

$$\begin{bmatrix} X \\ E \end{bmatrix} b \doteq \begin{bmatrix} y \\ z \end{bmatrix} \qquad \text{subject to} \quad Gb = a, \tag{3.16}$$

where X, E, y, and z are the same as in Section 3.1, G is an $m \times p$ $(m < p)$ matrix, and a is an $m \times 1$ vector. Again, for simplicity of presentation, it is assumed that the first r columns of X are linearly independent.

Let G be of full rank m, and partition G into $\begin{bmatrix} G_1 & G_2 \end{bmatrix}$, where G_2 is the first r columns of G. Define J by $R'J = G_1'$, so that J is an $r \times m$ matrix, and let $N = G_2 - J'RM$, giving an $m \times p-r$ matrix.

At this point it is convenient to comment on estimability of the constraints. First note that the rowspace of X is the same as the rowspace of $\begin{bmatrix} R & RM \end{bmatrix}$. Also, from the above definitions

$$\begin{bmatrix} E_1 & E_2 \\ G_1 & G_2 \end{bmatrix} = \begin{bmatrix} K'R & K'RM+H \\ J'R & J'RM+N \end{bmatrix}.$$

The first situation of interest occurs when N is a zero matrix. Then,

$$G = J'\begin{bmatrix} R & RM \end{bmatrix},$$

and so the constraints are jointly estimable in the initial problem. If N is nonzero, but there exists a matrix C such that $N = CH$, then

$$G = \begin{bmatrix} J'-CK' & C \end{bmatrix} \begin{bmatrix} R & RM \\ E_1 & E_2 \end{bmatrix}.$$

So the constraints are jointly estimable in the problem updated by the additional observations E, but not jointly estimable in the initial problem. The third situation of interest is when the constraints are all nonestimable. This occurs when no row of N is a linear combination of the rows of H.

The initial solution $\bar{\delta}$ is obtained as in Section 3.1. Let r_1 and r_2 be as in (3.6) and (3.7) respectively, and define $\bar{r}_3$ and r_3 by

$$\bar{r}_3 = a - G\bar{\delta} = a - G_1\bar{\delta}_1 ,$$

and

$$\begin{aligned} r_3 &= \bar{r}_3 - G\delta \\ &= \bar{r}_3 - G_1\delta_1 - G_2\delta_2 \\ &= \bar{r}_3 - J'R\delta_1 - N\delta_2 - J'RM\delta_2 \\ &= \bar{r}_3 - J'Rf - N\delta_2 \\ &= \bar{r}_3 - J'u - N\delta_2. \end{aligned}$$

The constraints have to be satisfied exactly, so $r_3 = 0$, and

$$J'u = \bar{r}_3 - N\delta_2.$$

This adds a constraint on (3.8), and on (3.9) which becomes

$$\min_{u,\delta_2} \left\| \begin{bmatrix} u \\ \bar{r}_2 - K'u - H\delta_2 \end{bmatrix} \right\|_2 \quad \text{subject to} \quad J'u = \bar{r}_3 - N\delta_2. \tag{3.17}$$

So for a fixed δ_2, for which the above constraints are consistent, this can be written as

$$\min \left\| \begin{bmatrix} u \\ v \end{bmatrix} \right\|_2 \quad \text{subject to} \quad \begin{aligned} K'u + v &= \bar{r}_2 - H\delta_2 \\ J'u &= \bar{r}_3 - N\delta_2. \end{aligned} \tag{3.18}$$

The solution to (3.18) is given by the minimum norm solution to

$$\begin{bmatrix} K' & I_q \\ J' & O \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \bar{r}_2 - H\delta_2 \\ \bar{r}_3 - N\delta_2 \end{bmatrix}. \tag{3.19}$$

Orthogonal factorization as in (3.12) may not be sufficient to solve this minimum norm problem, since J may not be full rank. The rank of J is the same as the rank of G_1. Assume that $m < r$, and $rank(G_1) = j \leqslant m$. Then, there exists a nonsingular matrix V of order m, such that $V_2G_1 = 0$, where V_1 and V_2 are the first j and the last $m - j$ rows of V respectively. Then $V_2J' = 0$, and so

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \begin{bmatrix} G_1 & G_2 \end{bmatrix} = \begin{bmatrix} V_1G_1 & V_1G_2 \\ O & V_2N \end{bmatrix}. \tag{3.20}$$

Of course if $j = m$, then V_2 is a null matrix, and $V_1 = V$ can be the identity matrix. Premultiplying (3.19) by

$$\begin{bmatrix} I_q & O \\ O & V_1 \\ O & V_2 \end{bmatrix} \tag{3.21}$$

gives

$$\begin{bmatrix} K' & I_q \\ V_1J' & O \\ O & O \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \bar{r}_2 - H\delta_2 \\ V_1\bar{r}_3 - V_1N\delta_2 \\ V_2\bar{r}_3 - V_2N\delta_2 \end{bmatrix}. \tag{3.22}$$

Note that the fixed δ_2 must satisfy

$$V_2\bar{r}_3 = V_2N\delta_2 ,$$

which is the same as

$$V_2a = V_2N\delta_2 .$$

Now apply orthogonal factorization to (3.22), so that

$$U' \begin{bmatrix} K & JV_1' & O \\ I_q & O & O \end{bmatrix} = \begin{bmatrix} L' & O \\ O & O \end{bmatrix}, \tag{3.23}$$

and

$$U' \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s \\ t \end{bmatrix},$$

where U is an orthogonal matrix of order $r + q$, L is a lower triangular matrix of order $q + j$, s is a vector of length $q + j$, and t is a vector of length $r - j$. The minimum norm problem (3.22) now becomes

$$\begin{bmatrix} L & O \\ O & O \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} \bar{r}_2 - H\delta_2 \\ V_1\bar{r}_3 - V_1N\delta_2 \\ V_2\bar{r}_3 - V_2N\delta_2 \end{bmatrix}. \tag{3.24}$$

Let e be the first $q + j$ elements of the right hand side of (3.24). The solution is then given by setting $t = 0$, and solving the triangular system $Ls = e$. Using this solution, (3.17) becomes a constrained least squares problem in δ_2,

$$\min_{\delta_2} \|w - S\delta_2\|_2 \qquad \text{subject to} \quad v = C\delta_2, \tag{3.25}$$

where $S = L^{-1}\begin{bmatrix} H \\ V_1N \end{bmatrix}$, $w = L^{-1}\begin{bmatrix} \bar{r}_2 \\ V_1\bar{r}_3 \end{bmatrix}$, $v = V_2a$, and $C = V_2N$.

In the case when $j = m$, the constraints are not present, as V_2 is a null matrix, and orthogonal decomposition (3.14) will solve the problem.

If $j < m$, since G is full rank and $rank(G_1) = j$, then by (3.20) V_2N has full rank $m - j$. This can be solved using a procedure given in, for example, Lawson and Hanson (1974). First, define the partitions

$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix}, \quad S = \begin{bmatrix} S_1 & S_2 \end{bmatrix}, \quad \text{and} \quad \delta_2 = \begin{bmatrix} \delta_{21} \\ \delta_{22} \end{bmatrix},$$

where C_1 is a square matrix of order $m - j$, S_1 is a $(q+j) \times (m-j)$ matrix, and all other partitions conform. For simplicity of presentation assume that the first $m - j$ columns of C are linearly independent. Solve the constraints for δ_{21} in terms of δ_{22}, and substitute into (3.25) giving an unconstrained least squares problem

$$\min_{\delta_{22}} \|(w - S_1C_1^{-1}v) - (S_2 - S_1C_1^{-1}C_2)\delta_{22}\|_2.$$

To simplify this, consider a factorization

$$\begin{bmatrix} C_1 & C_2 & v \end{bmatrix} = D'\begin{bmatrix} \tilde{C}_1 & \tilde{C}_2 & \tilde{v} \end{bmatrix}, \tag{3.26}$$

where D is an orthogonal matrix of order $m-j$, and $\tilde{C}_1$ is upper triangular. Solve a triangular system of equations $\tilde{S}_1\tilde{C}_1 = S_1$, and compute $\tilde{w} = w - \tilde{S}_1\tilde{v}$, and $\tilde{S}_2 = S_2 - \tilde{S}_1\tilde{C}_2$. So, now the unconstrained least squares problem becomes

$$\min_{\delta_{22}} \|\tilde{w} - \tilde{S}_2\delta_{22}\|_2,$$

which is solved by orthogonal decomposition like (3.14). Then, δ_{21} is obtained by solving the triangular system

$$\tilde{C}_1\delta_{21} = \tilde{v} - \tilde{C}_2\delta_{22}.$$

The solution to problem (3.16) is then obtained by solving for δ_1 and setting $b = \tilde{b} + \delta.$

The preceding development leads to Algorithm 3.2.1.

Algorithm 3.2.1

1. Perform steps 1 to 5 of Algorithm 3.1.1.

2. $\tilde{r}_3 = a - G_1\tilde{b}_1.$

3. Solve $R'J = G_1'.$

4. $N = G_2 - J'T.$

5. Compute factorization (3.20) to obtain V_1, V_2, and C. If V_2 is null, set $V_1 = I$, skip steps 8 to 12, and 15, and set $\delta_{21} = \emptyset$.

6. Compute orthogonal factorization (3.23).

7. Solve $LS = \begin{bmatrix} H \\ V_1 N \end{bmatrix}$, and $Lw = \begin{bmatrix} \bar{r}_2 \\ V_1 \bar{r}_3 \end{bmatrix}$.

8. $\nu = V_2 a$.

9. Compute orthogonal factorization (3.26)

10. Solve $\bar{S}_1 \bar{C}_1 = S_1$.

11. $\bar{w} = w - \bar{S}_1 \bar{\nu}$.

12. $\bar{S}_2 = S_2 - \bar{S}_1 \bar{C}_2$.

13. Compute orthogonal factorization $F' \begin{bmatrix} \bar{S}_2 & \bar{w} \end{bmatrix} = \begin{bmatrix} B & Z & g \\ O & O & h \end{bmatrix}$.

14. Solve $B \delta_{221} = g$.

15. Solve $\bar{C}_1 \delta_{21} = \bar{\nu} - \bar{C}_2 \begin{bmatrix} \delta_{221} \\ 0 \end{bmatrix}$.

16. $\delta_2 = \begin{bmatrix} \delta_{21} \\ \delta_{221} \\ 0 \end{bmatrix}$.

17. Solve $Ls = \bar{r}_2 - H\delta_2$.

18. Compute u in $\begin{bmatrix} u \\ v \end{bmatrix} = U \begin{bmatrix} s \\ 0 \end{bmatrix}$.

19. Solve $Rf = u$.

20. Solve $RM = T$.

21. $\delta_1 = f - M\delta_2$.

22. $b = \begin{bmatrix} \bar{\delta}_1 + \delta_1 \\ \delta_2 \end{bmatrix}.$

Similarly as in Algorithm 3.1.1, any matrix requiring additional storage, over that needed by R and T, has dimensions at most $q+m \times p$. So, assuming that $q + m$ is small, full storage can again be used. Note that this algorithm is quite similar to Algorithm 3.1.1 except for the complication due to the constraint in (3.25). Steps 5, 8 through 12, and 15 deal with this complication. If V_1 turns out to be a full rank matrix in step 5, then the constraint is not present, and V_1 can be set to the identity matrix. This has the effect, that steps 8 through 12, and 15 are not needed.

4. SPARSE MATRIX TECHNIQUES IN ANALYSIS OF VARIANCE

Regression in balanced designed experiments can be accomplished very efficiently by existing algorithms, which do not form the data matrix X explicitly. However, when the design is unbalanced, either due to missing observations or heteroschedasticity, the data matrix X or $X'X$ has to be formed explicitly. Generally, the matrices are stored in full storage mode. For a large model with many levels and interactions, both X and $X'X$ are very large and sparse matrices.

Gentleman (1973) reports that solving this least squares problem by Givens reduction of X has advantages and is quite efficient. His method was simply exploiting zeros in full storage mode, without considering any sparse matrix techniques. The matrix X, when the unbalance is due to missing observations, has also another property which should be exploited. All the nonzeros are ones.

Section 4.1 discusses sparse matrix techniques in Givens reduction of a model matrix, and then Section 4.2 looks at sparse matrix techniques in analysis of variance, and methods of obtaining estimable functions.

4.1 Givens Reduction of a Model Matrix

Normally, the symbolic reduction only obtains the row and column ordering and passes a data structure to the numerical reduction. The bigraph does not have any information on the values of nonzeros. In the case of a dummy variable matrix, the situation is different. The initial bipartite graph contains all information about the matrix, because all its nonzeros are ones. It may even seem that Theorem 2.5.2 can have applications here beyond giving just the nonzero structure of a partially factored matrix. However, as the factorization proceeds, the nonzeros become very rapidly diverse. More information is needed, than just the forest structure T^i, to construct a partially factored model matrix from B_0^0. The situation, though, is not completely hopless. In the initial stages of the reduction, the symbolic stage can perform some limited numerical work, although not through Theorem 2.5.2, as will be seen below.

The symbolic reduction of Chapter 2 assumed that no numerical cancellation takes place in (2.2.1). When this assumption fails, we simply obtain an upper bound on the nonzero structure. This upper bound is very good in general, but in the special case, when reducing a matrix of dummy variables, a large amount of cancellation can take place.

Definition 4.1.1 Numerical cancellation occurs at x_{ij} whenever

a. x_{ik} is the pivot element, x_{sk} is the element to be annihilated, and

$x_{ik}x_{ij} = -x_{sk}x_{sj}$, or

b. x_{sk} is the pivot element, x_{ik} is the element to be annihilated, and $x_{sk}x_{ij} = x_{ik}x_{sj}$.

Since the symbolic stage does not have the values of nonzeros, all possible cancellation, as defined above, cannot be implemented. However, a special case, which includes the majority of cancellation that occurs in processing a matrix with dummy variables, can be implemented easily. Note that if the nonzeros are the same within rows, cancellation of type (b) of Definition 4.1.1 will occur. The cancellation occurring in Figure 4.1 is exactly of this type. To obtain (d) in Figure 4.1, identical rows are rotated first, giving

$$\begin{bmatrix} \sqrt{2} & \sqrt{2} & 0 & 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2} & 0 & \sqrt{2} & 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} * & * & * & * & * & * \\ & * & & & * & \\ & * & * & & * & \\ & * & * & & * & \\ & * & * & * & * & * \end{bmatrix}$$

(a) (b)

$$\begin{bmatrix} * & * & * & * & * & * \\ & & & & & \\ & * & * & & & \\ & * & * & & & \\ & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} * & * & * & * & * & * \\ & & & & & \\ & * & * & & & \\ & & & & & \\ & * & * & * & * & * \end{bmatrix}$$

(c) (d)

(a) Original matrix with dummy variables (15 nonzeros)
(b) Nonzero structure of (a) after processing column 1 without accounting for cancellation (19 nonzeros)
(c) same as (b) but accounting for cancellation (15 nonzeros)
(d) same as (c) but using a variable pivot row (13 nonzeros)

Figure 4.1 A matrix of dummy variables, and resulting nonzero structures after processing column 1

where again the nonzeros are identical within rows, and then rows 1 & 3, and 1 & 5 are rotated.

Certainly cancellation is an important factor here, but the use of a variable pivot row is also advantageous. The variable pivot above was used to take advantage of cancellation, when a set of rows is identical. Identical rows represent multiple observations per cell of a design. The following formalize the concepts illustrated in Figure 4.1.

Lemma 4.1.1 Suppose $x_{i1} = x_{is}$, and $x_{j1} = x_{js}$ for some s in (2.2.1). Let $\tilde{X}$ be the matrix X after the single Givens rotation of (2.2.1). Then

$\tilde{x}_{i1} = \tilde{x}_{is} = (x_{i1}^2 + x_{j1}^2)^{½}$, and $\tilde{x}_{j1} = \tilde{x}_{js} = 0$.

Proof: Apply (2.2). □

Proposition 4.1.1 Suppose $x_{i_j1} = x_{i_js}$, $j = 1, 2, ..., k$, and x_{i_11} is the pivot element for a sequence of $k - 1$ Givens rotations involving rows $i_1, i_2, ..., i_k$. Then, if $\tilde{X}$ is the matrix X after the rotations, $\tilde{x}_{i_11} = x_{i_1s} = \left[\sum_{m=1}^{k} x_{i_m1}\right]^{1/2}$, and $\tilde{x}_{i_j1} = \tilde{x}_{i_js} = 0$, $j = 2, 3, ..., k$.

Proof: Apply Lemma 4.1.1 $k - 1$ times. □

Corollary 4.1.1 Suppose k identical rows of dummy variables are processed with a sequence of $k - 1$ Givens rotations. The result is $k - 1$ rows of zeros, and one row with 1's replaced by $\sqrt{k}$.

Proof: This is a direct result of Proposition 4.4.1. □

Corollary 4.1.1 thus can be used to reduce a matrix with multiple observations per cell of a design to a matrix with a single observation per cell. The resulting matrix still has the property, that nonzeros are the same within rows. So additional cancellation, as described by Proposition 4.1.1, can occur during subsequent steps of the reduction. Since the results of Chapter 2 apply to single pivot row orderings, the application of Corollary 4.1.1, which uses a variable pivot, can be viewed as a pre-processing step.

Further cancellation, as described in Proposition 4.1.1 can be partially implemented by modifying Algorithm 2.5.2. Suppose the first j row nodes in θ_α^i are single root nodes in T^{i-1}, which means that these were not involved in any previous major steps, except possibly the pre-processing step. An equivalent condition is that these rows have no parent in T^{i-1}. Then, whenever there exists an $m \in Adj(\{\theta_{\alpha s}^i\}, B_0^{i-1})$ for $s = 1, 2, ..., u$, $(u \leq j)$, cancellation will occur in column m of the first u rows of θ_α^i. Note that with this cancellation, the relationship in Theorem 2.3.1 still holds, thus the results on row orderings in Section 2.3

are applicable. Algorithm 4.1.1 is a modification of Algorithm 2.5.2, which partially implements the cancellation discussed above.

Algorithm 4.1.1

```
1.  if $k_i = 1$ then set $P$ to 2 for each member of supernode $\{\theta_\alpha^i\}_1$
2.  else
3.    if $\{\theta_\alpha^i\}_1$ has no parent in $T^{i-1}$ then
4.      $\Omega \leftarrow Adj(\{\theta_\alpha^i\}_1, B_0^{i-1})$
5.      $Adj(\{\theta_\alpha^i\}_1, B_0^{i-1}) \leftarrow \varnothing$
6.    else $\Omega \leftarrow \varnothing$
7.    for $j = 2$ to $k_i$ do
8.      if $\Omega \neq \varnothing$ then
9.        if $\{\theta_\alpha^i\}_j$ has no parent in $T^{i-1}$ then
10.         $\Lambda \leftarrow Adj(\{\theta_\alpha^i\}_j, B_{j-2}^{i-1})$
11.       else $\Lambda \leftarrow \varnothing$
12.       $Adj(\{\theta_\alpha^i\}_{j-1}, B_{j-2}^{i-1}) \leftarrow Adj(\{\theta_\alpha^i\}_{j-1}, B_{j-2}^{i-1}) \bigcup \{\Omega - \Lambda\}$
13.       $\Omega \leftarrow \Omega \bigcap \Lambda$
14.     endif
15.     for each $r$ in supernode $\{\theta_\alpha^i\}_j$ do
16.       $Adj(r, B_{j-1}^{i-1}) \leftarrow Adj(r, B_{j-2}^{i-1}) - Adj(Fam(\{\theta_\alpha^i\}_{j-1}, T^{i-1}), B_{j-2}^{i-1})$
17.     endfor
```

```
18.  endfor
19.  link supernode ${\theta^i_\alpha}_1$ to supernode ${\theta^i_\alpha}_2$
20.  for each $r$ in supernode ${\theta^i_\alpha}_1$ do
21.      $\Phi \leftarrow Fam(\{\theta^i_\alpha\}_2, T^{i-1}) - Fam(\{\theta^i_\alpha\}_1, T^{i-1})$
22.      $Adj(r, B^i_0) \leftarrow Adj(r, B^{i-1}_{k_i-1}) - Adj(\Phi, B^{i-1}_{k_i-1}) - \{c_i\}$
23.  endfor
24.  $P_{\{\theta^i_\alpha\}_1} \leftarrow 1$
```

This algorithm carries the cancellation for $s = 1, 2, ..., u-1$, except when $u = k_i$ (k_i is the number of rows involved in major step i), then the cancellation is carried through for $s = u$. This is done, so that the fixed data structure of Section 2.5 can be retained. Note that in step 12 there is room in the data structure to accommodate the union, since we are simply putting back what was taken out.

Symbolic Givens reduction of a pre-processed matrix of dummy variables can thus be accomplished using Algorithm 2.5.3 with only step 20 replaced with Algorithm 4.1.1. Since not all cancellation is detected in this algorithm, it is possible that during the numerical reduction a null column is encountered. This is not a problem, since this event can be handled as any other rank deficiency by the Heath (1982) extension of the Givens algorithm.

Generally, the matrix of dummy variables is not of full rank. The Givens factorization of X, assuming that the first columns are linearly independent, takes the form (3.2). Only R is needed for computing a solution or a sum of squares for a given hypothesis, since

$$\begin{bmatrix} R^{-1}(R^{-1})' & O \\ O & O \end{bmatrix} \tag{4.1}$$

is a generalized inverse of $X'X$. Identification of a set of r linearly independent columns of X, or at least a set of linearly dependent columns, which can be removed without changing the rank of X, would be useful. One way to do this is to discard all columns associated with the first level of each main effect in both main effect and interaction columns. This gives a full rank subset of columns of X. The problem with this approach is that most of the columns discarded are associated with the interactions, which are the columns with greatest sparsity. The model statement contains some other information on linear dependencies among columns of X. For example, the columns associated with main effects A and B are linearly dependent on the columns associated with the interaction AB. The columns associated with A and B should be discarded, since columns associated with AB have fewer nonzeros. Also the sum of columns associated with any main effect or interaction is a column of ones.

Definition 4.1.2 An effect E_1 is contained in an effect E_2, if E_2 is an interaction containing E_1 or E_2 is nested within E_1.

In general thus, if an effect E_1 is contained in an effect E_2, then columns of E_1 are linear combinations of columns of E_2. This is true regardless of the imbalance in the data. Algorithm 4.1.2 uses these ideas to discard a set of relatively dense columns from X.

Algorithm 4.1.2

```
1. $S \leftarrow$ set of all effects in the model
2. while $S \neq \emptyset$ do
3.     $s \leftarrow$ an effect in $S$ with most levels
4.     $D \leftarrow \{ d \in S \mid d$ is contained in $s \}$
5.     $S \leftarrow S - \{ D \bigcup s \}$
```

```
6.    if $s$ is the first effect selected, then
7.        generate columns for all levels of $s$
8.    else generate columns for all except the most replicated level of $s$
9. endwhile
```

Let X_1 be the matrix of columns of X generated by Algorithm 4.1.2. Note that if an interaction containing all effects is present, the pre-processing step is all that is needed to produce R, which will be diagonal. Generally, however this is not the case, and Givens reduction must be applied to the pre-processed matrix to obtain R. Let X_{11} be the nonzero part of the pre-processed X_1. Thus so far,

$$X = \begin{bmatrix} X_1 & X_2 \end{bmatrix}, \quad \text{and}$$
$$X_1 = Q_1 \begin{bmatrix} X_{11} \\ O \end{bmatrix},$$

where Q_1 is a product of Givens rotation matrices for the pre-processing step, and rows of X_1 have been appropriately permuted. The modified Algorithm 2.5.2 can now be used to symbolicaly reduce X_{11} to upper trapezoidal form. Both row and column ordering strategies of Chapter 2 can be used. Since this is a very structured setting, it may be possible to determine optimal row and column orderings for certain classes of designs.

Some progress can be made by looking at the ordering problem analytically. Partition X_{11} as

$$X_{11} = \begin{bmatrix} E_1 & E_2 & \cdots & E_k \end{bmatrix},$$

where E_1 are the columns associated with an effect with the most levels, and E_2 to E_k correspond to the remaining effects. Note that no fill has been produced thus far, and the nonzeros

$$\begin{bmatrix}
* & & & & * & & \\
* & & & & & * & \\
* & & & & & & * \\
 & * & & & * & & \\
 & * & & & & * & \\
 & * & & & & & * \\
 & & * & & * & & \\
 & & * & & & * & \\
 & & * & & & & * \\
 & & & * & * & & \\
 & & & * & & * & \\
 & & & * & & & *
\end{bmatrix}$$

(a)

$$\begin{bmatrix}
* & & & & * & * & * \\
 & & & & * & * & \\
 & & & & * & * & * \\
 & * & & & * & * & * \\
 & & & & * & * & \\
 & & & & * & * & * \\
 & & * & & * & * & * \\
 & & & & * & * & \\
 & & & & * & * & * \\
 & & & * & * & * & * \\
 & & & & * & * & \\
 & & & & * & * & *
\end{bmatrix}$$

(b)

$$\begin{bmatrix}
* & * & * & * & * & & \\
* & * & * & * & & * & \\
* & * & * & * & & & * \\
* & * & & & & & \\
* & * & & & & & \\
* & * & & & & & \\
* & * & * & & & & \\
* & * & * & & & & \\
* & * & * & & & & \\
* & * & * & * & & & \\
* & * & * & * & & & \\
* & * & * & * & & &
\end{bmatrix}$$

(c)

(a) Original matrix with first four columns corresponding to E_1, and last three corresponding to E_2.
(b) The resulting matrix after processing E_1 (36 nonzeros).
(c) The resulting matrix after processing E_2 (42 nonzeros).

Figure 4.2 Processing one effect of a matrix containing columns of two effects

are identical within rows. If $k = 1$, then X_{11} is a diagonal matrix, and no further processing is needed. For $k = 2$, rows belonging to one level of one effect each have a nonzero in a unique column of the other effect. See Figure 4.2 (a) for an example, when E_1 has four levels and E_2 has three levels. While processing the columns of one effect, it is unavoidable to produce fill in columns of the other. It is however better to process the effect with more levels first. This is illustrated in Figure 4.2 (b) and (c). Note that all of the column ordering strategies discussed in Section 2.4 would make the same decision. If any rows were missing in Figure 4.2, the same conclusion would be reached. For $k > 2$, the situation becomes rapidly very complex, but it seems that a similar argument as above could be made for processing E_1 first.

Columns of an entire effect should be processed first, rather than mixing effects, since this processes rows in disjoint sets, and thus allows for taking advantage of cancellation. Recall that cancellation requires that a row was not previously processed. A further benefit of this is that the resulting portion of R has a simple form. Let Q_2 be the matrix of Givens rotations necessary to process E_1, then

$$X_{11} = Q_2 \begin{bmatrix} D & T \\ O & X_{12} \end{bmatrix},$$

where D corresponds to columns of E_1 and is diagonal. Normally, Givens reduction without square roots, due to Gentleman (1973), would be used. If the initial matrix X has integer nonzeros, which is the case with dummy variables, both D and T can be represented by integers. This can be seen by factoring out $\frac{1}{S}$ from (2.2.1) each time the transformation is applied, and then S^2 instead of S is stored at the end of a major step.

At this point it remains to factor X_{12}. It no longer has a simple structure as X_{11}, so Algorithm 2.5.3 should be used to find a good ordering. It is possible that some experimentation with this algorithm will lead to a good ordering obtainable from the structure of $E_1, \ldots, E_k$, or the model statement.

The numerical phase of the reduction begins with X_{11}, since X_{11} can be produced symbolically from X_1. It seems also possible that D, T, and X_{12} could be produced symbolically, as the diversity of the nonzero entries may still be manageable at this stage. Some additional research into this may prove fruitful.

4.2 Analysis of Variance and Estimable Functions

The main concern in regression on dummy variables is usually to test hypotheses about model parameters. Each hypothesis test has an underlying estimable function of the parameters. That is, if we wish to test if $H\beta = d$, where H is a $k \times p$ $(k < p)$ matrix of rank k, then $H\beta$ must be estimable.

Definition 4.2.1 $H\beta$ is estimable iff there is a matrix L such that $\mathrm{E}\{Ly\} = H\beta$.

Kennedy and Gentle (1980) discuss computational methods for testing such hypotheses. The sum of squares necessary for testing the above hypothesis is given by

$$(Hb - d)'[H(X'X)^{-}H']^{-1}(Hb - d), \tag{4.2}$$

where $(X'X)^{-}$ is any generalized inverse of $X'X$, and b is any solution to $X'Xb = X'y$. Thus to calculate the above sum of squares, two basic components are needed. First, a generalized inverse of $X'X$ must be obtained either explicitly or implicitly, which in turn gives also a solution b. Second, H must be obtained for some hypothesis of interest.

A generalized inverse of $X'X$ is given by (4.1). Note that this generalized inverse satisfies the first two Moore-Penrose conditions. That is, if $A = X'X$, and A^{*} is the matrix in (4.1), then $AA^{*}A = A$ and $A^{*}AA^{*} = A^{*}$. A generalized inverse, which satisfies these two conditions, will be denoted by superscript "$*$". The final column order of the reduced matrix R in (4.1) depends on the initial nonzero structure of X, since row and column permutations are performed to preserve sparsity, and to take advantage of cancellation. Thus the columns, from which the R factor is formed, are a subset of the columns selected by Algorithm 4.1.2, whose order depends on their nonzero structure.

Formulating a hypothesis of interest in an unbalanced or incomplete data is not an easy task, because a hypothesis is testable only if the underlying H is estimable. And conversely, not all estimable functions form "interesting" hypotheses. Since the methods of this chapter are aimed primarily at large models, it is important that H can be computer generated rather than required to be defined by the user. This raises the question of which hypotheses are appropriate in a wide range of model settings. There is general agreement on what hypotheses should be tested with balanced and complete data. The question of what should be tested in the unbalanced data case has recently received considerable attention in the literature. The emerging philosophy seems to be to test the hypotheses of the balanced case as much as the data allow. See for example Hocking, Speed, and Coleman (1980). The type III, and IV hypotheses of Goodnight (1978) are constructed according to this philosophy. With unbalanced data but no missing cells, the resulting type III, and IV hypotheses give the Yates' (1934) weighted squares of means technique, and are the same as the "usual" hypotheses for the same size model but with balanced data. When missing cells are present, often the "usual" hypotheses of the balanced case cannot be tested because of estimability problems. In this case, the type IV hypotheses are constructed to retain a property of the "usual" hypotheses, namely that the coefficients for any effect are distributed equitably across higher order effects which contain it. The exact procedure will be described below.

To construct any hypothesis, a generating set of estimable functions is needed. Since $\mathrm{E}\{y\} = X\beta$, we have $\mathrm{E}\{Ly\} = LX\beta$. So $H\beta$ is estimable if there is a matrix S such that $L = SX$. The rows of X thus form a generating set for all estimable functions. Another generating set is given by the rows of $X'X$, and also by the rows of $\begin{bmatrix} R & T \end{bmatrix}$. Any matrix with the same rowspace as X can be used as a generating set. Goodnight uses yet another generating set, the rows of $(X'X)^*X'X$, since this is available as a byproduct of the generalized sweep operator, which he uses to obtain a $(X'X)^-$ and a solution. A form of $(X'X)^*X'X$ can also be obtained from an orthogonal factorization. Suppose the first r columns of X are the columns associated with R, then

$$(X'X)^*X'X = \begin{bmatrix} R^{-1}(R^{-1})' & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X_1'X_1 & X_1'X_2 \\ X_2'X_1 & X_2'X_2 \end{bmatrix}$$
$$= \begin{bmatrix} I & R^{-1}(R^{-1})'X_1'X_2 \\ 0 & 0 \end{bmatrix}.$$

This can be computed from R by solving two triangular systems $R'B = X_1'X_2$, and $RA = B$, so that $A = R^{-1}(R^{-1})'X_1'X_2$. This generating set, however, need not be the same as the one obtained from the generalized sweep operator, since $(X'X)^*$ is not unique.

Which is the best generating set from the sparsity point of view? Consider the operations which need to be performed on the generating set to obtain type III, and IV estimable functions. Both type III, and IV estimable functions have the property, that those involving an effect E will also involve all effects which contain E, and will not involve any effects which do not contain E. This can be accomplished by "adjusting" each effect for all effects that do not contain it. For example, consider a three factor model with all interactions. The model statement is

$$y_{ijkl} \doteq \mu + a_i + b_j + c_k + ab_{ij} + ac_{ik} + bc_{jk} + abc_{ijk}. \qquad (4.3)$$

And the required adjustment is

$$\begin{array}{ll} a & \text{for } \mu,\ b,\ c,\ bc, \\ b & \text{for } \mu,\ a,\ c,\ ac, \\ c & \text{for } \mu,\ a,\ b,\ ab, \\ ab & \text{for } \mu,\ a,\ b,\ c,\ bc,\ ac, \end{array} \qquad (4.4)$$

ac for μ, a, b, c, ab, bc,

bc for μ, a, b, c, ab, ac,

and abc for μ, a, b, c, ab, ac, bc.

Goodnight chose $(X'X)^{*}X'X$ as the generating set, produced by the generalized sweep operator, since it has only r nonzero rows, and its elements are generally 0, 1, or -1. This matrix also is upper trapezoidal, with some of the required adjustment already done as a byproduct of the sweep operations. The $(X'X)^{*}X'X$ computed from R above, however does not have this nice form. The factor R together with T also forms a generating set, where some adjustment has already been done. Each effect has been adjusted for all effects whose columns precede it. However the order of the columns is determined by the sparsity pattern and not by the requirement above. In fact a highest order interaction is ordered first, and all effects, including the ones contained in it, are adjusted for it. Thus R together with T do not form a good generating set. Note that the sparsity preservation objective in computing a solution and the required adjustment above are in conflict. This is because the solution computations tend to order highest order interactions first, whereas the reverse is required to accomplish most of the above adjustment. For this reason, it seems wise to separate the two activities. The original matrix X is, of course, a generating set of estimable functions. The numerical computation of R will identify a set of r linearly independent rows of X, and X has a particularly nice form, since it contains only 0's and 1's. All of the required adjustment remains to be done, but at least a minimal set of rows has been identified.

Let Z_1 be a matrix of r linearly independent rows of X. Arrange the columns of Z_1 so that the mean goes first, then all main effects, then all 2-way interactions, then all 3-way interactions, etc. Using Gaussian elimination to put Z_1 into an upper trapezoidal form will accomplish most of the required adjustment. In fact if there are no missing cells, no further adjustment is needed. Let $Z_2 = G_1 Z_1$, where G_1 is the matrix representing the Gaussian elimination. The remainder of the adjustment can be performed by selective Gaussian elimination above the main diagonal. Let $Z_3 = G_2 Z_2$, where G_2 is the matrix representing the selective Gaussian elimination. The nature of this selective elimination is best illustrated by an exampl:. Figure 4.3 gives matrix Z_3, partitioned to show the eliminated parts, for the model in (4.3).

$$\begin{bmatrix} Z_\mu^\mu & Z_a^\mu & Z_b^\mu & Z_c^\mu & Z_{ab}^\mu & Z_{ac}^\mu & Z_{bc}^\mu & Z_{abc}^\mu \\ 0 & Z_a^a & 0 & 0 & Z_{ab}^a & Z_{ac}^a & 0 & Z_{abc}^a \\ 0 & 0 & Z_b^b & 0 & Z_{ab}^b & 0 & Z_{bc}^b & Z_{abc}^b \\ 0 & 0 & 0 & Z_c^c & 0 & Z_{ac}^c & Z_{bc}^c & Z_{abc}^c \\ 0 & 0 & 0 & 0 & Z_{ab}^{ab} & 0 & 0 & Z_{abc}^{ab} \\ 0 & 0 & 0 & 0 & 0 & Z_{ac}^{ac} & 0 & Z_{abc}^{ac} \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_{bc}^{bc} & Z_{abc}^{bc} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_{abc}^{abc} \end{bmatrix}$$

Figure 4.3 The structure of a generating set of estimable functions for model (4.3) after the adjustment (4.4). Estimable functions for effect e are given by the row containing superscript e, and the number of rows for an effect gives the degrees of freedom associated with that effect. The subscripts denote the effects involved in a given set of estimable functions

Trials with a few Z_1 matrices for various models and unbalance patterns show that Gaussian elimination as described above will rarely produce entries other than 0, 1, and -1. This suggests that a directed bipartite graph can be used to represent the portions (probably most and in many cases all) of the matrix, which have only 0, 1, and -1 entries.

Definition 4.2.2 A directed bipartite graph is a bipartite graph where each edge is an ordered pair.

Definition 4.2.3 The directed bipartite graph, representing an $n \times p$ matrix with entries 0, 1, and -1, is an ordered bipartite graph, where $(r_i,c_j) \in E$ iff $x_{ij} = 1$, and $(c_j,r_i) \in E$ iff $x_{ij} = -1$.

The computer representation of a directed bipartite graph can take the form of a row adjacency list, where the entries are positive or negative to indicate the direction. Figure 4.4 gives an

$$y_{ijkl} \doteq \mu + a_i + b_j + c_k + ab_{ij} + ac_{ik}$$

$$\begin{array}{l}
\left[\begin{array}{ccccccccccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}\right]
\end{array}
\qquad
\begin{array}{l}
1,2,4,6,8,12 \\
1,2,4,7,8,13 \\
1,2,5,6,9,12 \\
1,3,4,6,10,14 \\
1,3,4,7,10,15 \\
1,3,5,6,11,14
\end{array}$$

1,2,4,6,8,12	1,2,4,6,8,12	1,2,4,6,8,12
-6,7,-12,13	-6,7,-12,13	-6,7,-12,13
-4,5,-8,9	-4,5,-8,9	-4,5,-8,9
-2,3,-8,10,-12,14	-2,3,-8,10,-12,14	-2,3,-8,10,-12,14
-2,3,-6,7,-8,10,-12,15	-6,7,-14,15	-6,7,-14,15
-2,3,-4,5,-8,11,-12,14	-4,5,-10,11	8,-9,-10,11

$$\begin{array}{l|ccccccccccccccc|}
1,2,4,6,8,12 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
-6,7,-12,13 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\
-4,5,-8,9 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2,3,-8,10,-12,14 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
12,-13,-14,15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
8,-9,-10,11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1
\end{array}$$

Figure 4.4 An example of a model, an associated Z_1 and its row adjacency list representation, followed by four successive Gaussian elimination steps, and the resulting matrix with rows permuted in the order 1,4,3,2,6,5 to obtain the form of Figure 4.3

example of Gaussian elimination performed on an adjacency list of a Z_1 matrix. The final matrix is row permuted into the form of Figure 4.3. This is an example of a situation, where only 1, -1, and 0 occur. The following two strategies for selecting the pivot row should have the effect of completely avoiding, or at least minimizing, the number of nonzeros other than 1 and -1.

1. If a nonzero other than 1 or -1 occurs, try using a different pivot row.

2. Avoid using rows with nonzeros other than 1 or -1 as pivots.

When a nonzero other than 1 or -1 is unavoidable, a flag can be set for that row, or perhaps only that entry, and the nonzero stored in additional storage. The test cases considered so far have not required any additional storage with the above strategies. On the other hand no proof is available that only 1 and -1 nonzeros will occur. Another item, that remains to be resolved, is the actual data structure for this representation. The representation, as defined above, will require a dynamic data structure. Further research into this problem is required. In particular, some experimentation with larger computer generated examples may lead to a better data structure or representation.

The solution b obtained by the procedure of Section 4.1 has only r nonzero elements. It is exactly those r elements, which correspond to the columns of X that produced the columns of R. For simplicity of presentation, assume that this is the first r elements of b, and thus partition b into b_1 and 0. Partition H as $\begin{bmatrix} H_1 & H_2 \end{bmatrix}$ accordingly, so that (4.2) becomes

$$(H_1 b_1 - d)'[H_1 R^{-1}(H_1 R^{-1})']^{-1}(H_1 b_1 - d) \tag{4.5}$$

To obtain type III, or IV estimable functions for an effect, take the rows of the adjusted matrix Z_3 corresponding to that effect, and add appropriate multiples of rows corresponding to effects which contain the given effect. In fact it is only columns of the effects which were selected by Algorithm 4.1.2 that need to be involved. The columns forming H_1 are a subset of these columns. Algorithm 4.1.1 performs this task for effect e_0 and obtains a set of type III estimable functions. The notation used is that of Figure 4.3. The type III estimable functions have the property, that estimable functions for an effect e are orthogonal to estimable functions for any effect that contains e.

Algorithm 4.2.1

```
1. $S \leftarrow$ set of effects which contain $e_0$ and satisfy Algorithm 4.1.2, excluding $e_0$
2. for each $e \in S$ do
3.     $Z_e^{e_0} \leftarrow Z_e^{e_0}\left[ I - Z_e^{e\prime}(Z_e^e Z_e^{e\prime})^{-1} Z_e^e \right]$
4. endfor
5. $H_1 \leftarrow$ columns corresponding to $b_1$ from $Z_e^{e_0}$, $e \in S$
```

To obtain a type IV estimable function for effect e_0, Algorithm 4.1.1 can be used with a modification to step 3. This step should be replaced with $Z_e^{e_0} \leftarrow Z_e^{e_0} + KZ_e^e$, where K is a set of coefficients, one column for each row of $Z_e^{e_0}$. Each column of coefficients is determined from a given row of $Z_e^{e_0}$ as follows:

1. If any level of e_0 has a zero entry in the given row of $Z_e^{e_0}$, and that level of e_0 has a nonzero in Z_e^e, then set the coefficients corresponding to the nonzero rows of Z_e^e to zero.
2. Check to see if any coefficients corresponding to a level of e_0 is zero, when the level of e_0 is nonzero. If this is the case, the type IV estimable functions are not unique.
3. For each level of e_0, which has a nonzero entry in the given row, count the number of times that level occurs in effect e, then set each coefficient corresponding to that level to the nonzero entry divided by that count.

When no missing cells occur, and also for some missing cell patterns, the type IV estimable functions are the same as type III estimable functions.

Suppose H_1 has q rows, so that the rank of H_1 is $q < r$. Given that we have R, b_1, and H_1, the computation of (4.5) can be done by Algorithm 4.2.2, which follows.

Algorithm 4.2.2

```
1. $\tilde{d} \leftarrow H_1 b_1 - d$.
2. Solve $R'\tilde{H}_1 = H_1'$ .
3. Compute factorization $\tilde{H}_1 = U\begin{bmatrix} T \\ O \end{bmatrix}$.
4. Solve $T'v = \tilde{d}$.
5. $v'v$ gives the required sum of squares.
```

Note that the algorithm does not need any additional storage, since all computations can be done in place. The most storage is occupied by the matrix H_1, which is $q \times r$.

Further research is needed into the methods of this section. In particular, as noted earlier, the representation of Gaussian elimination to form Z_3 can likely be improved. Another question is whether it is worthwhile to use sparse matrix methods in Algorithm 4.2.2. The answer will depend on the size and sparsity of matrices involved. Some experimentation must be done with larger computer generated models, since the level of complexity obtained from small hand computed examples is not sufficient to answer these questions.

5. COMPUTER IMPLEMENTATION AND TESTING

The preceding three chapters include computer algorithms, which need implementation and testing on the computer. Chapter 2 discusses a bipartite graph model for performing symbolic Givens factorization of a sparse matrix. The FORTRAN program implementing this and several row and column ordering strategies are discussed in Section 5.1. This program is then used to compare the ordering strategies on a number of test problems in Section 5.2. Section 5.3 then discusses the program implementing the updating procedure of Chapter 3. The sparse matrix methods for analysis of variance, discussed in Chapter 4, still have a number of problems, which need to be researched. For this reason, no implementation is given here.

5.1 Symbolic Givens Reduction

Symbolic reduction only manipulates row and column indices, so integer arithmetic is used throughout. For this reason, concerns about precision do not arise. Algorithm 2.5.3 and its component Algorithms 2.5.1 and 2.5.2 perform the symbolic reduction. All of these were programmed in FORTRAN IV, and tested both on a FORTRAN H compiler and a VAX/UNIX FORTRAN compiler. Also included in this were row and column ordering strategies. The source code, including numerous comments, is in Appendix A. The subprograms are listed in alphabetical order, and the main routine is listed first.

The column ordering strategies programmed are

- natural ordering (no ordering),
- minimum column count, first tied
- minimum column count, last tied,
- minimum degree, first tied,

- minimum degree, last tied, and
- minimum degree with column count tiebreaking.

Algorithm 2.5.5 forms an integral part of strategies 2, 3, and 6, as it updates the column counts (number of nonzeros in a column). Algorithm 2.5.4 forms an integral part of strategies 4, 5, and 6, as it updates the degree of each column.

The row ordering strategies programmed are

- natural order (no ordering),
- minimum row count, and
- minimum pivotal row fill.

Here, Algorithm 2.5.6 is used to update the set of rows competing for the next position within a given major step. Note that strategy 1 must perform some limited ordering, since the next row may not contain a nonzero in the current pivot column. In a case when the next row does not have a nonzero in the current pivot column, the first possible subsequent row is taken. The comments in the source code should be sufficient to explain the programming details of these strategies.

Figure 5.1 contains the call tree structure of the program, and some correspondences to the algorithms of Section 2.5. Calls to some utility routines have been left out from the tree for simplification. Note also that the correspondence to the algorithms of Section 2.5 is not exact, however the essence is the same. Some loops have been combined to improve efficiency.

Although a great deal has been done to make the program more efficient compared to its initial version, there is still much room to improve its efficiency. For example, special handling of situations, such as when only a single row is competing for next position, should still achieve large gains in speed, particularly in the later stages of a factorization. There are other possibilities,

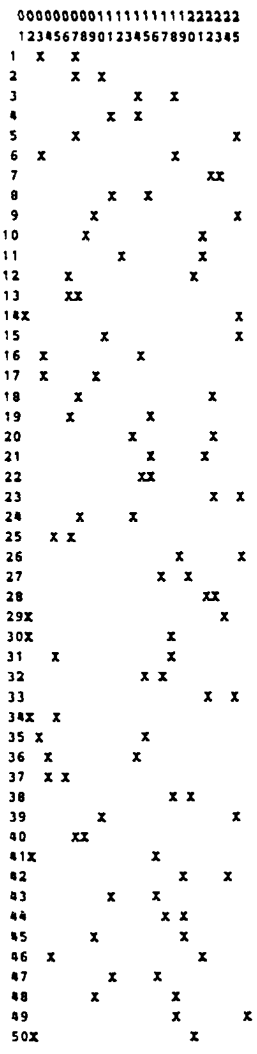
```
MAIN  GETMAT                        generates or reads a matrix
      SETUP  CNZUD
             DEGUD                  initialization
             INIT
      REDUCE MINDEG
             MINDG1
             MINDG2                 column selection strategies
             MRJCJF
             MRJCJL
             MAJOR  RFIND  CFIND
                           ANCSTR
                           FAM      Algorithm 2.5.3, steps 7-17
                           MARKIT
                           GETUNL
                    GETNOD MININD
                           MINRF    row selection strategies
                           MPFILL
                           ANCSTR
                           CKCHLD   Algorithm 2.5.6
                           REPACK
                    CUTREE          Algorithm 2.5.1, step 2
                    ADJUST
                    FAM             Algorithm 2.5.2, steps 4-6,9-11
                    EXTRA
                    SETREE          Algorithm 2.5.1, step 3
             JOIN                   Algorithm 2.5.2, step 12
             DISCON                 Algorithm 2.5.2, step 1
             DEGUD  NEWDEG CFIND
                           FROOT
                           FAM      Algorithm 2.5.4
                           MARKIT
                           EXTRA
             CNZUD  COLNZ  CFIND
                           ANCSTR
                           GETUNL   Algorithm 2.5.5
                           FAM
                           MARKIT
             RECORD                 records structure of new row of  R
             PRTVEC                 printing routine for debugging
             PRTX                   prints partially factored matrix
```

Figure 5.1 The call tree of the symbolic Givens factorization program, and some correspondences to algorithms of Section 2.5. Note that some calls to routines INIT, ADD, ANCSTR, and MARKIT are omitted

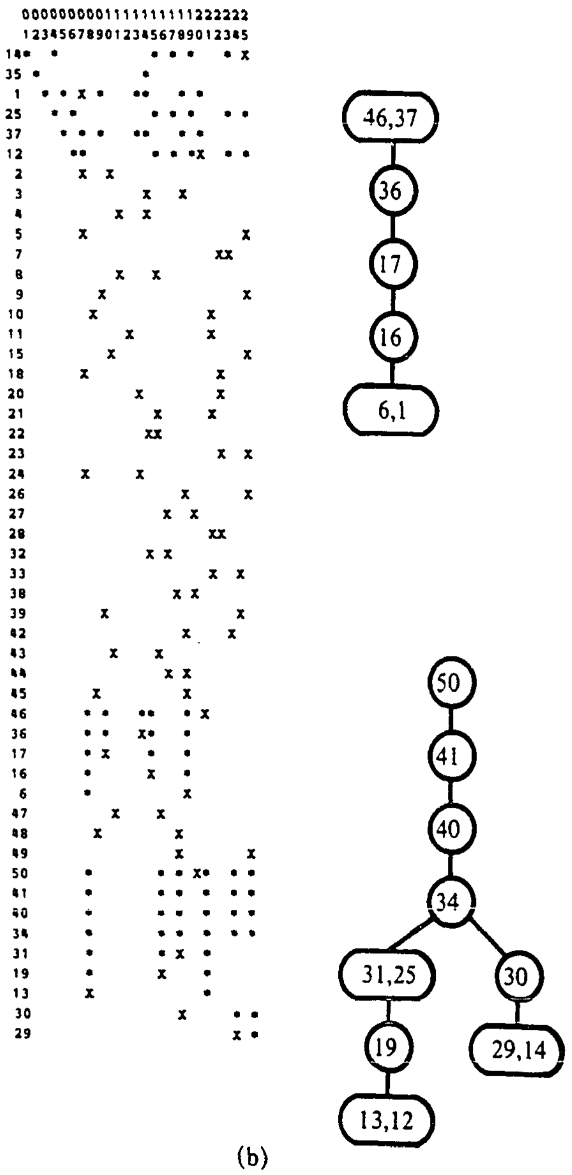
where special handling of a frequently occurring situation will improve efficiency. These will become apparent as more experience is gained with this program.

No attempt has been made to compare the speed of this algorithm to the Cholesky symbolic factorization. The computer package SPARSPAK, developed at the University of Waterloo by George, Liu, and Ng (1980), contains a very efficient version of Cholesky symbolic factorization. Givens symbolic factorization, as it is now implemented, is expected to be slower than Cholesky symbolic factorization except in some special cases. The reasons for this are several. In many sparse problems, X contains more nonzeros than half of $X'X$; each major step, which contains several minor steps, of Givens factorization is equivalent to one step of Cholesky factorization; and, as was pointed out above, the Givens symbolic algorithm is still not "mature", and will undergo some improvements. The special cases, where it may be faster, are when X contains considerably fewer nonzeros than half of $X'X$. This situation can be created, when X consists of sets of identical rows. Each set of identical rows can be reduced to a single row before processing. Matrices of this type are discussed in Chapter 4, and test problem 3 of the next section is of this type.

For certain classes of problems, the increase in symbolic factorization time should be more than offset by better orderings for the numerical stage. Particularly large gains should be made, where several least squares problems with the same nonzero structure need to be solved. Such situations occur, for example, where a nonlinear least squares problem is solved by several iterations of linear least squares problems with identical nonzero structures. Better column orderings are possible; since some heuristic column ordering algorithms, not available with the Cholesky symbolic algorithm, become available. Also, heuristic row ordering algorithms can now be applied. Although there is the added cost of sorting the rows into the necessary order, but the bulk of this cost would be input/output, since the order is known from the symbolic step, and no comparisons



(a)



(b)

Figure 5.2a A matrix with two random entries per row

Figure 5.2b The matrix of Figure 5.2a after six major steps and the associated T^6. Only nontrivial trees of T^6 are displayed

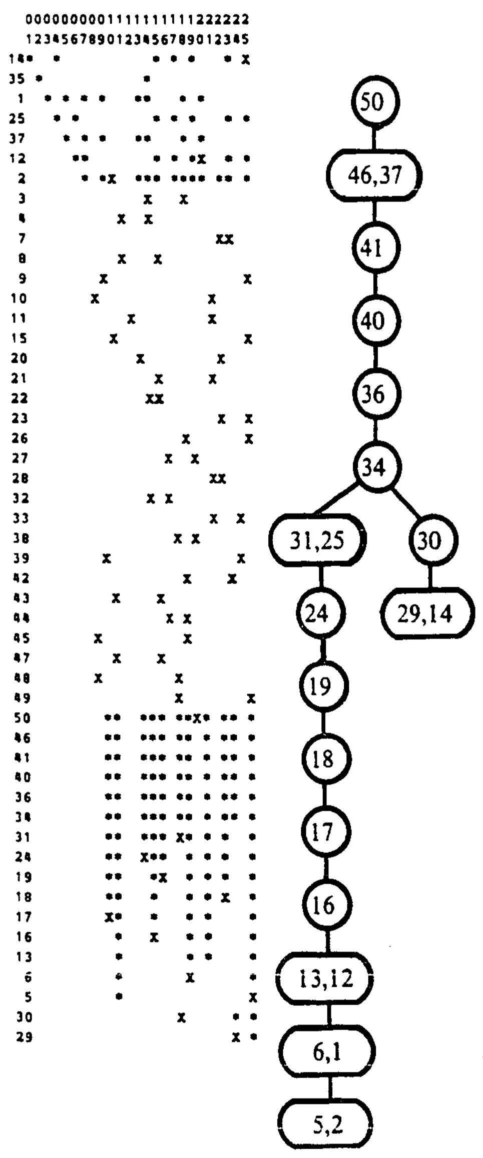

(c)

Figure 5.2c The matrix of Figure 5.2 a after seven major steps, and the associated T^7

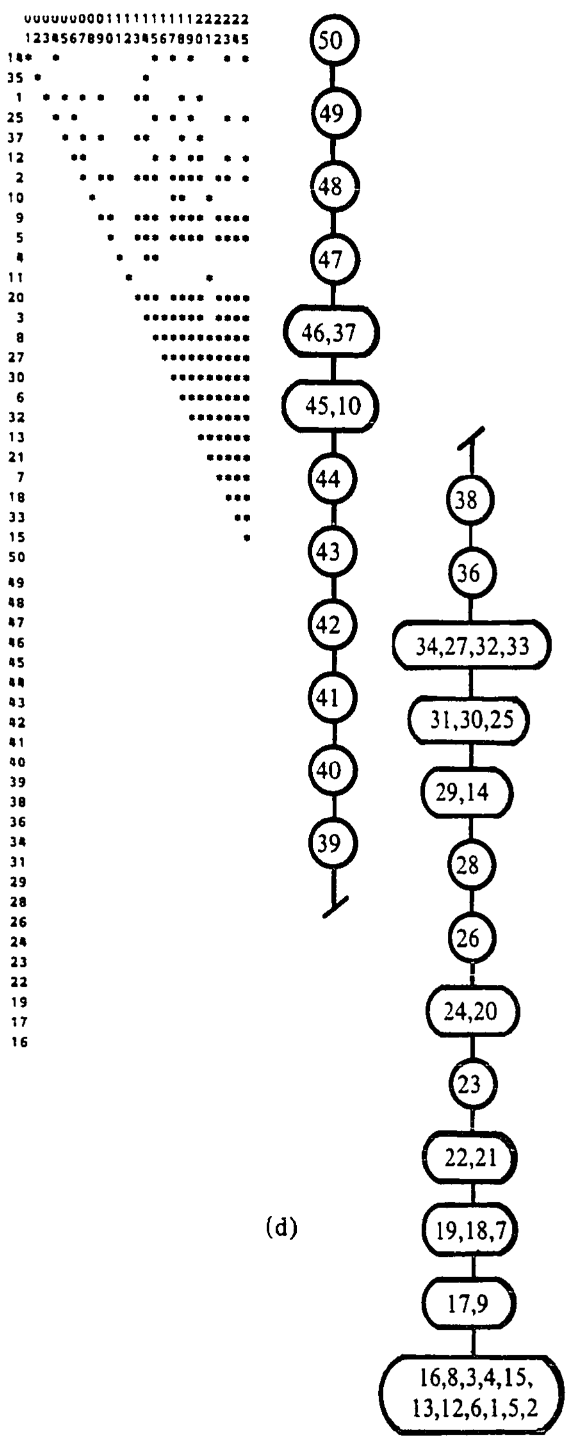

(d)

Figure 5.2d The final R factor and $T^{25}_?$ after completing the reduction of the matrix of Figure 5.2a

need to be done. In the case where the entire problem is in core, there is no sorting cost, since the ordering is done by indexing. Another feature, not yet implemented, is the selection of rows to be left out from the initial factorization, and then used to update only the solution. This can also potentially speed up the numerical factorization and reduce storage requirements for certain classes of problems.

For purposes of debugging, a capability of printing the nonzero structure of a partially reduced matrix was programmed. The output of this capability has proved to be very useful in illustrating the row structure described by Theorem 2.3.1. The rows of the unreduced portion of a matrix are ordered by trees of the forest T^i, and in preorder within trees. It is also used to illustrate which elements of the matrix are represented by edges in $\mathbf{B}_0^i$.

Figure 5.2 gives a random 50×25 matrix with two nonzeros per row, two partially reduced matrices after 6, and 7 major steps, and the final R factor. Only nontrivial trees of the forests T^6, T^7, and T^{25} are included with each matrix. The representative of each supernode in a tree is listed first. Both "X" and "*" represent nonzeros, but only the "X"s are stored in the data structure, and the "*"s are generated from the "X"s by the forest structure T^i. That is, each "X" corresponds to an edge in $\mathbf{B}_0^i$, but the edges in B_0^i correspond to both "X"s and "*"s. Note that each row still has at most two "X"s, and these are in the positions of nonzeros of the original matrix. As the factorization proceeds, there are fewer "X"s. Also note that major step 7 involved rows 2, 5, 18, 24, 46, 36, 17, 16, 6, 50, 41, 40, 34, 31, 19, and 13, which form a path from the pivot row 2 to the root of the tree in T^7. These rows form a structure in Figure 5.2 (c), as described by Theorem 2.3.1.

5.2 Comparison of Ordering Strategies

The previous section lists the six column ordering and three row ordering strategies programmed. All 18 combinations of these strategies were compared on a few test problems. There are five test problems, one of which is real, and the other four are artificially generated.

Two of the generated problems are matrices with three random entries per row. Random matrices are generally considered the most difficult, since there are no patterns to exploit. One problem is an artificially generated least squares problem on an 8×8 square grid. Such problems arise in the natural factor formulation of finite element methods. The last artificially generated problem involves a network, like those arising in geodetic adjustment applications. The real problem is from a survey conducted in Sudan. The programs to generate the artificial test problems, as well as the real problem were kindly provided by M. T. Heath at Oak Ridge National Laboratory. For a more detailed description of the generated problems see Heath (1983). Table 5.1 lists the test problems and their characteristics.

Table 5.1 Characteristics of the Test Problems

problem number	rows	columns	nonzeros	remarks
1	100	50	300	random
2	100	75	300	random
3	196	64	588	8×8 grid problem
4	306	160	1448	4×2 network, $l = 2$, $\mu = 1$
5	313	176	1557	Sudan survey data

The row and column ordering strategies are compared on the basis of three criteria. The first criterion is the number of nonzeros in the final R factor, which is directly related to the storage requirement for the numerical phase. The second and third criteria are the number of Givens rotations and the number of operations required for the factorization respectively. One operation is defined as the processing of one column of the two rows involved in a minor step. Thus, the number of operations in a minor step is given by the number of nonzeros in the pivot row after completion of the minor step. The time required for a numerical factorization should be approximately a linear function of these two counts. Tables 5.2 through 5.6 give these counts for test problems 1 through 5 respectively.

Before commenting on overall performance of the orderings, note the counts for natural column order in Table 5.4. In terms of operation counts the natural row ordering is the best. The nonzeros in this problem are in a band from upper left to lower right of the X matrix. A

Table 5.2 Problem 1

column ordering	row ordering		
	natural order	minimum row count	min. pivot row fill
natural order	929[a] 1769[b] 26297[c]	929 1648 22741	929 1684 23164
min. column count (first tied)	657 1268 14459	714 1157 12349	722 1112 11368
min. column count (last tied)	670 1282 14971	665 1152 12192	666 1104 11311
minimum degree (first tied)	642 1480 17792	642 1486 17818	642 1474 17396
minimum degree (last tied)	631 1553 19598	631 1495 18181	631 1382 15356
minimum degree (col. count tiebr.)	629 1238 13878	629 1221 13156	629 1125 11290

[a] Nonzeros in R factor.

[b] Number of Givens rotations.

[c] Number of operations.

Table 5.3 Problem 2

column ordering	row ordering		
	natural order	minimum row count	min. pivot row fill
natural order	1771 1880 41834	1771 1718 35335	1771 1682 33628
min. column count (first tied)	1002 953 13092	1036 836 10262	1076 827 9963
min. column count (last tied)	1005 967 13017	1040 838 10304	1040 799 9394
minimum degree (first tied)	944 1177 17826	944 1141 16676	944 1103 15476
minimum degree (last tied)	935 1167 17306	935 1083 15302	935 1076 14813
minimum degree (col. count tiebr.)	934 1020 13965	935 916 11859	934 853 10234

closer inspection of the symbolic reduction process with the natural ordering revealed, that the natural row ordering is not locally acceptable. Since the two locally acceptable row orderings performed worse, this seemed like a good test for Corollary 2.3.1. It was found, that when the minimum pivot row fill ordering is modified so that a new row (not previously processed) is never taken as a pivot (except in the first major step, of course), then a row ordering which is at least as good as the natural row ordering is produced. This new ordering gives exactly the same operation and rotation counts, and is locally acceptable, thus illustrating Corollary 2.3.1. The superiority of the natural row ordering is due to the fact, that when the R factor is formed in this order, some

Table 5.4 Problem 3

column ordering	row ordering		
	natural order	minimum row count	min. pivot row fill
natural order	568 1682 8956	568 4454 36161	568 3437 26321
min. column count (first tied)	672 2286 20547	698 2103 15205	687 1837 13295
min. column count (last tied)	616 2176 18175	639 2061 15315	694 1994 15203
minimum degree (first tied)	492 1749 10221	492 2017 12812	492 1766 10500
minimum degree (last tied)	492 1666 9687	492 1925 11868	492 1741 10021
minimum degree (col. count tiebr.)	503 1938 13004	485 1940 12134	503 1847 11782

rows during processing are structurally dependent on it, and are eliminated before completion of the reduction. Their absence during the remainder of the reduction greatly reduces the operations count. From the point of view of processing by columns, this can be viewed as symbolic cancellation. This emphasizes that the ordering strategies are only heuristics, and need not produce orderings close to the optimum.

The following are some observations from Tables 5.2 through 5.6:

- Based on the first criterion, the three minimum degree column ordering variations are better than the other strategies. So if storage is of primary concern, minimum degree column ordering should be used.

Table 5.5 Problem 4

column ordering	row ordering		
	natural order	minimum row count	min. pivot row fill
natural order	2648 3836 41082	2648 6392 94882	2648 5334 66147
min. column count (first tied)	1724 2700 20668	1680 3820 35794	1844 3280 27168
min. column count (last tied)	1636 2634 18931	1660 3426 29769	1664 3006 22935
minimum degree (first tied)	1600 3196 24320	1600 4016 34290	1600 4356 37188
minimum degree (last tied)	1568 2694 18493	1568 3520 28012	1568 3048 22392
minimum degree (col. count tiebr.)	1584 2604 18212	1588 3682 30925	1584 2880 20656

- The handling of ties in column ordering strategies has little effect on the first criterion.
- The handling of ties in column ordering strategies can have a large effect on operation and rotation counts.
- With a few exceptions, the minimum pivotal row fill row ordering strategy performs better than the minimum row count row ordering strategy.
- Less structured problems (1, 2, and 5) benefit more from a row ordering strategy than do more structured problems.

Table 5.6 Problem 5

column ordering	row ordering		
	natural order	minimum row count	min. pivot row fill
natural order	6794 8355 212687	6794 16533 482035	6794 13941 392239
min. column count (first tied)	2504 3680 41655	2660 3370 34758	2676 3013 29423
min. column count (last tied)	2547 3720 40845	3095 3596 39978	2899 3268 32799
minimum degree (first tied)	1591 3890 29830	1591 4694 37521	1591 2693 17498
minimum degree (last tied)	1631 4058 32405	1631 4575 37419	1631 2883 19824
minimum degree (col. count tiebr.)	1630 3682 27868	1630 4138 32312	1630 2596 17112

- The minimum column count strategy performs well on the random matrices in terms of rotation and operation counts at the expense of a few extra nonzeros in R. When column count is used as tiebreaking for minimum degree on the random matrices, rotation and operation counts are reduced without adding nonzeros to R.

Only the first two observations are not problem dependent, although it may be that other types of test problems could lead to different conclusions. Perhaps one of the most useful applications of the symbolic algorithm would be to consider a much larger and broader set of test problems, and determine which strategies work best on which classes of problems. The five test problems considered here are a small step in that direction.

5.3 Implementation of an Updating Algorithm

The updating algorithm of Section 3.1 performs matrix operations, where only matrices R and T are stored in sparse storage mode, and all other matrices are in full storage mode. For the purpose of testing, the algorithm was programmed in the APL programming language. This programming language naturally lends itself to matrix operations, and facilitates easy implementation of the algorithm. Since the test problems used were not very large, both R and T were also used in full storage mode, thus eliminating special handling required for sparse storage mode. Of course, this APL program is only for testing purposes, and in time a FORTRAN version should be programmed.

Algorithm 3.1.1 is presented with the assumption, that every time a factorization is done, the leading rows of the matrix are linearly independent. This eliminates the need to clutter the presentation with permutation matrices, thus giving a clearer picture of of the basic algorithm. However, these permutation matrices must be included in the computer implementation, since this assumption generally does not hold. Producing the correct permutations is not a trivial matter, but APL provides an easy facility for performing these.

Because the APL programs are quite concise, they are included and documented in this section. All factorizations are performed using Householder orthogonal transformations with pivoting for stability. The APL function HHT performs this factorization, and returns the factors as well as the permutations used to obtain them. This function is used by the function UPDATE which computes the initial solution to a least squares problem, updates it by additional rows, and then computes also a complete solution directly for comparison. Both APL functions follow.

```
    ∇ VR←HHT VX;K;T;P;J;I;Q1;M;N;S;V;Q1;ROW;RK
[1]   ⍝ ORTHOGONAL FACTORIZATION BY HOUSEHOLDER TRANSFORMATIONS
           WITH PIVOTING FOR STABILITY
[2]   ⍝ INPUT MATRIX: VX
[3]   ⍝ OUTPUT: TRIANGULAR FACTOR - VR
[4]   ⍝         RANK DEFFICIENCY FACTOR - VT
[5]   ⍝         COLUMN PERMUTATION FOR VR - COLS
[6]   ⍝         COLUMN PERMUTATION FOR VT - CC
[7]   ⍝         ROW PERMUTATION - ROWS
[8]    VR←VX
[9]    ROWS←0⍴0
[10]   N←1↑⍴VR
[11]   ROW←N⍴J←1
[12]   VQ←I←(N,N)⍴1,N⍴0
[13]   V←(⍴VR)⍴COLS←(P←(-1)↑⍴VR)⍴0
[14]  L1:→L2×⍳TOL≥M←⌈/T←ROW+.×VR×VR
[15]   S←(T+.×T←ROW×VR[;COLS[J]←T⍳M])*0.5
[16]   RK←K←(|T)⍳M←⌈/|T
[17]   ROW[RK]←0
[18]   ROWS←ROWS,RK
[19]   V[ROW/⍳N;J]←(ROW/T)÷(2-4×T[K]<0)×S×V[K;J]←(0.5×1+M÷S)*0.5
[20]   Q1←I-V[;J]∘.×2×V[;J]
[21]   VQ←(N,N)⍴Q1+.×VQ
[22]   VR←(N,P)⍴Q1+.×VR
[23]   →L1×⍳P≥J←J+1
[24]  L2:CC←⍳P
[25]   COLS←(COLS>0)/COLS
[26]   CC[COLS]←(⍴COLS)⍴0
[27]   CC←(CC>0)/CC
[28]   VT←VR[ROWS;CC]
[29]   VR←VR[ROWS;COLS]
    ∇
```

```
    ∇ UPDATE;UR;K;UU;US;M;T;CC;Q;H;W;G;F;D2;D21;D1;U;S;R2;B1;VT;V
          Q;COLS;ROWS;COL1;CC1;CC2;L
[1]   ⍝ INPUT: MATRICES X AND E, AND VECTORS Y AND Z
[2]   ⍝ OUTPUT: SOLUTION VECTOR B OBTAINED THROUGH UPDATE
[3]   ⍝         SOLUTION VECTOR BC OBTAINED DIRECTLY
[4]   ⍝ NUMBERS ON RIGHT REFER TO STEPS OF ALGORITHM 3.1.1
[5]    P←(-1)↑⍴X
[6]    R←1↑⍴UR←HHT X                                          ⍝ 1
[7]    T←VT                                                   ⍝ 1
[8]    B←P⍴0                                                  ⍝ 2
[9]    B[COL1]←B1←(VQ[ROWS;]+.×Y)⌹UR                          ⍝ 2
[10]   R2←Z-E[;COL1←COLS]+.×B1                                ⍝ 3
[11]   Q←(-1)↑⍴K←(⍉E[;COL1])⌹(⍉UR)                            ⍝ 4
[12]   H←E[;CC1←CC]-(⍉K)+.×T                                  ⍝ 5
[13]   L←⍉HHT K,[1](Q,Q)⍴1,Q⍴0                                ⍝ 6
[14]   UU←⍉VQ[ROWS;⍳R]                                        ⍝ 6
```

```
[15]   US←H[CC2←COLS;]⌹L                              ⍝ 7
[16]   W←R2[CC2]⌹L                                    ⍝ 7
[17]   US←HHT US                                      ⍝ 8
[18]   G←(⍴ROWS)⍴VQ[ROWS;]+.×W                        ⍝ 8
[19]   D21←G⌹US                                       ⍝ 9
[20]   D2←(P-R)⍴0
[21]   D2[COLS]←D21
[22]   S←(R2[CC2]-H[CC2;]+.×D2)⌹L                     ⍝ 10
[23]   U←UU+.×S                                       ⍝ 11
[24]   F←U⌹UR                                         ⍝ 12
[25]   M←T⌹UR                                         ⍝ 13
[26]   D1←F-M+.×D2                                    ⍝ 14
[27]   B[COL1]←B[COL1]+D1                             ⍝ 15
[28]   B[CC1]←D2                                      ⍝ 15
[29]   BC←P⍴0                                         ⍝ DIRECT
[30]   BC[COLS]←(VQ[ROWS;]+.×Y,Z)⌹HHT X,[1]E          ⍝ SOLUTION
     ∇
```

The program was run under UNIX APL\11 on the VAX 11/780 computer. This version of APL performs all calculations in double precision. Several test problems were artificially generated, and solved by the program. The solution by updating and a direct solution had a maximum relative difference of 10^{-15}. Although these generated problems were small, and probably well conditioned, this shows that the algorithm has promise. More thorough testing should be done with known ill-conditioned problems, using a FORTRAN version of the algorithm.

Algorithm 3.2.1 was not implemented at this time. Its behavior is expected to be similar to Algorithm 3.1.1. It is in fact this more general algorithm, which should be programmed in FORTRAN, and subjected to thorough testing.

6. BIBLIOGRAPHY

Björck, Åke. 1976. Methods for sparse linear least squares problems. Pp.177-199 in Sparse Matrix Computations. J. R. Bunch and D. J. Rose, eds. Academic Press, New York.

Björck, Åke and Duff, Iain S. 1980. A direct method for the solution of sparse linear least squares problems. Linear Algebra and Its Applications 34:43-67.

Duff, Iain S. 1974. Pivot selection and row ordering in Givens reduction on sparse matrices. Computing 13:239-248.

Duff, Iain S. 1983. Direct methods for solving sparse systems of linear equations. AERE Report CSS 131. (Computer Science and Systems Division, A.E.R.E., Harwell, England)

Duff, Iain S. and Reid, J. K. 1976. A comparison of some methods for the solution of sparse over-determined systems of linear equations. Journal of the Institute of Mathematics and its Applications 17:267-280.

Eisenstat, S. C. 1983. Iterative methods for solving large sparse linear systems. Invited paper at Sparse Matrix Symposium 1982, Fairfield Glade, Tennessee, October 24-27, 1982.

Gentleman, W. Morven. 1973. Least squares computations by Givens transformations without square roots. Journal of the Institute of Mathematics and Its Applications 12:329-336.

George, J. Alan and Heath, Michael T. 1980. Solution of sparse least squares problems using Givens rotations. Linear Algebra and Its Applications 34:69-83.

George, J. Alan and Liu, Joseph W. H. 1981. Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs, New Jersey. 324pp.

George, J. Alan, Heath, Michael T. and Ng, Esmond. 1983. A Comparison of some methods for solving sparse linear least-squares problems. SIAM Journal of Scientific and Statistical Computing 4(2):177-187.

George, J. Alan, Heath, Michael T. and Plemmons, Robert J. 1981. Solution of large-scale least squares problems using auxilliary storage. SIAM Journal of Scientific and Statistical Computing 2(4):416-429.

George, J. Alan, Liu, Joseph W. H., and Ng, Esmond. 1980. User's Guide for SPARSPAK: Waterloo sparse linear equations package. Research Report CS-78-30 (revised). Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Goodnight, James H. 1978. Tests of hypothesis in fixed effects linear models. SAS Technical Report R-101. SAS Institute Inc., Cary, North Carolina.

Heath, Michael T. 1982. Some extentions of an algorithm for sparse linear least squares problems. SIAM Journal of Scientific and Statistical Computing 3(2):223-237.

Heath, Michael T. 1983. Numerical methods for large sparse linear least squares problems. Technical Report ORNL/CSD-114. Union Carbide Corp., Oak Ridge, Tennessee.

Hocking, R. R. and Speed, F. M. 1975. A full rank analysis of some linear model problems. Journal of the American Statistical Association 70(351):706-712.

Hocking, R. R., Speed, F. M. and Coleman, A. T. 1980. Hypotheses to be tested with unbalanced data. Communications in Statistics - Theory and Methods A9(2):117-129.

Kennedy, William J., Jr. and Gentle James E. 1980. Statistical Computing. Marcel Dekker, New York. 591pp.

Knuth, D. E. 1968. The Art of Computer Programming, Vol. 1: Algorithms and Data Structures. Addison-Wesley, Reading, Mass.

Kolata, G. B. 1978. Geodesy: Dealing with an enormous computer task. Science 200:421-422.

Lawson, C. L. and Hanson, R. J. 1974. Solving Least Squares Problems. Prentice-Hall, Englewood Cliffs, New Jersey.

Markowitz, Harry M. 1957. The elimination form of the inverse and its application to linear programming. Management Science 3:255-269.

Parter, S. V. 1961. The use of linear graphs in Gauss elimination. SIAM Review 3:119-130.

Peters, G. and Wilkinson, J. H. 1970. The least squares problem and pseudo-inverses. The Computer Journal 13(3):309-316.

Rose, Donald J. 1972. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. Pp.183-217 in Graph Theory and Computing. R. C. Read, ed. Academic Press, New York.

Tewarson, Reginald P. 1973. Sparse Matrices. Academic Press, New York. 160pp.

Tinney, W. F. 1969. Comments on using sparsity techniques for power system problems. Sparse Matrix Proceedings, IBM Research Report, RAI 3-12-69.

Yates, F. 1934. The analysis of multiple classifications with unequal numbers in different classes. Journal of the American Statistical Assocation 29:51-66.

7. APPENDIX A

7.1 The Symbolic Givens Reduction FORTRAN Program

```
      INTEGER RADJ(1000),ADJNCY(3000),PARENT(1000),CHILD(1000),
     +        SIBLNG(1000),NONZER(1000),RFAC(300),FACADJ(6000),
     +        WRKROW(300),WORKR1(1000),RSTAC(1000),WRK2(300),
     +        DEGREE(300),CNONZ(300),CORDER(300),CLIST(300),
     +        RORDER(300),RSTAT(1000),SUPERN(1000),WORKR2(1000),
     +        RROW(300),WORKZ1(3000),WORKZ2(3000)
      INTEGER PFAC,TITLE(20),CSTRAT,RSTRAT
      INTEGER NROW,NCOL,NNZER,NMOD,NROWP1,NRZER,IPRINT,NOPER,MINOR
C
C     (RADJ,ADJNCY) - INPUT ROW ADJACENCY STRUCTURE OF THE MATRIX X
C     (PARENT,CHILD,SIBLNG) - FOREST STRUCTURE  T
C     NONZER - NUMBER OF NONZEROS IN EACH ROW
C     (RCFAC,FACADJ) - OUTPUT ROW ADJACENCY STRUCTURE OF R FACTOR
C     DEGREE - DEGREE OF EACH COLUMN
C     CNONZ - NUMBER OF NONZEROS IN EACH COLUMN
C     CORDER,CLIST,RORDER - KEEP TRACK OF COLUMN ORDER
C     SUPERN - VECTOR TO LINK NODES IN SUPERNODES
C     RSTAT - ROW STATUS  0 - IN UNREDUCED PORTION
C                         1 - IN R, BUT CONTRIBUTING TO UNRED. PORTION
C                         2 - IN R, AND NOT CONTRIB. TO UNRED. PORTION
C     RROW - KEPPS TRACK OF PIVOT ROWS
C     WORKZ1,WORKZ2 - WORK VECTORS FOR READING A MATRIX IN I-J FORMAT
C     WRKROW,WRK2 - WORK VECTORS FOR CURRENT ROW STRUCTURE
C     WORKR1,WORKR2 - WORK VECTORS FOR ROW INDICES
C     NROW - NUMBER OF ROWS IN X
C     NCOL - NUMBER OF COLUMNS IN X
C     NNZER - NUMBER OF NONZEROS IN X
C     IPRINT - REGULATES AMOUNT OF OUTPUT -3 GIVES MINIMAL OUTPUT
C                                          2 GIVES MAXIMAL OUTPUT
C              (SEE SUBROUTINE REDUCE FOR MEANING OF INDIVIDUAL VALUES)
C     NMOD - USED IN CONJUNCTION WITH IPRINT>-1, OUTPUTS NONZERO STRUCT
C            EVERY NMOD MAJOR STEPS
C     NOPER - COUNTS OPERATIONS
C     MINOR - COUNTS GIVENS ROTATIONS
C     NRZER - COUNTS NONZEROS IN R
C     CSTRAT - COLUMN STRATEGY (SEE SUBROUTINE REDUCE FOR VALUES)
C     RSTRAT - ROW STRATEGY (SEE SUBROUTINE GETNOD FOR VALUES)
C
      COMMON /IO/ NOUT,MOUT,INX,IOUT
      NOUT = 10
      MOUT = 6
      IOUT = 8
      INX = 5
      PFAC = 1
      MINOR = 0
      NOPER = 0
C
      READ(INX,102) TITLE
      WRITE(IOUT,203) TITLE
      READ(INX,101) IPRINT,IREP,NMOD,CSTRAT,RSTRAT
C
      CALL GETMAT(NROW,NCOL,NNZER,RADJ,ADJNCY,NONZER,WORKR1,WORKZ1,
     +            WORKZ2,TITLE)
C
      WRITE(IOUT,204) NROW,NCOL,NNZER,IPRINT,IREP,NMOD,CSTRAT,RSTRAT
C
      NROWP1 = NROW + 1
      NRZER = NCOL * (NCOL+1) / 2
C
      CALL SETUP(NROW,NCOL,NNZER,NROWP1,PARENT,CHILD,SIBLNG,RSTAT,
     +           SUPERN,WRKROW,WRK2,RADJ,ADJNCY,WORKR1,WORKR2,RSTAC,
     +           NONZER,CNONZ,DEGREE,RORDER,CORDER,CLIST,IPRINT)
C
C     COMPUTE NUMBER OF NONZEROS IN HALF OF X'X
      NZXX = 0
      DO 10 I = 1,NCOL
        NZXX = NZXX + DEGREE(I)
   10 CONTINUE
      NZXX = NZXX / 2
C
      CALL REDUCE(NROW,NCOL,NNZER,NROWP1,PARENT,CHILD,SIBLNG,RSTAT,
     +            SUPERN,WRKROW,WRK2,RADJ,ADJNCY,WORKR1,WORKR2,RSTAC,
     +            NONZER,CNONZ,DEGREE,RORDER,CORDER,CLIST,IPRINT,CSTRAT,
     +            NOPER,MINOR,RFAC,FACADJ,RROW,PFAC,NRZER,RSTRAT,NMOD,
     +            IREP)
C
      PFAC = PFAC - 1
      WRITE(IOUT,201) NNZER,NZXX,PFAC,MINOR,NOPER
C
      STOP
C
  101 FORMAT(20I4)
  102 FORMAT(20A4)
  201 FORMAT('1','ORIGINAL MATRIX: ',I8,' NONZEROS'/1X,
     +       'HALF OF X''X: ',I8,' NONZEROS'/1X,
     +       'RFACTOR: ',I8,' NONZEROS'/1X,
     +       'TOTAL OF ',I8,' GIVENS ROTATIONS'/1X,
     +       'TOTAL OF ',I8,' OPERATIONS')
  203 FORMAT(1X,20A4)
  204 FORMAT(1X,'ROWS = ',I5,'  COLUMNS = ',I5,'  NONZEROS = ',I6/1X,
     +       '  IPRINT = ',I2,'  IREP = ',I2,'  NMOD = ',I5,
     +       '  COLUMN STRATEGY = ',I2,'  ROW STRATEGY = ',I2)
      END
      SUBROUTINE ADD(IT,ARRAY,LEN,POINT)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     ADD  ADDS IT TO END OF ARRAY                                     C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER POINT,IT,LEN
      INTEGER ARRAY(LEN)
C
      POINT = POINT + 1
      ARRAY(POINT) = IT
      RETURN
      END
      SUBROUTINE ADJUST(WRKROW,RADJ,ADJNCY,SUPERN,NROWP1,NROW,NCOL,

     +                  NNZER,ROOT)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     ADJUST  ADJUSTS SUPERNODE CONTAINING ROOT FOR ALL COLUMNS        C
C             IN WRKROW, THEN ADJUSTS WORKROW FOR SUBTREE OF ROOT.     C
C             ALSO UPDATES NONZER OF NODE ROOT.                        C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NCOL,NROWP1,NNZER,ROOT,START,STOP
      INTEGER WRKROW(NCOL),RADJ(NROWP1),ADJNCY(NNZER),SUPERN(NROW)
C
      MROW = ROOT
C
C     ADJUST ROOT SUPERNODE FOR WORKROW
   10 CONTINUE
        START = RADJ(MROW)
        STOP = RADJ(MROW+1)-1
        DO 100 J = START,STOP
          ICOL = ADJNCY(J)
          IF(ICOL .LE. 0)GO TO 100
          IF(WRKROW(ICOL) .EQ. 0)GO TO 100
            ADJNCY(J) = -ICOL
  100   CONTINUE
        MROW = SUPERN(MROW)
      IF(MROW .GT. 0)GO TO 10
C
      RETURN
      END
      INTEGER FUNCTION ANCSTR(NODE,PARENT,RSTAT,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     ANCSTR  FINDS THE NEXT LIVING (NOT ELIMINATED) ANCESTOR OF NODE. C
C             IF NODE IS LIVING, RETURNS NODE.                         C
C             IF NO LIVING ANCESTOR, RETURNS ZERO.                     C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER PARENT(NROW),RSTAT(NROW)
C
      ANCSTR = NODE
C
   10 CONTINUE
        IF(ANCSTR .EQ. 0)RETURN
         IF(RSTAT(ANCSTR) .EQ. 0)RETURN
           ANCSTR = PARENT(ANCSTR)
      GO TO 10
C
      END
      INTEGER FUNCTION CFIND(COL,ROW,RADJ,ADJNCY,NROW,NROWP1,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     CFIND  CHECKS ROW FOR COL.  RETURNS INDEX IN ADJNCY IF FOUND,    C
C            RETURNS ZERO OTHERWISE.                                   C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER COL,ROW,START,STOP
      INTEGER RADJ(NROWP1),ADJNCY(NNZER)

C
      START = RADJ(ROW)
      STOP = RADJ(ROW+1)-1
C
      DO 100 I = START,STOP
        IF(ADJNCY(I) .EQ. COL)GO TO 200
  100 CONTINUE
C
      I = 0
  200 CONTINUE
      CFIND = I
      RETURN
      END
      INTEGER FUNCTION CKCHLD(NEWNOD,CHILD,SIBLNG,SUPERN,RSTAT,
     +                        WORKR2,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     CKCHLD  RETURNS 0 IF NO CHILDREN OF NEWNOD SUPERNODE ARE MARKED  C
C             IN WORKR2.  RETURNS 1 OTHERWISE.                         C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INTEGER CHILD(NROW),SIBLNG(NROW),WORKR2(NROW),SUPERN(NROW),
     +        RSTAT(NROW)
      INTEGER CLD
C
      CKCHLD = 0
C
      NOD = NEWNOD
C
   10 CONTINUE
        CLD = CHILD(NOD)
C
   20   CONTINUE
          IF(CLD .EQ. 0)GO TO 30
            IF(RSTAT(CLD) .EQ. 0 .AND. WORKR2(CLD) .EQ. 1)GO TO 40
              CLD = SIBLNG(CLD)
        GO TO 20
C
   30   CONTINUE
        NOD = SUPERN(NOD)
      IF(NOD .GT. 0)GO TO 10
      RETURN
C
   40 CONTINUE
      CKCHLD = 1
      RETURN
      END
      SUBROUTINE CNZUD(WRKROW,CNONZ,WORKR1,WORKR2,RADJ,RSTAT,ADJNCY,
     +                 PARENT,CHILD,SIBLNG,NROWP1,NROW,NNZER,NCOL)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     CNZUD  UPDATES NONZERO COUNTS FOR COLUMNS MARKED IN WRKROW       C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER COLNZ
      INTEGER WRKROW(NCOL),CNONZ(NCOL),WORKR1(NROW),WORKR2(NROW),
     +        RADJ(NROWP1),ADJNCY(NNZER),PARENT(NROW),
     +        CHILD(NROW),SIBLNG(NROW),RSTAT(NROW)
C
      DO 100 I = 1,NCOL
C
        IF(WRKROW(I) .EQ. 0)GO TO 100
C
        CNZ = COLNZ(I,WORKR1,WORKR2,RADJ,RSTAT,ADJNCY,PARENT,CHILD,
     +                 SIBLNG,NROW,NROWP1,NNZER)
        IF(CNZ .EQ. 0)CNZ = NROW + 1
        CNONZ(I) = CNZ
C
  100 CONTINUE
C
      RETURN
      END
      INTEGER FUNCTION COLNZ(DCOL,WORKR1,WORKR2,RADJ,RSTAT,ADJNCY,
     +                       PARENT,CHILD,SIBLNG,NROW,NROWP1,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     COLNZ RETURNS NUMBER OF NONZEROS IN COLUMN DCOL                  C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER DCOL,ANCSTR,CFIND,CROW
      INTEGER WORKR1(NROW),WORKR2(NROW),RADJ(NROWP1),ADJNCY(NNZER),
     +        PARENT(NROW),CHILD(NROW),SIBLNG(NROW),RSTAT(NROW)
C
      COLNZ = 0
C
      CALL INIT(WORKR2,NROW,0)
C
      DO 100 IROW = 1,NROW
C
        IF(WORKR2(IROW) .EQ. 1)GO TO 100
          IF(RSTAT(IROW) .GT. 1)GO TO 100
            I = CFIND(DCOL,IROW,RADJ,ADJNCY,NROW,NROWP1,NNZER)
            IF(I .EQ. 0)GO TO 100
              CROW = ANCSTR(IROW,PARENT,RSTAT,NROW)
              IF(CROW .EQ. 0)GO TO 100
                COLNZ = COLNZ + 1
                CALL GETUNL(CROW,PARENT,WORKR2,WORKR1,NWR1,NROW)
                IF(NWR1 .EQ. 0)GO TO 60
C
                  DO 50 L = 1,NWR1
                    JROW = WORKR1(L)
                    WORKR2(JROW) = 1
                    IF(RSTAT(JROW) .EQ. 0)COLNZ = COLNZ + 1
   50             CONTINUE
C
   60           CONTINUE
                CALL FAM(CHILD,SIBLNG,CROW,WORKR1,NWR1,NROW)
                CALL MARKIT(WORKR2,NROW,WORKR1,NROW,NWR1,1)
C
  100 CONTINUE
C

      RETURN
      END
      SUBROUTINE CUTREE(PARENT,CHILD,SIBLNG,NROW,ROOT)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     CUTREE  SEPARATES A SUBTREE ROOTED AT NODE ROOT                  C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER ROOT
      INTEGER PARENT(NROW),CHILD(NROW),SIBLNG(NROW)
      INTEGER PAR,SIB,CHLD
C
      PAR = PARENT(ROOT)
      IF(PAR .EQ. 0)RETURN
      SIB = SIBLNG(ROOT)
      CHLD = CHILD(PAR)
      PARENT(ROOT) = 0
      IF(CHLD .NE. ROOT)GO TO 100
C
C       DIRECT CHILD
          CHILD(PAR) = SIB
          SIBLNG(ROOT) = 0
          RETURN
C
  100 CONTINUE
C       CHILD IN SIBLING CHAIN
C
C       PASS OVER SIBLINGS
        LCHILD = CHLD
        CHLD = SIBLNG(LCHILD)
        IF(CHLD .NE. ROOT)GO TO 100
C
C       REMOVE ROOT FROM SIBLING CHAIN
        SIBLNG(LCHILD) = SIBLNG(ROOT)
        SIBLNG(ROOT) = 0
        RETURN
C
      END
      SUBROUTINE DEGUD(RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,WRKROW,
     +                 WRK2,WORKR1,WORKR2,DEGREE,NCOL,NROW,
     +                 NROWP1,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     DEGUD  DEGREE UPDATE                                             C
C            UPDATES DEGREE OF ALL COLUMNS IN WRKROW                   C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER RADJ(NROWP1),ADJNCY(NNZER),PARENT(NROW),WRK2(NCOL),
     +        CHILD(NROW),SIBLNG(NROW),WRKROW(NCOL),DEGREE(NCOL),
     +        WORKR1(NROW),RSTAT(NROW),WORKR2(NROW)
C
      INTEGER NEWDEG
C
      DO 100 I = 1,NCOL
C
        IF(WRKROW(I) .EQ. 0)GO TO 100

C
        DEG = NEWDEG(I,RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,WRK2,
     +                      WORKR1,WORKR2,NCOL,NROW,NROWP1,NNZER)
```

```
         IF(DEG .LT. 0)DEG = NCOL
         DEGREE(I) = DEG
C
  100 CONTINUE
C
      RETURN
      END
      SUBROUTINE DISCON(PROW,SUPERN,CHILD,SIBLNG,PARENT,RSTAT,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     DISCON  DISMANTLES A SUPERNODE AND MARKS ALL ITS NODES WITH      C
C             RSTAT=2, MEANING THAT THESE NODES NO LONGER              C
C             CONTRIBUTE TO THE BIGRAPH.                               C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER PROW,XNODE,CNODE
      INTEGER SUPERN(NROW),CHILD(NROW),SIBLNG(NROW),PARENT(NROW),
     +        RSTAT(NROW)
C
      XNODE = PROW
C
  100 CONTINUE
         CNODE = CHILD(XNODE)
         IF(CNODE .EQ. 0)GO TO 300
C
  200     CONTINUE
            CALL CUTREE(PARENT,CHILD,SIBLNG,NROW,CNODE)
            CNODE = CHILD(XNODE)
          IF(CNODE .GT. 0)GO TO 200
C
  300   CONTINUE
        RSTAT(XNODE) = 2
        XNODE = SUPERN(XNODE)
       IF(XNODE .GT. 0)GO TO 100
C
      RETURN
      END
      SUBROUTINE EXTRA(WRKROW,RADJ,ADJNCY,NEXTRA,NROWP1,NROW,NCOL,
     +                 WORKR1,NWR1,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     EXTRA  COUNTS NUMBER OF DISTINCT NONZERO COLUMNS IN ALL          C
C     NODES IN ARRAY WORKR1 WHICH ARE NOT RECORDED IN WRKROW.          C
C     THE COUNT IS STORED IN NEXTRA. WRKROW IS THEN UPDATED FOR        C
C     THESE COLUMNS.                                                   C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER WRKROW(NCOL),RADJ(NROWP1),ADJNCY(NNZER),WORKR1(NROW)
      INTEGER START,STOP
C
```

```
      NEXTRA = 0
C
      DO 100 I = 1,NWR1

C
        IROW = WORKR1(I)
        START = RADJ(IROW)
        STOP = RADJ(IROW+1)-1
C
        DO 50 J = START,STOP
          JCOL = ADJNCY(J)
          IF(JCOL .LE. 0)GO TO 50
          IF(WRKROW(JCOL) .GT. 0)GO TO 50
            NEXTRA = NEXTRA + 1
            WRKROW(JCOL) = 1
   50   CONTINUE
C
  100 CONTINUE
C
      RETURN
      END
      SUBROUTINE FAM(CHILD,SIBLNG,IROW,WORKR1,NWR1,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     FAM - PREORDER TRAVERSAL OF SUBTREE (USING ITS BINARY TREE       C
C           REP) STARTING AT NODE IROW. COLLECTS ALL NODES IN SUBTREE  C
C           INTO ARRAY WORKR1.                                         C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER IROW,NWR1
      INTEGER CHILD(NROW),SIBLNG(NROW),WORKR1(NROW)
C     NOTE: RSTAC MUST BE DIMENSIONED AT LEAST NROW/2
      INTEGER T,LT,PSTACK,RSTAC(100)
C
      LT = IROW
      PSTACK = 0
      NWR1 = 1
      WORKR1(1) = IROW
C
   50 CONTINUE
        T = CHILD(LT)
        IF(T .GT. 0)GO TO 100
          IF(PSTACK .EQ. 0)RETURN
          T = RSTAC(PSTACK)
          PSTACK = PSTACK - 1
  100   CONTINUE
        CALL ADD(T,WORKR1,NROW,NWR1)
        LT = T
```

```
          T = SIBLNG(LT)
          IF(T .GT. 0)CALL ADD(T,RSTAC,50,PSTACK)
          GO TO 50
C
      END
      INTEGER FUNCTION FROOT(ROW,PARENT,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     RETURNS ROOT OF TREE CONTAINING ROW                              C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER ROW,PARENT(NROW)
C
      NEXT = ROW
C
   10 CONTINUE
        FROOT = NEXT
        NEXT = PARENT(FROOT)
      IF(NEXT .NE. 0)GO TO 10
C
      RETURN
      END


      SUBROUTINE GETMAT(NROW,NCOL,NNZER,RADJ,ADJNCY,NONZER,WORKR1,
     +                  WORKZ1,WORKZ2,TITLE)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     GETMAT GETS A MATRIX TO BE REDUCED                               C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER RADJ(1),ADJNCY(1),NONZER(1),PROBL,WORKR1(1),
     +        WORKZ1(1),WORKZ2(1),TITLE(20)
      COMMON /IO/ NOUT,MOUT,INX,IOUT
C
C     HERE BELONGS USER SUPPLIED CODE TO READ IN OR GENERATE THE
C     NONZERO STRUCTURE OF A MATRIX IN ROW ADJACENCY FORM.
C     ADJNCY SHOULD CONTAIN A LIST OF COLUMN INDICES, AND RADJ
C     SHOULD CONTAIN POINTERS TO FIRST ENTRY OF EACH ROW IN ADJNCY.
C     RADJ(NROW+1) MUST EQUAL NNZER+1.
C
      RETURN
      END
      INTEGER FUNCTION GETNOD(RSTAC,PSTAC,NONZER,PARENT,CHILD,SIBLNG,
     +                        SUPERN,WORKR2,NROW,RSTAT,RSTRAT,WRKROW,
     +                        WRK2,NCOL,WORKR1,RADJ,NROWP1,ADJNCY,
     +                        NNZER,NEWEND)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     GETNOD  RETURNS THE NEXT NODE TO BE ORDERED, AND REPLACES        C
C             IT IN RSTAC BY ITS PARENT.                               C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER PSTAC,ANCSTR,RSTRAT,CKCHLD
      INTEGER RSTAC(NROW),NONZER(NROW),PARENT(NROW),CHILD(NROW),
     +        SIBLNG(NROW),RSTAT(NROW),WORKR2(NROW),SUPERN(NROW),
     +        WRKROW(NCOL),WRK2(NCOL),WORKR1(NROW),RADJ(NROWP1),
     +        ADJNCY(NNZER)
C
      NODIND = 1
      IF(PSTAC .EQ. 1)GO TO 100
C
      GO TO (10,20,30),RSTRAT
C
   10 CONTINUE
C       NATURAL ORDER (SMALLEST INDEX)
        NODIND = MININD(RSTAC,PSTAC,NROW)
        GO TO 100
C
   20 CONTINUE
C       LEAST NONZEROS (FIRST TIED)
        NODIND = MINRF(RSTAC,PSTAC,NONZER,NROW)
        GO TO 100
C
   30 CONTINUE
C       LEAST PIVOTAL ROW FILL
        NODIND = MPFILL(RSTAC,PSTAC,WRKROW,WRK2,WORKR1,NROW,NCOL,NONZER,
     +                  CHILD,SIBLNG,RADJ,NROWP1,ADJNCY,NNZER)
C
  100 CONTINUE
      GETNOD = RSTAC(NODIND)
      NEWNOD = ANCSTR(PARENT(GETNOD),PARENT,RSTAT,NROW)
      RSTAC(NODIND) = NEWNOD
      WORKR2(GETNOD) = 0
      IF(NEWNOD .EQ. 0)GO TO 200
C
C     CHECK IF CHILDREN OF NEWNOD SUPERNODE ARE MARKED IN WORKR2 (IF MARK
C     THEN NEWNOD HAS A DESCENDANT IN RSTAC, AND CANNOT GO INTO RSTAC)
      IF(CKCHLD(NEWNOD,CHILD,SIBLNG,SUPERN,RSTAT,WORKR2,NROW) .EQ. 0)
     +                                                            RETURN
  200 CONTINUE
      CALL REPACK(RSTAC,PSTAC,NODIND)
      RETURN
      END
      SUBROUTINE GETUNL(IROW,LINK,MARK,WORKR1,NWR1,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     PUTS ALL UNMARKED NODES LINKED TO IROW INTO WORKR1                C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER LINK(NROW),WORKR1(NROW),MARK(NROW)
C
      NWR1 = 0
      I = LINK(IROW)
      IF(I .EQ. 0)RETURN
C
   10 CONTINUE
        NWR1 = NWR1 + 1
        WORKR1(NWR1) = I
        I = LINK(I)
        IF(I .EQ. 0)RETURN
        IF(MARK(I) .EQ. 1)RETURN
      GO TO 10
C
      RETURN
      END
      SUBROUTINE INIT(LINE,N,SYMB)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     INIT  INITIALIZES LINE TO SYMB                                   C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER LINE(N),SYMB
C
      DO 10 I = 1,N
        LINE(I) = SYMB
   10 CONTINUE
C
      RETURN
      END
      SUBROUTINE JOIN(XNODE,JNODE,SUPERN,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     JOIN  SUPERNODE XNODE BECOMES PART OF SUPERNODE JNODE            C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER XNODE,JNODE,SUPERN(NROW)
      AJ = JNODE
C
C     FIND END OF JNODE CHAIN IN SUPERN
   10 CONTINUE
        LAJ = AJ
        AJ = SUPERN(LAJ)
      IF(AJ .GT. 0)GO TO 10
```

```
C
C     CONNECT JNODE LIST TO XNODE LIST
      SUPERN(LAJ) = XNODE
C
      RETURN
      END
      SUBROUTINE MAJOR(PCOL,RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,
     +                 SUPERN,NROW,NROWP1,NCOL,NONZER,WRKROW,WRK2,
     +                 NNZER,WORKR1,WORKR2,RSTAC,MINOR,NOPER,RSTRAT,
     +                 PROW,PSTAC)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                  C
C     MAJOR  PROCESSES A MAJOR STEP (A COLUMN)                     C
C                                                                  C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NROW,NCOL,NNZER,NROWP1,MINOR,NOPER,NZCNT,NEWEND,NEXNOD
      INTEGER RADJ(NROWP1),ADJNCY(NNZER),PARENT(NROW),CHILD(NROW),
     +        SIBLNG(NROW),NONZER(NROW),WRKROW(NCOL),WRK2(NCOL),
     +        WORKR1(NROW),WORKR2(NROW),RSTAC(NROW),RSTAT(NROW),
     +        SUPERN(NROW)
      INTEGER PSTAC,PCOL,PROW,RSTRAT,GETNOD
C
C
C     COUNTS # NONZEROS CURRENTLY IN PIVOT ROW
      NZCNT = 0
C
C     INITIALIZE WORKING ROW FOR ACCUMULATION OF PIVOT ROW STRUCTURE
      CALL INIT(WRKROW,NCOL,0)
C
C     FIND ALL ROWS INVOLVED IN THIS MAJOR STEP WHICH COMPETE
C     FOR FIRST PLACE, AND DELETE PCOL FROM ALL ROWS
      CALL RFIND(PCOL,RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,
     +           NROW,NROWP1,NNZER,RSTAC,PSTAC,WORKR1,WORKR2)
C
C     INITIALIZE ROOT OF TREE BEING CONSTRUCTED
      NEWEND = 0
C
      IF(PSTAC .GT. 0)GO TO 1
C       NO ROW WITH PCOL FOUND (NOTHING TO BE DONE IN THIS MAJOR STEP)
        PSTAC = -1
        RETURN
C
C     LOOP UNTIL ALL ROWS PROCESSED (UNTIL PSTAC = 0)
    1 CONTINUE
C
C       GET NEXT ROW NODE
        NEXNOD = GETNOD(RSTAC,PSTAC,NONZER,PARENT,CHILD,SIBLNG,
     +                  SUPERN,WORKR2,NROW,RSTAT,RSTRAT,WRKROW,
     +                  WRK2,NCOL,WORKR1,RADJ,NROWP1,ADJNCY,NNZER,
     +                  NEWEND)
```

```
      IF(NEWEND .EQ. 0)PROW = NEXNOD
C
C     CUT THE SUBTREE ROOTED AT NEXNOD FROM THE FOREST
      CALL CUTREE(PARENT,CHILD,SIBLNG,NROW,NEXNOD)
C
      IF(NEWEND .EQ. 0)GO TO 30
C       ADJUST SUPERNODE OF NEXNOD FOR WORKROW
        CALL ADJUST(WRKROW,RADJ,ADJNCY,SUPERN,NROWP1,NROW,NCOL,NNZER,
     +              NEXNOD)
C
        IF(IFLAG2 .EQ. 0)GO TO 30
C         ADJUST PIVOT ROW SUPERNODE FOR SUBTREE OF SECOND ROW
          CALL FAM(CHILD,SIBLNG,NEXNOD,WORKR1,NWR1,NROW)
          IF(NWR1 .EQ. 1)GO TO 30
            WORKR1(1) = WORKR1(NWR1)
            NWR1 = NWR1 - 1
            CALL INIT(WRK2,NCOL,0)
            CALL EXTRA(WRK2,RADJ,ADJNCY,NEXTRA,NROWP1,NROW,NCOL,
     +                 WORKR1,NWR1,NNZER)
            CALL ADJUST(WRK2,RADJ,ADJNCY,SUPERN,NROWP1,NROW,NCOL,
     +                  NNZER,PROW)
C
   30 CONTINUE
C
C     ADJUST WORKROW FOR SUBTREE OF NEXNOD
      CALL FAM(CHILD,SIBLNG,NEXNOD,WORKR1,NWR1,NROW)
      CALL EXTRA(WRKROW,RADJ,ADJNCY,NEXTRA,NROWP1,NROW,NCOL,WORKR1,
     +           NWR1,NNZER)
C     UPDATE NONZERO COUNT FOR CURRENT ROW
      NZCNT = NZCNT + NEXTRA
      NONZER(NEXNOD) = NZCNT
C
      IFLAG2 = 1
C     NOTE NEWEND = 0 ONLY WHEN PROCESSING PIVOT ROW
      IF(NEWEND .EQ. 0)GO TO 40
C       LOWER SECOND ROW FLAG
        IFLAG2 = 0
C       COUNT OPERATIONS
        NOPER = NOPER + NZCNT + 1
C       COUNT ROTATIONS
        MINOR = MINOR + 1
   40 CONTINUE
C
C     CONNECT THE TREE OF NEXNOD TO THE NEW TREE OF NEWEND
      CALL SETREE(PARENT,CHILD,SIBLNG,NROW,NEXNOD,NEWEND)
C
      IF(PSTAC .GT. 0)GO TO 1
C
      RETURN
      END
      SUBROUTINE MARKIT(IT,LEN1,POS,LEN2,NUM,SYMB)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C     MARKS POSITIONS POS OF IT BY SYMB                                 C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER IT(LEN1),POS(LEN2),SYMB
C
      IF(NUM .EQ. 0)RETURN
C
      DO 10 I = 1,NUM
        IT(POS(I)) = SYMB
   10 CONTINUE
C
      RETURN
      END
      INTEGER FUNCTION MINDEG(DEGREE,CNONZ,NCOL)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C     MINDEG  FINDS UNPROCESSED COLUMN OF MINIMUM DEGREE.                C
C     TIEBREAKING DONE WITH # OF NONZEROS IN A COLUMN.                  C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER DEGREE(NCOL),CNONZ(NCOL)
C
      MINDEG = 1
      MIN = DEGREE(1)
      MINNZ = CNONZ(1)
C
      DO 50 I = 2,NCOL
C
        DEG = DEGREE(I)
        IF(DEG .GT. MIN)GO TO 50
          IF(DEG .EQ. MIN)GO TO 40
            MINDEG = I
            MIN = DEG
            MINNZ = CNONZ(I)
            GO TO 50
   40     CONTINUE
          NZ = CNONZ(I)
          IF(NZ .GE. MINNZ)GO TO 50
          MINDEG = I
          MINNZ = NZ
C
   50 CONTINUE
C
C     SET DEGREE TO NCOL (LARGEST POSSIBLE DEGREE IS NCOL-1)
      DEGREE(MINDEG) = NCOL
      RETURN
      END
      INTEGER FUNCTION MINDG1(DEGREE,NCOL)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MINDG1  FINDS UNPROCESSED COLUMN OF MINIMUM DEGREE.              C
C     FIRST TIED COLUMN IS TAKEN.                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER DEGREE(NCOL)
C
      MINDG1 = 1
      MIN = DEGREE(1)
C
      DO 50 I = 2,NCOL
C
        DEG = DEGREE(I)
        IF(DEG .GE. MIN)GO TO 50
          MINDG1 = I
          MIN = DEG
C
   50 CONTINUE
C
C     SET DEGREE TO NCOL (LARGEST POSSIBLE DEGREE IS NCOL-1)
      DEGREE(MINDG1) = NCOL
C
      RETURN
      END
      INTEGER FUNCTION MINDG2(DEGREE,NCOL)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MINDG2  FINDS UNPROCESSED COLUMN OF MINIMUM DEGREE.              C
C     LAST TIED COLUMN IS TAKEN.                                       C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER DEGREE(NCOL)
C
      MINDG2 = 1
      MIN = DEGREE(1)
C
      DO 50 I = 2,NCOL
C
        DEG = DEGREE(I)
        IF(DEG .GT. MIN)GO TO 50
          MINDG2 = I
          MIN = DEG
C
   50 CONTINUE
C
C     SET DEGREE TO NCOL (LARGEST POSSIBLE DEGREE IS NCOL-1)
      DEGREE(MINDG2) = NCOL
      RETURN
      END
      INTEGER FUNCTION MININD(RSTAC,PSTAC,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MININD  RETURNS INDEX IN RSTAC OF SMALLEST ROW NUMBER            C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      INTEGER RSTAC(NROW),PSTAC
C
      MININD = 1
      MIN = RSTAC(1)
      IF(PSTAC .EQ. 1)RETURN
C
       DO 10 I = 2,PSTAC
         IND = RSTAC(I)
         IF(IND .GT. MIN)GO TO 10
         MIN = IND
         MININD = I
   10   CONTINUE
C
      RETURN
      END
      INTEGER FUNCTION MINRF(RSTAC,PSTAC,NONZER,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MINRF  RETUNS THE INDEX IN RSTAC OF THE FIRST TIED ROW NODE      C
C            WITH THE SMALLEST NONZERO COUNT IN NONZER.                C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER RSTAC(NROW),NONZER(NROW),PSTAC
C
      MINRF = 1
      MINZ = NONZER(RSTAC(1))
      IF(PSTAC .EQ. 1)RETURN
C
      DO 10 I = 2,PSTAC
       NONZ = NONZER(RSTAC(I))
       IF(NONZ .GE. MINZ)GO TO 10
         MINRF = I
         MINZ = NONZ
   10 CONTINUE
C
      RETURN
      END
      INTEGER FUNCTION MPFILL(RSTAC,PSTAC,WRKROW,WRK2,WORKR1,NROW,NCOL,
     +                        NONZER,CHILD,SIBLNG,RADJ,NROWP1,ADJNCY,
     +                        NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MPFILL  FINDS ROW IN RSTAC CAUSING MINIMUM FILL IN PIVOT ROW      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER RSTAC(NROW),WRKROW(NCOL),WRK2(NCOL),WORKR1(NROW),
     +        CHILD(NROW),SIBLNG(NROW),RADJ(NROWP1),ADJNCY(NNZER),
     +        NONZER(NROW)
      INTEGER PFILL,PSTAC,FILL
C
      MPFILL = 1
      MFILL = PFILL(RSTAC(1),WRKROW,WRK2,WORKR1,NROW,NCOL,CHILD,SIBLNG,
     +              RADJ,NROWP1,ADJNCY,NNZER)
      NONZP = NONZER(RSTAC(1))
C
      DO 10 I = 2,PSTAC
       IROW = RSTAC(I)
       FILL = PFILL(IROW,WRKROW,WRK2,WORKR1,NROW,NCOL,CHILD,SIBLNG,
     +              RADJ,NROWP1,ADJNCY,NNZER)
       IF(FILL .GT. MFILL)GO TO 10
         NZERO = NONZER(IROW)
         IF(FILL .NE. MFILL)GO TO 5
           IF(NZERO .GE. NONZP)GO TO 10
    5    CONTINUE
         NONZP = NZERO
         MPFILL = I
         MFILL = FILL
   10 CONTINUE
C
      RETURN
      END
      INTEGER FUNCTION MRJCJF(CNONZ,NCOL,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MRJCJF  FINDS UNPROCESSED COLUMN WITH MINIMUM # OF NONZEROS.     C
C     FIRST TIED COLUMN IS TAKEN.                                      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER CNONZ(NCOL)
C
      MRJCJF = 1
      MIN = CNONZ(1)
C
```

```
      DO 50 I = 2,NCOL
C
        NZ = CNONZ(I)
        IF(NZ .GE. MIN)GO TO 50
          MRJCJF = I
          MIN = NZ
C
   50 CONTINUE
C
C     SET CNONZ TO NROW+1 (LARGEST POSSIBLE CNONZ IS NROW)
      CNONZ(MRJCJF) = NROW + 1
C
      RETURN
      END
      INTEGER FUNCTION MRJCJL(CNONZ,NCOL,NROW)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     MRJCJL  FINDS UNPROCESSED COLUMN WITH MINIMUM # OF NONZEROS.     C
C     LAST TIED COLUMN IS TAKEN.                                       C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER CNONZ(NCOL)
C
      MRJCJL = 1
      MIN = CNONZ(1)
C
      DO 50 I = 2,NCOL
C
        NZ = CNONZ(I)
        IF(NZ .GT. MIN)GO TO 50
          MRJCJL = I
          MIN = NZ
C
   50 CONTINUE
C
C     SET CNONZ TO NROW+1 (LARGEST POSSIBLE CNONZ IS NROW)
      CNONZ(MRJCJL) = NROW + 1
C
      RETURN
      END
      INTEGER FUNCTION NEWDEG(DCOL,RADJ,RSTAT,ADJNCY,PARENT,CHILD,
     +                        SIBLNG,WRK2,WORKR1,WORKR2,NCOL,
     +                        NROW,NROWP1,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     NEWDEG RETURNS THE DEGREE OF COLUMN DCOL                         C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NCOL,NROWP1,NNZER,DCOL,FROOT,TROW,CFIND
      INTEGER RADJ(NROWP1),ADJNCY(NNZER),PARENT(NROW),CHILD(NROW),
     +        SIBLNG(NROW),WRK2(NCOL),WORKR1(NROW),
     +        WORKR2(NROW),RSTAT(NROW)
C
      CALL INIT(WORKR2,NROW,0)
      CALL INIT(WRK2,NCOL,0)
      NEWDEG = 0
C
      DO 200 IROW = 1,NROW
C
       IF(WORKR2(IROW) .EQ. 1)GO TO 200
         IF(RSTAT(IROW) .GT. 1)GO TO 200
           I = CFIND(DCOL,IROW,RADJ,ADJNCY,NROW,NROWP1,NNZER)
           IF(I .EQ. 0)GO TO 200
             TROW = FROOT(IROW,PARENT,NROW)
             CALL FAM(CHILD,SIBLNG,TROW,WORKR1,NWR1,NROW)
             CALL MARKIT(WORKR2,NROW,WORKR1,NROW,NWR1,1)
             CALL EXTRA(WRK2,RADJ,ADJNCY,NEXTRA,NROWP1,NROW,NCOL,
     +                  WORKR1,NWR1,NNZER)
             NEWDEG = NEWDEG + NEXTRA
C
  200 CONTINUE
C
      NEWDEG = NEWDEG - 1
C
      RETURN
      END
      INTEGER FUNCTION PFILL(RNODE,WRKROW,WRK2,WORKR1,NROW,NCOL,CHILD,
     +                       SIBLNG,RADJ,NROWP1,ADJNCY,NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     PFILL  RETURNS PIVOTAL ROW FILL CAUSED BY RNODE                  C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER WRKROW(NCOL),WRK2(NCOL),WORKR1(NROW),CHILD(NROW),
     +        SIBLNG(NROW),RADJ(NROWP1),ADJNCY(NNZER)
      INTEGER START,STOP,RNODE
C
      PFILL = 0
C
      CALL INIT(WRK2,NCOL,0)
      CALL FAM(CHILD,SIBLNG,RNODE,WORKR1,NWR1,NROW)
C
      DO 100 I = 1,NWR1
        IROW = WORKR1(I)
        START = RADJ(IROW)
        STOP = RADJ(IROW+1) - 1
        DO 50 J = START,STOP
          JCOL = ADJNCY(J)
          IF(JCOL .GT. 0)WRK2(JCOL) = 1
   50   CONTINUE
  100 CONTINUE
C
      DO 200 IC = 1,NCOL
        IND = WRK2(IC) - WRKROW(IC)
        IF(IND .EQ. 1)PFILL = PFILL + 1
  200 CONTINUE
C
      RETURN
      END
      SUBROUTINE PRTVEC(VECTOR,LEN,NAME)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     PRTVEC  PRINTS VECTOR ALONG WITH INDICES                         C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      COMMON /IO/ NOUT,MOUT,INX,IOUT
C
      INTEGER VECTOR(LEN)
      REAL NAME(2)
C
      WRITE(MOUT,201) NAME
C
      DO 20 I = 1,LEN,20
C
        II = MIN0(I + 19,LEN)
        WRITE(MOUT,101) (J,J=I,II)
        WRITE(MOUT,102) (VECTOR(J),J=I,II)
C
   20 CONTINUE
C
      RETURN
C
  101 FORMAT('0',33I4)
  102 FORMAT(' ',33I4)
  201 FORMAT(1X//1X,2A4)
      END
      SUBROUTINE PRTX(RADJ,RSTAT,ADJNCY,CHILD,SIBLNG,PARENT,NROW,NROWP1,
     +                NNZER,WORKR1,RSTAC,RORDER,CORDER,CLIST,II,RFAC,
     +                FACADJ,RROW,NRZER,NCOL,PFAC,IREP)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     PRTX  PRINTS THE NONZERO STRUCTURE OF A PARTIALLY FACTORED       C
C     MATRIX STORED IN B-BAR FORM WITH A FOREST ORDERING.              C
C     ROWS OF THE UNFACTORED PORTION ARE PRINTED GROUPED BY            C
C     TREES WITHIN THE FOREST, AND NODES OF EACH TREE ARE              C
C     PRINTED IN PREORDER.                                             C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NROWP1,NNZER,PFAC
      INTEGER RADJ(NROWP1),ADJNCY(NNZER),CHILD(NROW),SIBLNG(NROW),
     +        PARENT(NROW),WORKR1(NROW),RSTAC(NROW),RORDER(NCOL),
     +        CORDER(NCOL),CLIST(NCOL),RFAC(NCOL),FACADJ(NRZER),
     +        RSTAT(NROW),RROW(NCOL)
      INTEGER LINE(125),BLANK,STAR,PSTAC,CNUM
C
      COMMON /IO/ NOUT,MOUT,INX,IOUT
C
      DATA BLANK/'    '/,STAR/'*   '/
C
      ICOL = II
      DO 50 K = 1,NCOL
        CNUM = CLIST(K)
        IF(CNUM .EQ. 0)GO TO 50
        ICOL = ICOL + 1
        CORDER(CNUM) = ICOL
   50 CONTINUE
C
      WRITE(MOUT,103)
C
      IF(NCOL .LT. 100)GO TO 20
        CALL INIT(LINE,NCOL,0)
        DO 10 I = 100,NCOL
          LINE(CORDER(I)) = 1
   10   CONTINUE
        WRITE(MOUT,102) (LINE(K),K=1,NCOL)
C
   20 CONTINUE
      CALL INIT(LINE,NCOL,0)
      DO 30 I = 10,NCOL
        LINE(CORDER(I)) = MOD(I,100)/10
   30 CONTINUE
C
      WRITE(MOUT,102) (LINE(K),K=1,NCOL)
      CALL INIT(LINE,NCOL,0)
      DO 40 I = 1,NCOL
        LINE(CORDER(I)) = MOD(I,10)
   40 CONTINUE
      WRITE(MOUT,102) (LINE(K),K=1,NCOL)
C
      IF(II .LE. 0)GO TO 80
C
      DO 70 K = 1,II
        ICOL = RORDER(K)
        JSTART = RFAC(ICOL)
        JSTOP = PFAC - 1
        IF(K .EQ. II)GO TO 55
          ICOL2 = RORDER(K+1)
          JSTOP = RFAC(ICOL2)-1
   55   CONTINUE
```

```
      CALL INIT(LINE,125,BLANK)
      IF(JSTOP .LT. JSTART)GO TO 65
        DO 60 L = JSTART,JSTOP
          LINE(CORDER(FACADJ(L))) = STAR
 60     CONTINUE
 65   CONTINUE
     IROW = RROW(K)
      IF(IREP .GT. 0 .AND. RSTAT(IROW) .LT. 2)CALL REP(LINE,RADJ,
   +                           ADJNCY,CORDER,IROW,NCOL,NROWP1,NNZER)
      WRITE(MOUT,101) IROW,LINE
 70 CONTINUE
C
 80 CONTINUE
C
     DO 300 I = 1,NROW
       IF(PARENT(I) .GT. 0)GO TO 300
C        NODE WITHOUT PARENT IS ROOT OF A TREE
C        PUT ALL NODES(ROWS) IN THIS TREE INTO RSTAC
         CALL FAM(CHILD,SIBLNG,I,RSTAC,PSTAC,NROW)
         DO 250 J = 1,PSTAC
           IROW = RSTAC(J)
           IF(RSTAT(IROW) .GT. 0)GO TO 250
C
           CALL FAM(CHILD,SIBLNG,IROW,WORKR1,NWR1,NROW)
           CALL INIT(LINE,125,BLANK)
C
           DO 200 IR = 1,NWR1
C
             JSTART = RADJ(WORKR1(IR))
             JSTOP = RADJ(WORKR1(IR)+1)-1
C
             DO 100 JPTR = JSTART,JSTOP
               JCOL = ADJNCY(JPTR)
               IF(JCOL .GT. 0)LINE(CORDER(JCOL)) = STAR
 100         CONTINUE
C
 200       CONTINUE
C
           IF(IREP .GT. 0)CALL REP(LINE,RADJ,ADJNCY,CORDER,IROW,NCOL,
    +                                NROWP1,NNZER)
           WRITE(MOUT,101) IROW,LINE
C
 250     CONTINUE
C
 300 CONTINUE
C
     RETURN
C
```

```
  101 FORMAT(1X,I5,125A1)
  102 FORMAT(6X,125I1)
  103 FORMAT('1')
      END
      SUBROUTINE RECORD(WRKROW,NCOL,PCOL,RFAC,FACADJ,PFAC,NRZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C     RECORD  RECORDS NEW ROW OF R-FACTOR IN ROW-ADJACENCY FORM         C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NCOL,PCOL,PFAC
      INTEGER WRKROW(NCOL),RFAC(NCOL),FACADJ(NRZER)
C
      DO 10 I = 1,NCOL
C
        IF(WRKROW(I) .EQ. 0)GO TO 10
        FACADJ(PFAC) = I
        PFAC = PFAC + 1
C
   10 CONTINUE
C
      RETURN
      END
      SUBROUTINE REDUCE(NROW,NCOL,NNZER,NROWP1,PARENT,CHILD,SIBLNG,
     +                  RSTAT,SUPERN,WRKROW,WRK2,RADJ,ADJNCY,WORKR1,
     +                  WORKR2,RSTAC,NONZER,CNONZ,DEGREE,RORDER,CORDER,
     +                  CLIST,IPRINT,CSTRAT,NOPER,MINOR,RFAC,FACADJ,
     +                  RROW,PFAC,NRZER,RSTRAT,NMOD,IREP)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C     REDUCE  CONTROLS THE SYMBOLIC REDUCTION ACCORDING TO PARAMETERS   C
C             SPECIFIED.                                                C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER NROW,NCOL,NNZER,NROWP1,NOPER,MINOR,NRZER,NMOD
      INTEGER PARENT(NROW),CHILD(NROW),SIBLNG(NROW),WRK2(NCOL),
     +        WRKROW(NCOL),RADJ(NROWP1),ADJNCY(NNZER),WORKR1(NROW),
     +        RSTAC(NROW),NONZER(NROW),CNONZ(NCOL),DEGREE(NCOL),
     +        RORDER(NCOL),CORDER(NCOL),CLIST(NCOL),RFAC(NCOL),
     +        FACADJ(NRZER),RSTAT(NROW),SUPERN(NROW),WORKR2(NROW),
     +        RROW(NCOL)
      INTEGER PCOL,PROW,CSTRAT,PFAC,RSTRAT,PAR,PSTAC
      COMMON /IO/ NOUT,MOUT,INX,IOUT
C
      IF(IPRINT .GT. -3)WRITE(NOUT,156)
C
      DO 500 I = 1,NCOL
C
```

```
      GO TO (301,302,303,304,305,306),CSTRAT
C
  301 CONTINUE
C       MINIMUM DEGREE WITH TIEBREAKING
        PCOL = MINDEG(DEGREE,CNONZ,NCOL)
        GO TO 200
C
  302 CONTINUE
C       MINIMUM DEGREE, FIRST TIED
        PCOL = MINDG1(DEGREE,NCOL)
        GO TO 200
C
  303 CONTINUE
C       MINIMUM DEGREE, LAST TIED
        PCOL = MINDG2(DEGREE,NCOL)
        GO TO 200
C
  304 CONTINUE
C       MINIMUM COLUMN COUNT, FIRST TIED
        PCOL = MRJCJF(CNONZ,NCOL,NROW)
        GO TO 200
C
  305 CONTINUE
C       MINIMUM COLUMN COUNT, LAST TIED
        PCOL = MRJCJL(CNONZ,NCOL,NROW)
        GO TO 200
C
  306 CONTINUE
C       NATURAL ORDER
        PCOL = I
        IF(CNONZ(I) .EQ. NROW+1)GO TO 400
C
  200 CONTINUE
C
      CALL MAJOR(PCOL,RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,SUPERN,
     +           NROW,NROWP1,NCOL,NONZER,WRKROW,WRK2,NNZER,WORKR1,
     +           WORKR2,RSTAC,MINOR,NOPER,RSTRAT,PROW,PSTAC)
C
C     CHECK IF ANY PROCESSING DONE
      IF(PSTAC .EQ. -1)GO TO 400
C
C       MARK PIVOT ROW
        RSTAT(PROW) = 1
        RROW(I) = PROW
C
C       ADD PIVOT ROW SUPERNODE TO SUPERNODE OF ITS PARENT.  IF PIVOT
C       ROW HAS NO PARENT, DISCONNECT AND DISMANTLE SUPERNODE, AND
C       MARK THESE NODES WITH RSTAT=2.
        PAR = PARENT(PROW)
```

```
          IF(PAR .GT. 0)CALL JOIN(PROW,PAR,SUPERN,NROW)
          IF(PAR .EQ. 0)CALL DISCON(PROW,SUPERN,CHILD,SIBLNG,PARENT,
     +                              RSTAT,NROW)
C
          IF(IPRINT .GT. 1)CALL PRTVEC(RADJ,NROWP1,'RADJ     ')
          IF(IPRINT .GT. 1)CALL PRTVEC(RSTAT,NROW,'RSTAT    ')
          IF(IPRINT .GT. 1)CALL PRTVEC(SUPERN,NROW,'SUPERN   ')
          IF(IPRINT .GT. 1)CALL PRTVEC(ADJNCY,NNZER,'ADJNCY   ')
          IF(IPRINT .GT. 0)CALL PRTVEC(PARENT,NROW,'PARENT   ')
          IF(IPRINT .GT. 0)CALL PRTVEC(CHILD,NROW,'CHILD    ')
          IF(IPRINT .GT. 0)CALL PRTVEC(SIBLNG,NROW,'SIBLNG   ')
          IF(IPRINT .GT. 0)CALL PRTVEC(NONZER,NROW,'NONZER   ')
          IF(IPRINT .GT. 1)CALL PRTVEC(RFAC,NCOL,'RFAC     ')
          IF(IPRINT .GT. 1)CALL PRTVEC(FACADJ,PFAC,'FACADJ   ')
C
          IF(CSTRAT .LT. 4)CALL DEGUD(RADJ,RSTAT,ADJNCY,PARENT,CHILD,
     +                                SIBLNG,WRKROW,WRK2,WORKR1,WORKR2,
     +                                DEGREE,NCOL,NROW,NROWP1,NNZER)
C
          IF(IPRINT .GT. 0)CALL PRTVEC(DEGREE,NCOL,'DEGREE   ')
C
          IF(CSTRAT .EQ. 4 .OR. CSTRAT .EQ. 5)CALL CNZUD(WRKROW,CNONZ,
     +                          WORKR1,RSTAC,RADJ,RSTAT,ADJNCY,PARENT,
     +                          CHILD,SIBLNG,NROWP1,NROW,NNZER,NCOL)
C
C
C         MARK PIVOT COLUMN IN WORKROW
          WRKROW(PCOL) = 1
C
  400   CONTINUE
        CORDER(PCOL) = I
        RORDER(I) = PCOL
        CLIST(PCOL) = 0
C
C       PFAC POINTS TO NEXT AVAILABLE SPACE IN FACADJ
        RFAC(PCOL) = PFAC
C
C       RECORD THE STRUCTURE OF THE NEW ROW OF R-FACTOR
        CALL RECORD(WRKROW,NCOL,PCOL,RFAC,FACADJ,PFAC,NRZER)
C
        IF(IPRINT .GT. -1 .AND. MOD(I,NMOD) .EQ. 0)CALL PRTX(RADJ,
     +           RSTAT,ADJNCY,CHILD,SIBLNG,PARENT,NROW,NROWP1,NNZER,
     +           WORKR1,RSTAC,RORDER,CORDER,CLIST,I,RFAC,FACADJ,RROW,
     +           NRZER,NCOL,PFAC,IREP)
C
        IF(IPRINT .GT. -3)WRITE(NOUT,111) I,PCOL,PROW,PFAC,MINOR,NOPER
C
  500 CONTINUE
C
      RETURN
```

```
C
  111 FORMAT(1X,6I10)
  156 FORMAT(1X,'      STEP    COLUMN        ROW  R-NONZEROS',
     +       '  ROTATIONS  OPERATIONS')
      END
      SUBROUTINE REP(LINE,RADJ,ADJNCY,CORDER,IROW,NCOL,NROWP1,
     +               NNZER)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     REP  PLACES  X  INTO COLUMNS WHICH ARE REPRESENTED IN B-BAR      C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER LINE(125),RADJ(NROWP1),ADJNCY(NNZER),CORDER(NCOL)
      INTEGER START,STOP,EX
C
      DATA EX/'X   '/
C
      START = RADJ(IROW)
      STOP = RADJ(IROW+1) - 1
C
      DO 100 I = START,STOP
        ICOL = ADJNCY(I)
        IF(ICOL .GT. 0)LINE(CORDER(ICOL)) = EX
  100 CONTINUE
C
      RETURN
      END
      SUBROUTINE REPACK(IT,LEN,GAP)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     REPACK  REPACKS IT TO FILL THE GAP                               C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER IT(LEN),GAP
C
      LEN = LEN - 1
C
      DO 10 I = GAP,LEN
        IT(I) = IT(I+1)
   10 CONTINUE
      RETURN
      END
      SUBROUTINE RFIND(PCOL,RADJ,RSTAT,ADJNCY,PARENT,CHILD,
     +                 SIBLNG,NROW,NROWP1,NNZER,RSTAC,PSTAC,WORKR1,
     +                 WORKR2)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C     RFIND  FINDS ROWS CONTAINING PCOL COLUMN                         C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      INTEGER PCOL,NROWP1,NNZER,PSTAC,CFIND
      INTEGER RADJ(NROWP1),ADJNCY(NNZER),PARENT(NROW),RSTAC(NROW),
     +        CHILD(NROW),SIBLNG(NROW),WORKR1(NROW),WORKR2(NROW),
     +        RSTAT(NROW)
      INTEGER ANCSTR
C
      PSTAC = 0
      CALL INIT(WORKR2,NROW,0)
C
      DO 200 IROW = 1,NROW
C
        IF(WORKR2(IROW) .EQ. 1)GO TO 200
          IF(RSTAT(IROW) .GT. 1)GO TO 200
C
           I = CFIND(PCOL,IROW,RADJ,ADJNCY,NROW,NROWP1,NNZER)
           IF(I .EQ. 0)GO TO 200
C
C             IDENTIFIED ROW WITH PIVOT COLUMN
C               REMOVE PIVOT COLUMN
                ADJNCY(I) = PCOL*-1
C               FIND LIVING ANCESTOR
                JROW = ANCSTR(IROW,PARENT,RSTAT,NROW)
                IF(JROW .EQ. 0)GO TO 200
                  CALL ADD(JROW,RSTAC,NROW,PSTAC)
C                 MARK DESCENDANTS
                  CALL FAM(CHILD,SIBLNG,JROW,WORKR1,NWR1,NROW)
                  CALL MARKIT(WORKR2,NROW,WORKR1,NROW,NWR1,1)
C                 MARK UNMARKED ANCESTORS
                  CALL GETUNL(JROW,PARENT,WORKR2,WORKR1,NWR1,NROW)
                  CALL MARKIT(WORKR2,NROW,WORKR1,NROW,NWR1,1)
                  GO TO 200
C
  200 CONTINUE
C
      RETURN
      END
      SUBROUTINE SETREE(PARENT,CHILD,SIBLNG,NROW,ROOT,NEWEND)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                        C
C     SETREE  CONNECTS THE TREE OF NODE NEWEND TO NODE ROOT              C
C                                                                        C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      INTEGER ROOT,NEWEND
      INTEGER PARENT(NROW),CHILD(NROW),SIBLNG(NROW)
C
      IF(NEWEND .EQ. 0)GO TO 10
        PARENT(NEWEND) = ROOT
        SIBLNG(NEWEND) = CHILD(ROOT)
        CHILD(ROOT) = NEWEND
   10 CONTINUE
```

```
      NEWEND = ROOT
C
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
C     SETUP  INITIALIZES ARRAYS PRIOR TO BEGINNING OF REDUCTION       C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE SETUP(NROW,NCOL,NNZER,NROWP1,PARENT,CHILD,SIBLNG,
     +                 RSTAT,SUPERN,WRKROW,WRK2,RADJ,ADJNCY,WORKR1,
     +                 WORKR2,RSTAC,NONZER,CNONZ,DEGREE,RORDER,CORDER,
     +                 CLIST,IPRINT)
C
      INTEGER NROW,NCOL,NNZER,NROWP1,IPRINT
      INTEGER PARENT(NROW),CHILD(NROW),SIBLNG(NROW),WRK2(NCOL),
     +        WRKROW(NCOL),RADJ(NROWP1),ADJNCY(NNZER),WORKR1(NROW),
     +        RSTAC(NROW),NONZER(NROW),CNONZ(NCOL),DEGREE(NCOL),
     +        RORDER(NCOL),CORDER(NCOL),CLIST(NCOL),RSTAT(NROW),
     +        SUPERN(NROW),WORKR2(NROW)
      COMMON /IO/ NOUT,MOUT,INX,IOUT
C
      CALL INIT(PARENT,NROW,0)
      CALL INIT(CHILD,NROW,0)
      CALL INIT(SIBLNG,NROW,0)
      CALL INIT(RSTAT,NROW,0)
      CALL INIT(SUPERN,NROW,0)
      CALL INIT(WRKROW,NCOL,1)
      CALL DEGUD(RADJ,RSTAT,ADJNCY,PARENT,CHILD,SIBLNG,WRKROW,WRK2,
     +           WORKR1,WORKR2,DEGREE,NCOL,NROW,NROWP1,NNZER)
      CALL CNZUD(WRKROW,CNONZ,WORKR1,RSTAC,RADJ,RSTAT,ADJNCY,PARENT,
     +           CHILD,SIBLNG,NROWP1,NROW,NNZER,NCOL)
C
      LP = RADJ(1)
      DO 10 I = 1,NROW
        IP = RADJ(I+1)
        NONZER(I) = IP - LP
        LP = IP
   10 CONTINUE
C
      IF(IPRINT .GT. 1)CALL PRTVEC(RADJ,NROWP1,'RADJ     ')
      IF(IPRINT .GT. 1)CALL PRTVEC(ADJNCY,NNZER,'ADJNCY   ')
      IF(IPRINT .GT. 0)CALL PRTVEC(NONZER,NROW,'NONZER   ')
      IF(IPRINT .GT. 0)CALL PRTVEC(DEGREE,NCOL,'DEGREE   ')
      IF(IPRINT .GT. 0)CALL PRTVEC(CNONZ,NCOL,'CNONZ    ')
C
      DO 60 I = 1,NCOL
        CLIST(I) = I
        CORDER(I) = I
   60 CONTINUE
```

```
C
      IF(IPRINT .GT. -2)CALL PRTX(RADJ,RSTAT,ADJNCY,CHILD,SIBLNG,PARENT,
     +                   NROW,NROWP1,NNZER,WORKR1,RSTAC,RORDER,CORDER,
     +                   CLIST,0,RFAC,FACADJ,CNONZ,NRZER,NCOL,PFAC,1)
C
      RETURN
      END
      SUBROUTINE TREAD(R,ADJNCY,RADJ,NNZER,NROW,NROWP1,C)
      INTEGER R(NNZER),C(NNZER),RADJ(NROWP1),ADJNCY(NNZER)
      READ(5,11) (R(I),C(I),I=1,NNZER)
C
      K = 1
      DO 200 IROW = 1,NROW
        RADJ(I) = K
        DO 100 I = 1,NNZER
          IF(C(I) .NE. IROW)GO TO 100
          ADJNCY(K) = R(I)
          K = K + 1
  100   CONTINUE
  200 CONTINUE
      RADJ(NROWP1) = NNZER + 1
      RETURN
   11 FORMAT(4(2I4,12X))
      END
```

8. APPENDIX B

8.1 Structure of the Inverse of a Triangular Sparse Matrix

Let R be an upper triangular matrix of order n, and let A be its inverse. Then, $RA = AR = I$. Clearly, A must also be upper triangular. Let $G = (C;E)$ be the labelled graph associated with R, where $(c_i, c_j) \in E$ iff $r_{ij} \neq 0$ $(i < j)$.

Definition 8.1.1 A path $(c_{i_1}, c_{i_2}, \cdots, c_{i_\lambda})$ in $G = (C;E)$ is monotone if $i_1 < i_2 < \cdots < i_\lambda$.

Theorem 8.1.1 A monotone path exists from node i to node j in the graph G iff $a_{ij} \neq 0$ (assuming no cancellation in calculation of the inverse).

Proof: $A = \frac{R_{adj}}{|R|}$, where $|R|$ is the determinant of R, and R_{adj} is the adjoint of R. So each element of A is

$$a_{ij} = (-1)^{i+j}\sum_p \pm \left[\prod_{k=1}^{n} r_{kk}\right]^{-1}\left[r_{p(1),1} \cdots r_{p(i-1),i-1}\, r_{p(i+1),i+1} \cdots r_{p(n),n}\right]$$

where the summation is taken over all permutations p of $\{1, 2, \cdots, j-1, j+1, \cdots, n\}$, and the sign depends on whether p is even or odd. Since R is upper triangular, $r_{uv} = 0$ when $u > v$, and only permutations which satisfy $p(m) \leqslant m$, $m = 1, 2, \cdots, n$ can produce a nonzero term in the sum. Now $p(m) \leqslant m$ implies that $p(1) = 1$, $p(2) = 2$, $\cdots$, $p(i-1) = i-1$, and $p(n) = n$, $p(n-1) = n-1$, $\cdots$, $p(j+1) = j+1$. So

$$a_{ij} = (-1)^{i+j}\sum_{p} \pm \left[r_{ii} \ \cdots \ r_{jj}\right]^{-1}\left[r_{p(i+1),i+1} \ \cdots \ r_{p(j),j}\right].$$

It remains to assign $i, \cdots, j-1$ to $p(i+1), \cdots, p(j)$, such that $p(m) \leqslant m$. A typical nonzero term in the above sum is produced as follows. Suppose we first assign i to $p(k_1)$, where $i+1 \leqslant k_1 \leqslant j$, so $p(k_1) = i$. This means that $p(m) = m$ for $i < m < k_1$, so $i, \cdots, k_1-1$ are assigned. Next assign k_1 to $p(k_2)$, where $k_1+1 \leqslant k_2 \leqslant j$, so that $p(k_2) = k_1$ and $p(m) = m$ for $m = k_1, \cdots, k_2$. So at this point $i, \cdots, k_2-1$ are assigned. Suppose this is continued for a total of s times, thus assigning $i, \cdots, k_s-1$. Finally let $p(j) = k_s$, so that $p(m) = m$ for $m = k_s, \cdots, j$. Thus all $i, \cdots, j-1$ are assigned, and this gives

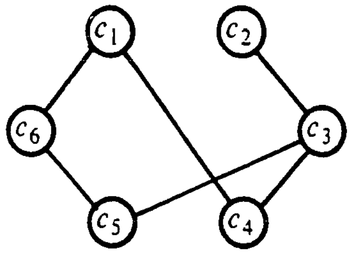
$$\left[r_{ii}r_{k_1k_1}r_{k_2k_2} \ \cdots \ r_{k_{s-1}k_{s-1}}r_{k_sk_s}r_{jj}\right]^{-1}\left[r_{ik_1}r_{k_1k_2} \ \cdots \ r_{k_{s-1}k_s}r_{k_sj}\right],$$

where $i < k_1 < k_2 < \cdots < k_s < j$, as the form of a typical term in the above sum. Note that all elements in the denominator are nonzero, and each nonzero element in the numerator is represented by an edge in G. If all the elements in the numerator are nonzero, the corresponding edges in G give a monotone path from node i to node j. If $a_{ij} \neq 0$, then at least one term in the sum must be nonzero, so there exists a monotone path from node i to node j in G. Conversely if there is a monotone path from node i to node j in G, this can be represented by a product such as the one in the above numerator with all the elements nonzero. Since the summation is over all permutations p, the nonzero product must be part of the sum, so assuming that cancellation does not occur in the sum we have $a_{ij} \neq 0$. $\square$

Figure 8.1 gives an example of an upper triangular matrix, and the structure of its inverse obtained by this theorem. To apply the result to a lower triangular matrix, simply take its transpose.

A possible use of the above theorem is in the explicit calculation of a variance-covariance matrix of estimated parameters in a sparse least squares problem. This matrix is given by $R^{-1}(R^{-1})'$, where R is the sparse upper triangular factor from Givens reduction of the data matrix X.

$$R = \begin{bmatrix} * & & & * & & * \\ & * & * & & & \\ & & * & * & * & \\ & & & * & & \\ & & & & * & * \\ & & & & & * \end{bmatrix}$$



$$A = \begin{bmatrix} * & & & * & & * \\ & * & * & \circledast & \circledast & \circledast \\ & & * & * & * & \circledast \\ & & & * & & \\ & & & & * & * \\ & & & & & * \end{bmatrix}$$

monotone paths:

(c_1,c_4), (c_1,c_6),

(c_2,c_3), (c_2,c_3,c_4), (c_2,c_3,c_5), (c_2,c_3,c_5,c_6),

(c_3,c_4), (c_3,c_5), (c_3,c_5,c_6),

(c_5,c_6)

Figure 8.1 An example of an upper triangular sparse matrix structure R, its graph representation, and the structure of its inverse A. The fill-in entries produced in A are circled. Note that each monotone path corresponds to exactly one off diagonal nonzero in A

9. ACKNOWLEDGEMENTS

I wish to acknowledge and thank Professor William J. Kennedy for suggesting this area of research, and for his advice and encouragement during the research.

The Mathematics and Statistics Research Department at Oak Ridge National Laboratory has provided support during preparation of this manuscript, and I wish to thank them also.

Finally, I wish to express my sincere thanks to my parents for their encouragement and moral support throughout the years of my education.