ON THE APPLICATION OF THE MINIMUM DEGREE ALGORITHM TO FINITE ELEMENT SYSTEMS

by
Alan George
and
David R. McIntyre
Department of Computer Science
University of Waterloo

CS-76-16

March 1976

ON THE APPLICATION OF THE MINIMUM DEGREE ALGORITHM TO FINITE ELEMENT SYSTEMS

by

Alan George

and

David R. McIntyre

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

ABSTRACT

We describe an efficient implementation of the so-called minimum degree algorithm, which experience has shown to produce efficient orderings for sparse positive definite systems. Our algorithm is a modification of the original, tailored to finite element problems, and is shown to induce a partitioning in a natural way. The partitioning is then refined so as to significantly reduce the number of non-null off-diagonal blocks. This refinement is important in practical terms because it reduces storage overhead in our linear equation solver, which utilizes the ordering and partitioning produced by our algorithm. Finally, we provide some numerical experiments comparing our ordering/solver package to more conventional band-oriented packages.

§1 Introduction

In this paper we consider the problem of directly solving the linear equations

(1.1) $Ax = b$,

where A is a sparse N by N positive definite matrix arising in certain finite element applications. We solve (1.1) using Cholesky's method or symmetric Gaussian elimination by first factoring A into the product LL^T, where L is lower triangular, and then solving the triangular systems $Ly = b$ and $L^Tx = y$.

It is well known that when a sparse matrix is factored using Cholesky's method, the matrix normally suffers fill; that is, the triangular factor L will typically have nonzeros in some of the positions which are zero in A. Thus, with the usual assumption that exact numerical cancellation does not occur, $L+L^T$ is usually fuller than A.

For any N by N permutation matrix P, the matrix PAP^T remains sparse and positive definite, so Cholesky's method still applies. Thus, we could instead solve

(1.2) $(PAP^T)(Px) = Pb$.

In general, the permuted matrix PAP^T fills in differently, and a judicious choice of P can often drastically reduce fill. If zeros are exploited, this can in turn imply a reduction in storage requirements and/or arithmetic requirements for the linear equation solver. The permutation P may also be chosen to ease data management, simplify coding etc. In general, these varying desiderata conflict, and various compromises must be made.

A heuristic algorithm which has been found to be very effective in finding efficient orderings (in the low-fill and low-arithmetic sense) is the so-called minimum degree algorithm [10]. The ordering algorithm we propose in this paper is a modification of this algorithm, changed in a number of ways to improve its performance for finite element matrix problems, which we now characterize.

Let M be any mesh formed by subdividing a planar region R with boundary ∂R by a number of lines, all of which terminate on a line or on ∂R. The mesh so formed consists of a set of regions enclosed by lines, which we call elements. The mesh M has a node at each vertex (a point of intersection of lines and/or ∂R), and may also have nodes on the lines, on ∂R, and in the interiors of the elements. An example of such a finite element mesh is given in Figure 1.1.

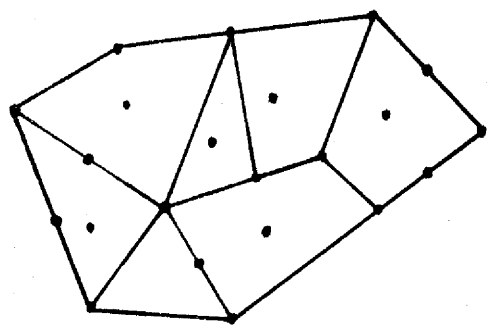


Figure 1.1 An example of a finite element mesh

Now let M have N nodes, labelled from 1 to N, and associate a variable x_i with the i-th node.

Definition 1.1 [4]

A finite element system of equations associated with the finite element mesh M is any N by N symmetric positive definite system $Ax = b$ having the property that $A_{ij} \neq 0$ if and only if x_i and x_j are associated with nodes of the same element. □

In one respect this definition is more general than required to characterize matrix problems arising in actual finite element applications in two dimensions. Usually M is restricted to consist of triangles or quadrilaterals, with adjacent elements having a common side. However, just as in [4], we intend to associate such meshes with matrices which arise when Gaussian elimination is applied to A, and these matrices require meshes having a less restricted topology in order that the correspondence be correct in the sense of Definition 1.1.

In a second respect, the above definition is not quite general enough to cover many matrix problems which arise in finite element applications because more than one variable is often associated with each node. However, the extension of our ideas to this situation is immediate, so to simplify the presentation we assume only one variable is associated with each node. (The code which produced the numerical results of section 5 works for this more general case with no changes.) Thus, in this paper we make no distinction between nodes and variables in this paper.

An outline of the paper is as follows. In section 2 we review two closely related models for symmetric Gaussian elimination, and describe the basic minimum degree algorithm. In section 3 we show how the special structure of finite element matrix problems can be exploited in the implementation of the minimum degree algorithm. We also show how the algorithm induces a natural partitioning of the matrix. In section 4 we describe a refinement of the partitioning produced by our version of the minimum degree algorithm which normally leads to a considerable reduction in the number of non-null off-diagonal blocks of the partitioning. This refinement is important since it reduces storage overhead when the matrix is processed as a block matrix, with only the non-null blocks being stored. Section 5 contains a brief description of the method of computer implementation of the ideas of sections 3 and 4, along with some numerical results. Section 6 contains our concluding remarks.

§2 Models for the Analysis of Sparse Symmetric Elimination

Following George [4] and Rose [10], we now review for completeness some basics of the elimination process for sparse positive definite matrices. We begin with some basic graph theory notions and define some quantities we need in subsequent sections. Much of the notation and the correspondence between symmetric Gaussian elimination and graph theory is due to the work of Parter [9] and Rose [10].

An undirected graph $G = (X,E)$ consists of a finite nonempty set X of nodes together with a set E of edges, which are unordered pairs of distinct nodes of X. A graph $G' = (X',E')$ is a subgraph of $G = (X,E)$ if $X' \subset X$ and $E' \subset E$. The nodes x and y of G are adjacent (connected) if $(x,y) \in E$. For $Y \subset X$, the adjacent set of Y, denoted by $\mathrm{Adj}(Y)$, is

$$\mathrm{Adj}(Y) = \{x | x \in X \backslash Y \text{ and } \exists y \in Y \ni (x,y) \in E\}.$$

When Y is a single node y, we write $\mathrm{Adj}(y)$ rather than $\mathrm{Adj}(\{y\})$. The degree of a node x, denoted by $\deg(x)$, is the number $|\mathrm{Adj}(x)|$, where $|S|$ denotes the cardinality of the finite set S. The incidence set of Y, $Y \subset X$, is denoted by $\mathrm{Inc}(Y)$ and defined by

$$\mathrm{Inc}(Y) = \{(x,y) | y \in Y \text{ and } x \in \mathrm{Adj}(Y)\}.$$

For a graph $G = (X,E)$ with $|X| = N$, an ordering (numbering, labelling) of G is a bijective mapping $\alpha:\{1,2,\ldots,N\} \to X$. We denote the labelled graph and node set by G^α and X^α respectively.

A path in G is a sequence of distinct edges $\{x_0,x_1\},\{x_1,x_2\}\ldots\{x_{r-1},x_r\}$ where all nodes except possibly x_0 and x_r are distinct. If $x_0 = x_r$, then the path is a cycle. The distance between two nodes is the length of the shortest path joining them. A graph G is connected if every pair of nodes is connected by at least one path. If G is disconnected, it consists of two or more maximal connected components.

We now establish a correspondence between graphs and matrices. Let A be an N by N symmetric matrix. The labelled undirected graph corresponding to A is denoted by $G^A = (X^A,E^A)$, and is one for which X^A is labelled as the rows of A, and $(x_i,x_j) \in E^A \Longleftrightarrow A_{ij} \neq 0,\ i \neq j$. The unlabelled graph corresponding to A is simply G^A with its labels removed. Obviously, for any N by N permutation matrix $P \neq I$, the unlabelled graphs of A and PAP^T are identical but the associated labellings differ. Thus, finding a good ordering of A can be viewed as finding a good labelling for the graph associated with A.

A symmetric matrix A is reducible if there exists a permutation matrix P such that PAP^T is block diagonal, with more than one diagonal block. This implies G^A is disconnected. In terms of solving linear equations, this means that solving $Ax = b$ can be reduced to that of solving two or more smaller problems. Thus, in this paper we assume A is irreducible, which means that the graph associated with A is connected.

Following Rose [10], we now make the connection between symmetric Gaussian elimination applied to A, and the corresponding graph transformations on G^A. Symmetric Gaussian elimination applied to A can be described by the following equations.

$$A = A_0 = H_0 = \begin{bmatrix} d_1 & v_1^T \\ v_1 & \bar{H}_1 \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ \frac{v_1}{\sqrt{d_1}} & I_{N-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \begin{bmatrix} \sqrt{d_1} & v_1^T/\sqrt{d_1} \\ 0 & I_{N-1} \end{bmatrix}$$

$$= \quad L_1 A_1 L_1^T,$$

(2.1) $$A_1 = \begin{bmatrix} 1 & & 0 \\ & d_2 & v_2^T \\ 0 & v_2 & \bar{H}_2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & \sqrt{d_2} & \\ 0 & \frac{v_2}{\sqrt{d_2}} & I_{N-2} \end{bmatrix} \begin{bmatrix} 1 & & 0 \\ & 1 & 0 \\ 0 & 0 & H_2 \end{bmatrix} \begin{bmatrix} 1 & & 0 \\ & \sqrt{d_2} & v_2^T/\sqrt{d_2} \\ & & I_{N-2} \end{bmatrix}$$

$$= L_2 A_2 L_2^T$$

$$\vdots$$

$$A_N = I$$

It is easy to verify that $A = LL^T$, where

(2.2) $$L = (\sum_{i=1}^{N} L_i) - (N-1)I.$$

Consider now the labelled graph G^A, with the labelling denoted by the mapping α. The deficiency Def(x) is the set of all pairs of distinct nodes in Adj(x) which are not themselves adjacent. Thus,

$$\text{Def}(x) = \{\{y,z\} | y,z \in \text{Adj}(x),\ y \neq z,\ y \notin \text{Adj}(z)\}.$$

For a graph $G = (X,E)$ and a subset $C \subseteq X$, the section graph $G(C)$ is the subgraph $G = (C,E(C))$, where

$$E(C) = \{(x,y) | (x,y) \in E, x \in C, y \in C\}.$$

Given a vertex y of a graph G, define the y-elimination graph G_y by

$$G_y = \{X \backslash \{y\}, E \backslash \mathrm{Inc}(y) \cup \mathrm{Def}(y)\}.$$

The sequence of elimination graphs $G_1, G_2, \ldots, G_{N-1}$ is then defined by $G_1 = G_{x_1}$ and $G_i = (G_{i-1})_{x_i}$, $i = 2,3,\ldots,N-1$.

The elimination graph G_i, $0 < i < N$, is simply the graph associated with the matrix H_i; i.e., $G_i = G^{H_i}$. We define $G_0 = G^A$, and note that G_{N-1} consists of a single node. The recipe for obtaining G_i from G_{i-1}, which is to delete x_i and its incident edges, and to then add edges so that $\mathrm{Adj}(x_i)$ is a clique, is due to Parter [9]. A clique is a set of nodes all of which are adjacent to each other.

This graph model has many advantages for describing and analyzing sparse matrix computations. However, except for rather small examples, it is not easy to visualize; although G_0 may sometimes be planar, the G_i rapidly become nonplanar with increasing i and become difficult to draw and

and interpret. For our class of matrix problems, which are associated with planar mesh problems, we can define a sequence of finite element meshes $M = M_0, M_1, \ldots, M_N$ such that G_i can easily be constructed from M_i, $i = 0,1,\ldots,N-1$.

Formally, a mesh $M = (X,S)$ is an ordered pair of sets, with X a (possibly empty) set of nodes and S a set of mesh lines, where each mesh line either joins two nodes, is incident to only one node (forming a loop), or else forms a nodeless loop. Let $M_0 = (X_0,S_0)$ be the original finite element mesh M, with nodes of the mesh forming the set X_0, and the lines joining the nodes comprising the set S_0. A <u>boundary mesh line</u> is a member of S shared by only one element.

Starting with M_0, the mesh $M_i = (X_i,S_i)$, $i = 1,2,\ldots,N$ is obtained from $M_{i-1} = (X_{i-1},S_{i-1})$ by

a) Deleting node x_i and its non-boundary incident mesh lines.

b) Repeatedly deleting mesh lines incident to a node having degree equal to one.

Here the degree of a node y in a mesh is the number of times mesh lines are incident to y. When x_i is eliminated from M_{i-1} and x_i has incident boundary lines, these boundary lines are simply "fused" to form a new line of S_i, as depicted in the transformations $M_5 \rightarrow M_6$ and $M_{12} \rightarrow M_{13}$ in Figure 2.1. The application of step b) is illustrated in the transformation $M_6 \rightarrow M_7$ and $M_{11} \rightarrow M_{12}$.

We now describe how to obtain G_i from M_i. Since the node sets are identical, we need only describe how to construct E_i. Since the sequence M_i is generated by removing nodes and/or mesh lines, the meshes of the sequence are all planar, having faces (elements) with nodes on their periphery and/or

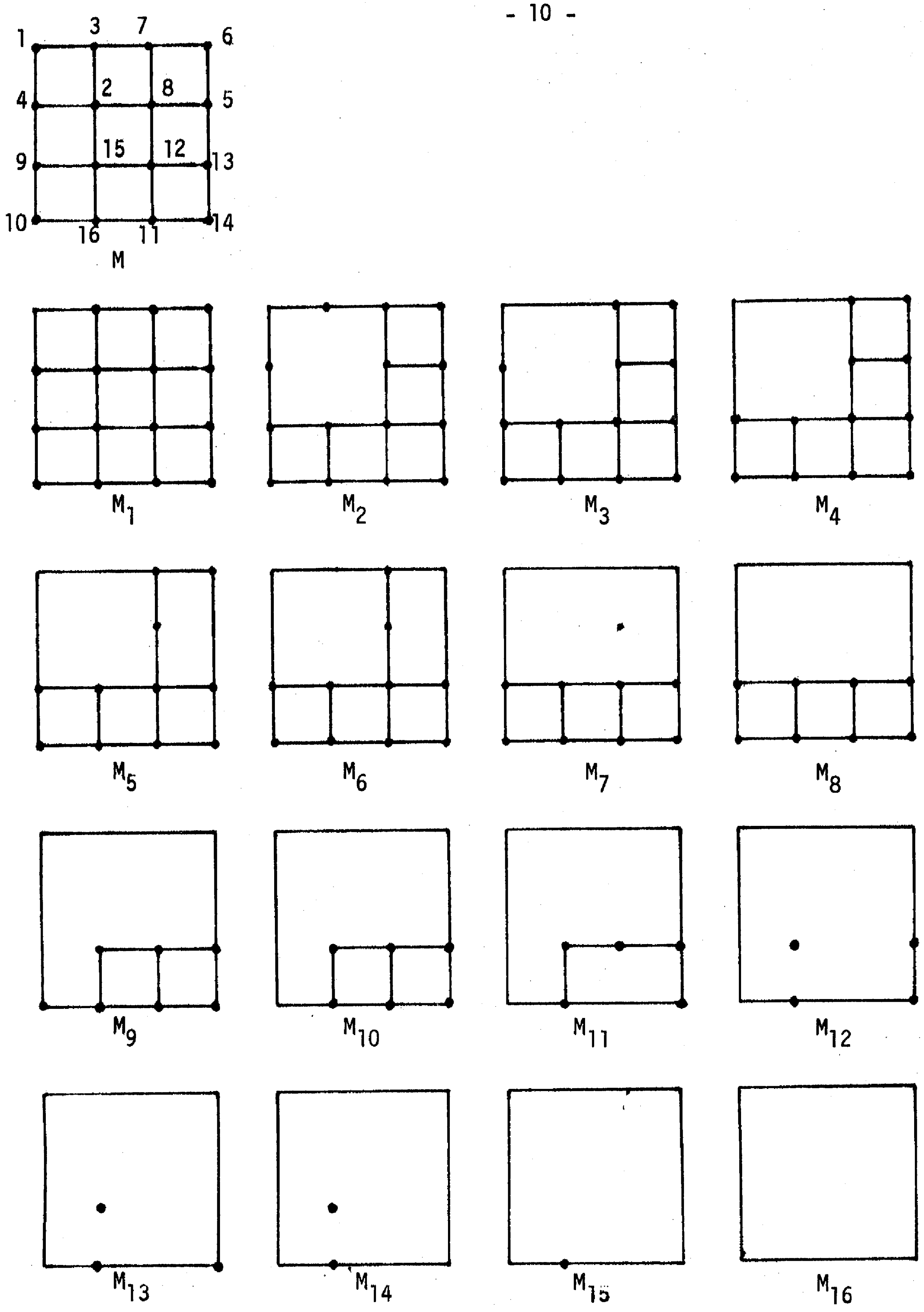


Figure 2.1 Meshes M_1 through M_{16}

in their interior as shown in Figure 2.1. Also recall that by definition 1.1, G_0 is a graph such that each set of nodes $C_0^\ell \subseteq X_0$ associated with an element (interior to and/or on the periphery of a face of M_0) forms a clique. This construction is illustrated in Figure 2.2.

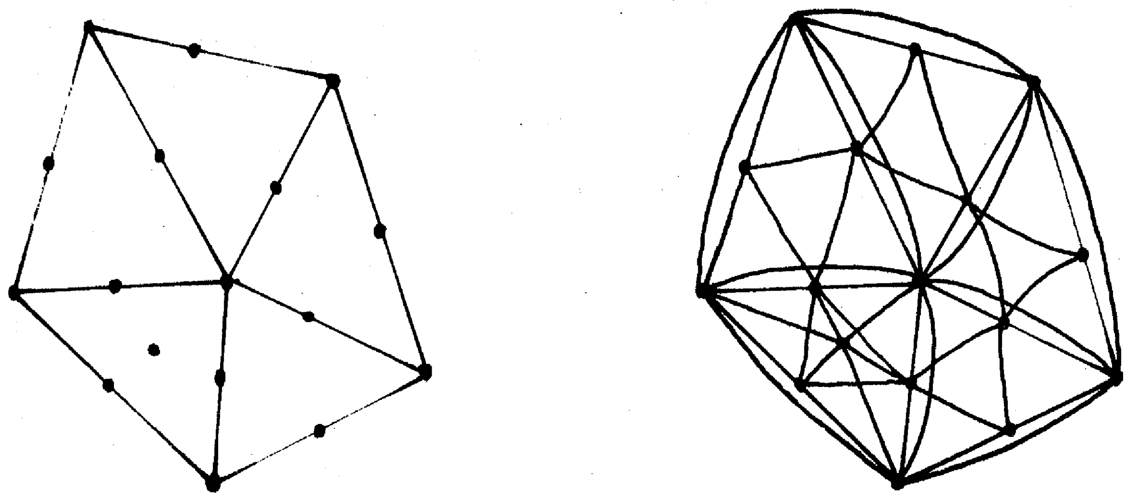


Figure 2.2 A mesh M_0 and corresponding graph G_0

The construction of G_i from M_i is essentially the same. The graph G_i is one having the same node set X_i as M_i, and which has an edge set E_i such that each set of nodes $C_i^\ell \subseteq X_i$ associated with the same mesh element forms a clique.

Using this mesh model, every numbering of $M = M_0$ determines a sequence of meshes M_i, $i = 1,2,\ldots,N$ which precisely reflect the structure of the part of the matrix remaining to be factored (H_i). The mesh M_N consists of a single element, devoid of nodes, whose boundary is ∂R. Thus, symmetric

Gaussian elimination on finite element matrices can be viewed in terms transforming finite element meshes, by a sequence of node and line removals, to a single element.

As mentioned before, our mesh model has the advantage that the meshes are planar and therefore easy to visualize and interpret. In addition, people involved in the application of the finite element method are accustomed to thinking in terms of elements, super-elements, substructures etc. [1,12]. The graph model, on the other hand, even if initially planar, rapidly becomes nonplanar as the elimination proceeds, and is less easy to visualize. It has the advantage that there is some well established notation and terminology for the description and manipulation of graphs. Our attitude is that the connection between the models is so close that statements about one can immediately be recast in terms of the other. We will therefore use both models in this paper, choosing the one which appears to transmit the information in the most lucid manner.

We now describe the minimum degree algorithm using the graph theory model. Let $G_0 = (X_0, E_0)$ be an unlabelled graph. The minimum degree algorithm labels X (determines the mapping α) according to the following pseudo-Algol program:

```
for $i = 1,2,\ldots,N$ do
1)    Find $y \in X_{i-1}$ such that $\deg(y) \le \deg(x)$ for all $x \in X_{i-1}$.
2)    Set $\alpha^{-1}(y) = i$ and if $i < N$ form $G_i$ from $G_{i-1}$.
```

Various strategies for breaking ties have been proposed, but in our application we found they made little difference to the quality of the ordering produced. Thus, we break ties arbitrarily.

Notice that the ordering algorithm requires knowledge about the current state of the factorization; that is, the choice of the i-th variable is a function of the structure of the partially factored matrix. Thus, one could argue that the ordering algorithm (i.e., the pivot selection) should be imbedded in the actual numerical factorization code. (Of course, this is more or less essential when the pivot selection is partially determined by numerical stability considerations.)

However, in our situation we have the option of isolating the ordering and the factorization in separate modules, and we prefer to do so for the following reasons:

1. If the ordering is done a-priori, the factorization code can utilize a static data structure, since the positions where fill will occur can be determined during the ordering. If the ordering is imbedded in the factorization, the data structure must be adaptive to accommodate fill as it occurs. By isolating the ordering and factorization, data structures can be tailored specifically for each function.

2. In some engineering design applications, many problems with the same matrix A, or with coefficient matrices having the same structure, must be solved. In these situations, it makes sense to find an efficient ordering and set up the appropriate data structures only once.

§3 The Minimum Degree Algorithm

From the graph model developed in section 2, we see that symmetric Gaussian elimination can be viewed as transforming G^A by a sequence of graph transformation rules to one having a single node and no edges. Using the mesh model, we see that the algorithm can also be viewed as transforming the original mesh according to some precise node and line removal rules so that the final mesh M_N consists of a single nodeless element. We also established a correspondence between the two models, so that G_i can be constructed from M_i, $0 \leq i < N$. In this section we show that for our finite element matrix problems, the local behavior of the minimum degree algorithm can be precisely characterized. In addition, we show that the algorithm induces a natural partitioning of the node set X, or equivalently, of the reordered matrix A. In the next section we show how this ordering can be refined so that the triangular factor L of A can be efficiently stored.

We begin with some definitions. For a graph $G = (X,E)$, let $C \subseteq X$ and let $G(C) = (C,E(C))$ be the corresponding section graph of G determined by C.(see section 2). The node x is an <u>interior node</u> of C if $x \in C$ and $\mathrm{Adj}(x) \subseteq C$. If $x \in C$ but $\mathrm{Adj}(x) \nsubseteq C$, then x is a <u>boundary node</u> of C.

As we have described before, nodes associated with a mesh element correspond to a clique in the corresponding graph. Generally, interior nodes of a clique in G_i correspond to interior nodes of an element in M_i. However, there are exceptions. The nodes on the periphery of an element which forms part of ∂R

are interior nodes of the corresponding graph clique. Also, when all the nodes form a clique, in the element model they could be interior nodes and/or boundary nodes. Thus although the element model is very helpful in visualizing the changing structure of the matrix during the decomposition, its meaning in terms of the corresponding graph must be interpreted carefully near ∂R and for the last clique of nodes.

In what follows, the sequence G_i, $i = 0,1,\ldots,N-1$ will refer to the graph sequence generated by the minimum degree algorithm.

Lemma 3.1

Let C be a clique in $G(X,E)$, let x be an interior node of C, and let y be a boundary node of C. Then $\deg(x) < \deg(y)$.

The proof is trivial and is omitted.

Lemma 3.2

Let C be a clique in $G_i = (X_i,E_i)$, $0 \le i < N$, and let x be an interior node of C. Then if x is eliminated, the degree of all nodes in C is reduced by one, and the degrees of all other nodes in G_i remain the same.

Proof Since $\mathrm{Adj}(x) \subset C$, the elimination of x cannot affect nodes $y \notin C$. Thus $\deg(y)$, $y \in X_{i+1} \backslash C$, is the same as it was in G_i. On the other hand, since $x \in C$ and C is a clique, the elimination of x causes no fill. Thus, E_{i+1} is obtained from E_i by deleting $\mathrm{Inc}(x)$ from E_i, thereby reducing the degree of all nodes in $C \backslash \{x\}$ by one. $\square$

Theorem 3.3

Let C be a clique in $G_i = (X_i,E_i)$, $0 \le i < N$, and let Q_i be the set of all interior nodes of C. Then if the minimum degree algorithm chooses $x \in Q_i$, at the i-th step, then it numbers the remaining $|Q_i|-1$ nodes of Q_i next.

Proof Let $\deg(x) = d$, and note that $\deg(y) \geq d$, $y \notin Q_i$, with $\deg(y) > d$ if $y \in C \backslash Q_i$. By Lemma 3.4, after x is eliminated, the degree of all nodes remaining in C will be reduced by one, and nodes not in C will have their degrees unchanged. Thus, if $Q_{i+1} = Q_i \backslash \{x\} \neq \phi$, then the node of minimum degree in G_{i+1} is in Q_{i+1}. Repeatedly using Lemma 3.2, we conclude that the minimum degree algorithm will choose nodes in Q_i until it is exhausted. □

Corollary 3.4

In terms of the mesh model, Lemma 3.2 and Theorem 3.3 imply that nodes in the interior of an element, or those on the boundary of an element which forms part of ∂R, will all be eliminated before other nodes associated with that element.

Theorem 3.5

Let C_1 and C_2 be two cliques in G_i, $0 \leq i < N$, with $K_i = C_1 \cap C_2 \neq \phi$, $C_1 \not\subset C_2$, and $C_2 \not\subset C_1$. Let

$$\bar{K}_i = \{y | y \in K_i \text{ and } \mathrm{Adj}(y) \subset C_1 \cup C_2\}.$$

Then if the minimum degree algorithm chooses $x \in \bar{K}_i$ at the i-th step, then it numbers the remaining nodes of $\bar{K}_{i+1} = \bar{K}_i \backslash \{x\}$ next, in arbitrary order.

Proof First, note that C_1 and C_2 cannot have any interior nodes, since otherwise the minimum degree algorithm would not choose a node from $\bar{K}_i$. Thus, for all $y \in \bar{K}_i$, $\deg(y) = d = |C_1 \cup C_2| - 1$. Also, note that if $y \in K_i \backslash \bar{K}_i$, then $\deg(y) > d$, because it is connected to all other nodes in $C_1 \cup C_2$, and at least one other node. (Otherwise, it would be in $\bar{K}_i$.) Finally, if $y \in X_i \backslash K_i$, then $\deg(y) \geq d$, because otherwise the minimum degree algorithm would not choose x. We now want to show that after x is eliminated, yielding

$G_{i+1} = (X_{i+1}, E_{i+1})$, that $\deg(y) = d-1$ if $y \in \bar{K}_{i+1}$, and $\deg(y) \geq d$ for $y \in X_{i+1} \backslash \bar{K}_{i+1}$.

First, if $\bar{K}_{i+1} = \phi$, there is nothing to prove, so suppose $\bar{K}_{i+1} \neq \phi$. Now the elimination of x renders $(C_1 \cup C_2)\backslash\{x\}$ a clique, but since $\mathrm{Adj}(x) \subset C_1 \cup C_2$, nodes $y \in X_i \backslash (C_1 \cup C_2)$ are not affected by the elimination of x, so $\deg(y) \geq d$ as before. Suppose $y \in C_1 \backslash C_2$, and let its degree in G_i be $p \geq d$. Then after elimination it is $p + |C_2 \backslash C_1| - 1 \geq d$, since $|C_2 \backslash C_1| \neq \phi$. Similarly, after elimination of x, $\deg(y) \geq d$ for $y \in C_2 \backslash C_1$. Now consider $y \in K_{i+1} = K_i \backslash \{x\}$. Before elimination of x, y is connected to all nodes in $(C_1 \cup C_2)\backslash\{y\}$, since C_1 and C_2 are cliques. Since the elimination of x only involves nodes in $C_1 \cup C_2$, y cannot be connected to any new variables. Moreover, after elimination of x, y is no longer connected to x, so $\deg(y)$ is reduced by one for $y \in K_{i+1}$. Thus, in G_{i+1}, $\deg(y) = d-1$ for $y \in \bar{K}_{i+1}$ and $\deg(y) \geq d$ for $y \in K_{i+1} \backslash \bar{K}_{i+1}$.

The minimum degree algorithm will now choose a node in $\bar{K}_{i+1}$, and by Theorem 3.3, it will continue to choose nodes from $\bar{K}_{i+1}$ until it is exhausted. $\square$

<u>Corollary 3.6</u>

Let C_k, $k = 1,2,\ldots,r$ be cliques in G_i, $i \in \{0,1,\ldots,N-1\}$ $K_i = \bigcap_{k=1}^{r} C_k \neq \phi$ and $C_\ell \cap (X_i \backslash (\bigcup_{k \neq \ell} C_k)) \neq \phi$. Let

$$\bar{K}_i = \{y \mid y \in K_i \text{ and } \mathrm{Adj}(y) \subset (\bigcup_{k=1}^{r} C_k)\}.$$

Then if the minimum degree algorithm chooses $x \in \bar{K}_i$ at the i-th step, then it numbers the remaining nodes of $\bar{K}_{i+1} = \bar{K}_i \backslash \{x\}$ next, in arbitrary order. The proof is similar to that of Theorem 3.5 and is omitted.

Theorems 3.3 and 3.5, and Corollary 3.6 have important practical implications. It is easy to recognize when their hypotheses apply, and they allow us to immediately number <u>sets</u> of nodes by doing only <u>one</u> minimum degree search.

Thus, after each minimum degree search, a <u>set</u> of $r \geq 1$ nodes will be numbered, inducing a <u>partitioning</u> of X. We will denote this partitioning by $P = \{P_1, P_2, P_3, \ldots, P_p\}$, where $\bigcup_{i=1}^{p} P_i = X$. Figures 3.1a,b contain an L-shaped triangular mesh M_0 together with a few of the meshes M_i, $0 \leq i \leq N$ generated by the minimum degree algorithm. Figure 3.2 shows the matrix structure of $L+L^T$ corresponding to the ordering, with the column partitioning P indicated by the vertical lines.

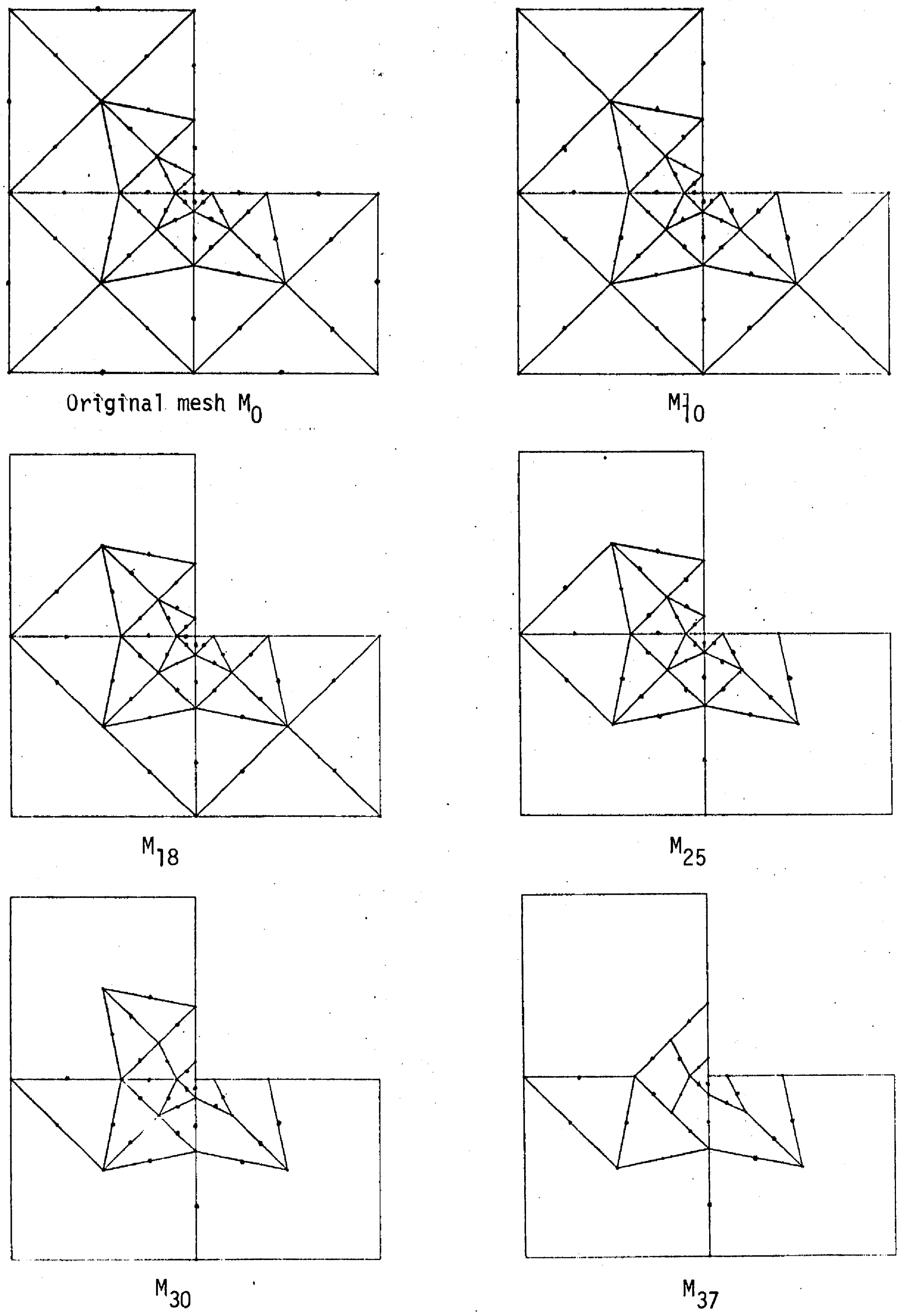


Figure 3.1a A selection of meshes from the sequence M_k, $k = 0,2,\ldots,N$

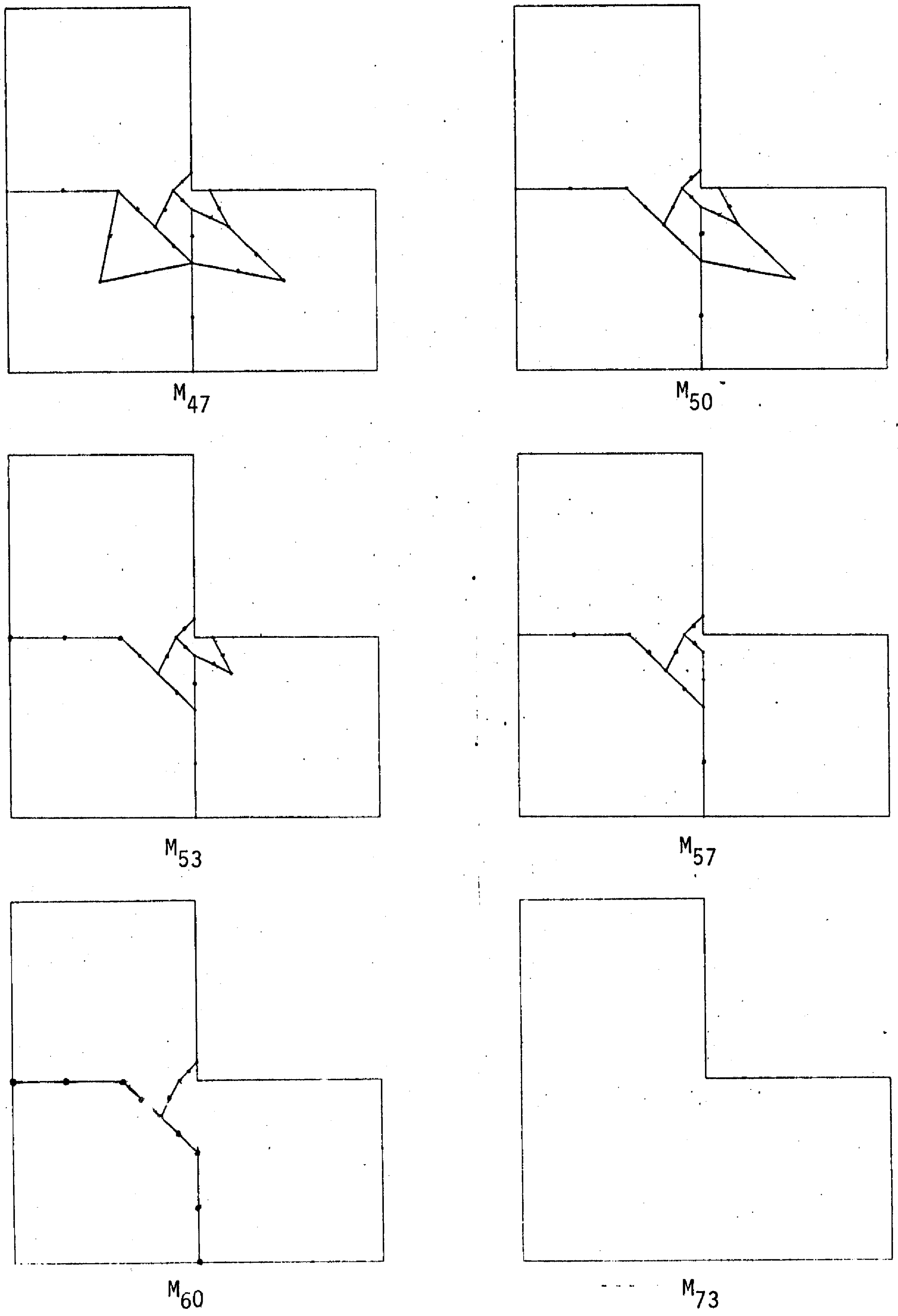


Figure 3.1b A selection of meshes from the sequence M_k, $k = 0,1,\ldots,N$

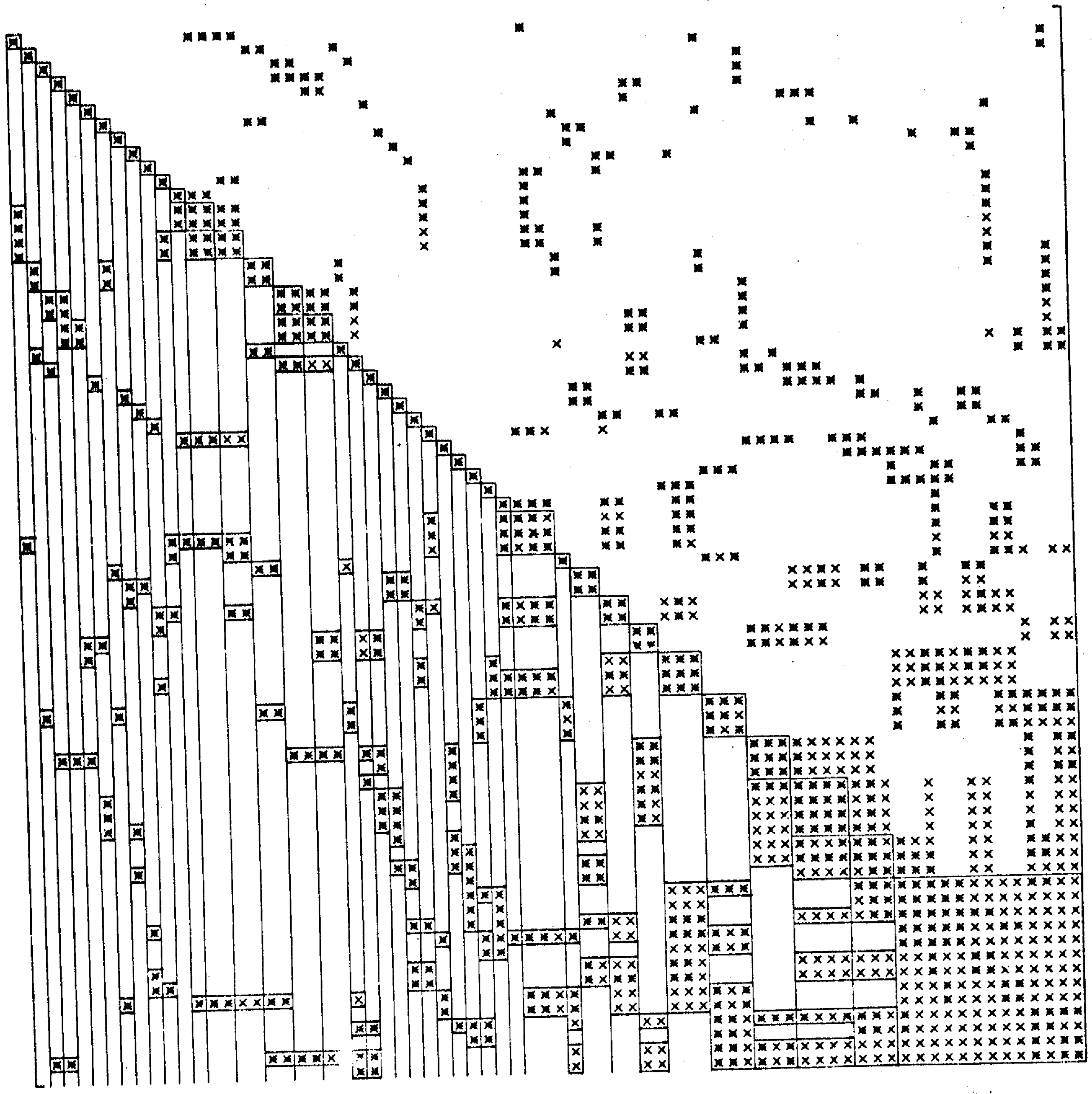


Figure 3.2 Structure of $L+L^T$ corresponding to the ordering produced for the mesh of Figure 3.1. The partitioning P is indicated by vertical lines. The character * represents nonzeros of L which correspond to nonzeros in A, while X represents fill.

§4 A Refinement of the Minimum Degree Ordering

In the last section we saw how the minimum degree algorithm naturally induces a partitioning of the matrix A (and therefore also of L). Consider each "block column" of L, determined by the members of the partitioning $\mathcal{P}$. Within each such block column the rows are either empty or full, and we can partition the rows according to contiguous sets of non-null rows, as indicated in Figure 3.2.

Basically, the storage scheme we use in our linear equation solver is one which stores the dense submatrices determined by this row-within-(block) column partitioning. Now each non-null block incurs a certain amount of storage overhead, so we would like the number of these blocks to be as small as possible. The purpose of this section is to describe a way of reordering the members of each partition member $\mathcal{P}_i$ so as to reduce the number of non-null blocks in this row-within-column partitioning.

Our reordering scheme is most easily motivated using the mesh model of elimination we introduced in section 2. First, note that most of the partition members (except for some of the initial ones) will correspond to nodes lying on a side of an element in some mesh M_k, $0 \le k \le N$. It is clear that the order in which these nodes are numbered is irrelevant as far as fill or operation counts are concerned, and their relative order is not specified by the minimum degree algorithm. Thus, we are free to choose the ordering within the partitions to reduce the number of individual blocks of L we must store.

How do we achieve this? Consider the schematic drawing in Figure 4.1, indicating a subsequence of meshes taken from the sequence M_k, $k = 0,1,\ldots,N$.

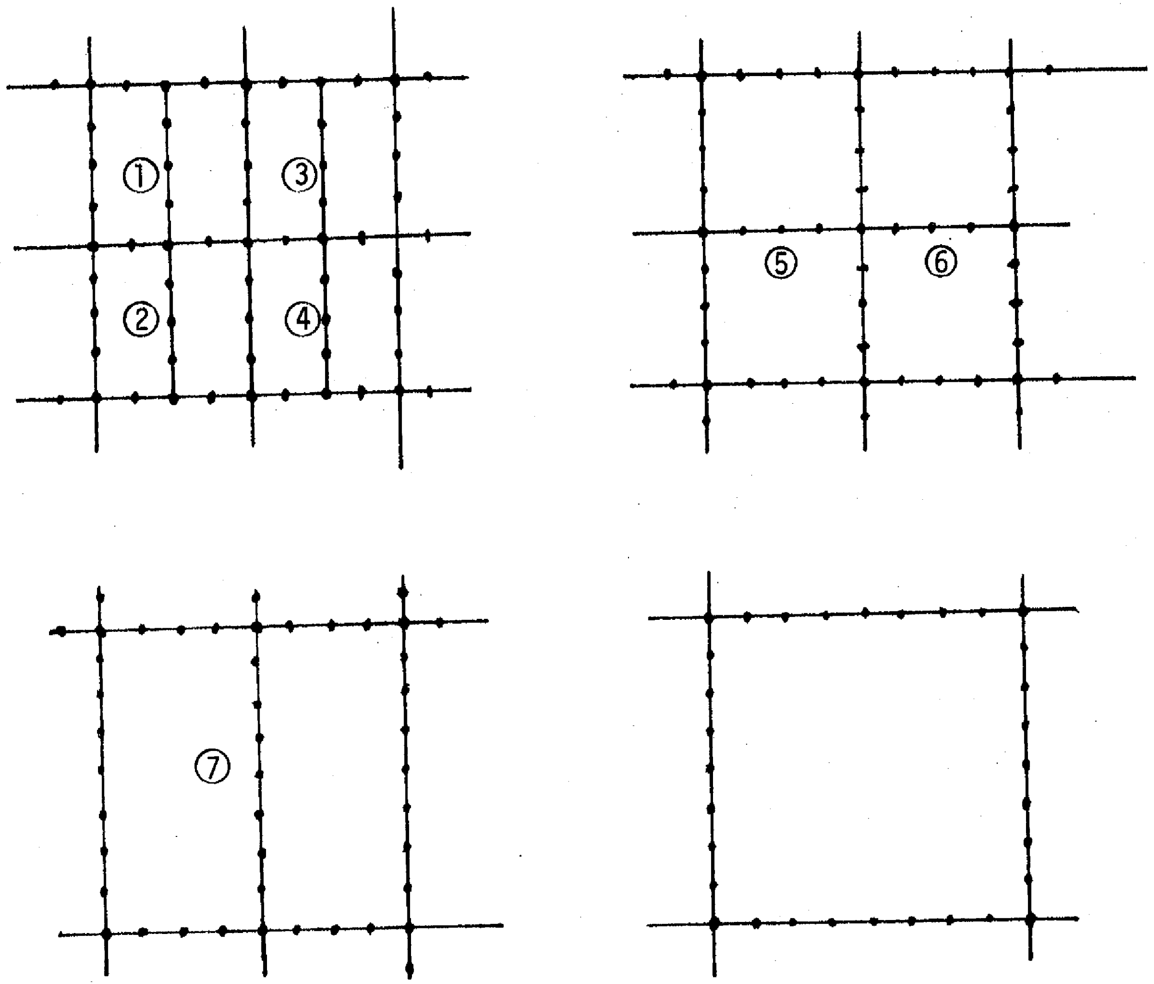


Figure 4.1 A subsequence of meshes from the sequence M_k, $k = 0,1,\ldots,N$

From what we have established about the behavior of the minimum degree algorithm through Theorems 3.3 and 3.5, it is clear that the mesh line segments (element sides) denoted by ① through ⑦ correspond to dense diagonal blocks of L. Now consider the block column of L corresponding to ①. Obviously, block ① is connected to part of block ⑦, but in general the connection will not be reflected as a dense off-diagonal block of L, unless the nodes of ⑦ which are connected to ① are numbered consecutively. Similarly, the connections of block ② to block ⑦ will correspond to a dense block of L only if nodes of ⑦ which are connected to ② are numbered consecutively.

This motivates our reordering algorithm for each partition P_i. We reorder the nodes of each P_i in a way which corresponds to numbering nodes on an element side consecutively, beginning at one end. Figure 4.2 shows the structure of $L+L^T$ corresponding to the reordering obtained by applying our reordering scheme to the problem which produced the Figures 3.1 and 3.2. The reduction in the number of off-diagonal blocks is apparent, but it is not particularly impressive because the problem is quite small. For larger problems, however, the reduction is usually very substantial.

Now that we have established what we want done, how do we achieve it? Obviously, if $|P_i| \le 2$, there is nothing to do. For $|P_i| > 2$, the nodes in P_i will typically all lie on an element side in some mesh M_k, $0 \le k < N$, such as indicated in Figure 4.3. (For P_p, the last partition, the situation is somewhat more complicated, since often three element sides are involved, as in M_{60}, Figure 3.1-b. We consider this problem later.)

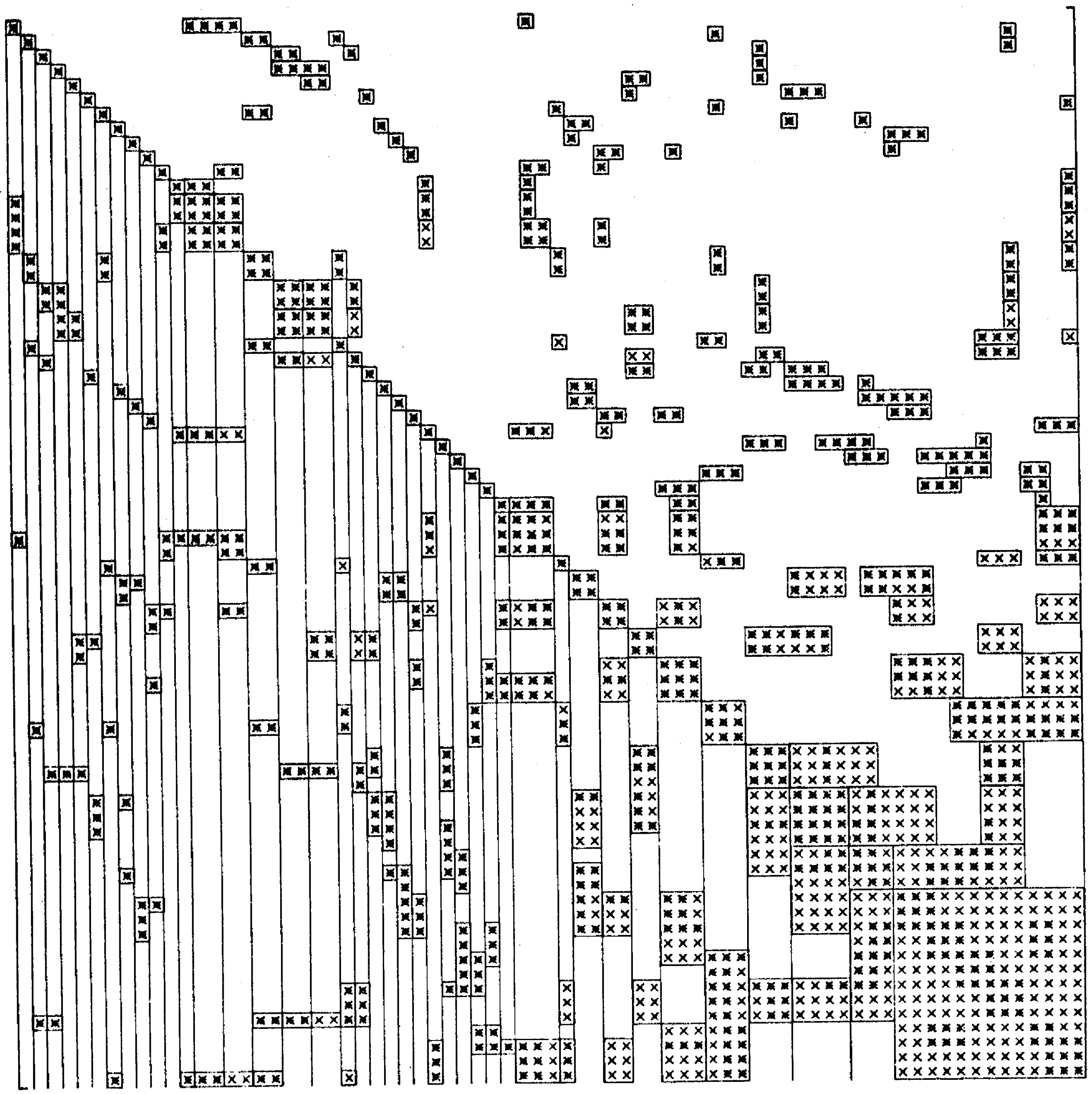


Figure 4.2 The structure of $L+L^T$ corresponding to the reordering of the problem which generated the matrix of Figure 3.2

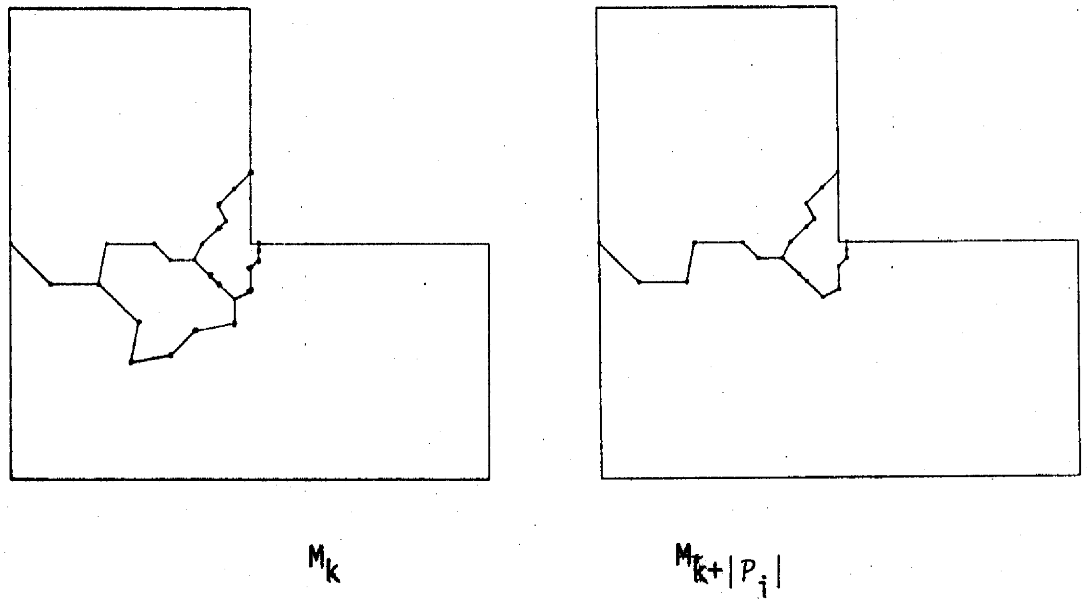


Figure 4.3 An example of a typical P_i, $1 \le i \le p$.

Let $G = (X,E)$ be the unlabeled graph corresponding to A, and let $\tilde{G} = (X,\tilde{E})$ be the subgraph of G obtained by interpreting the mesh M as a graph. Let G_{P_i} be the section graph $G(P_i)$ (see section 2). Now the element model makes it abundantly clear that in general G_{P_i} consists of <u>a single node</u>, or <u>a simple chain</u>, usually the latter. The graph G_{P_p} is a special case, typically consisting of three chains connected by virtue of a small shared clique or two chains connected by a cycle.

Our reordering algorithm is straightforward, and although we do not know if it is optimal, we do not know how it can be improved. It essentially involves the generation of two rooted spanning trees of G_{P_i}, the first of which is generated in such a way that the distance from any node x to the root r in the tree is the same as the distance from x to r in G_{P_i}. This can easily be done by generating the tree in a breadth-first manner, rather than in a depth-first manner [8].

Our reordering algorithm consists of two general stages, which we now informally describe. Here $r_1, r_2, \ldots, r_{|P_i|}$ are the consecutive integers assigned to the members of P_i by the minimum degree algorithm.

Stage 1:

Choose any node x in G_{P_i} and generate a breadth-first spanning tree T_1 for G_{P_i}, rooted at x. Any node y at the last level of the tree is chosen as a starting node for stage 2.

Stage 2:

In this stage R is a stack which is originally empty, and is only utilized if G_{P_i} is not a chain.

1) Label the node y provided by stage 1 as r_1.

2) For each $i = 2,3,\ldots,|P_i|$ do the following:

   a) If x_{r_i-1}, the last labelled node, has only one unlabelled node y adjacent to it, then label it r_i.

   b) If $Adj(x_{r_i-1})$ has more than one unlabelled node, of those not already in R, label one r_i and place the remainder on the stack R. If all nodes in $Adj(x_{r_i-1})$ are also in R, choose one of those and label it r_i.

   c) If the members of $Adj(x_{r_i-1})$ are all numbered, pop the stack R until an unlabelled node y is popped, and label it r_i.

Figure 4.4 illustrates the reordering algorithm. Phase 1 generates the tree T_1 rooted at x, and chooses the node y as the starting node for phase 2. Step a) of phase 2 is executed until node g is labelled. At the next step node h is placed in R and c is labeled. At the next step, the unlabeled nodes of Adj(c) are {h,x}, but since h is already in the stack, node x is labelled, and then step a) of phase 2 operates until node a is labeled. Since Adj(a) is all labelled, node h is obtained from R and labelled, followed by nodes i,j and k via steps a) and c) of phase 2.

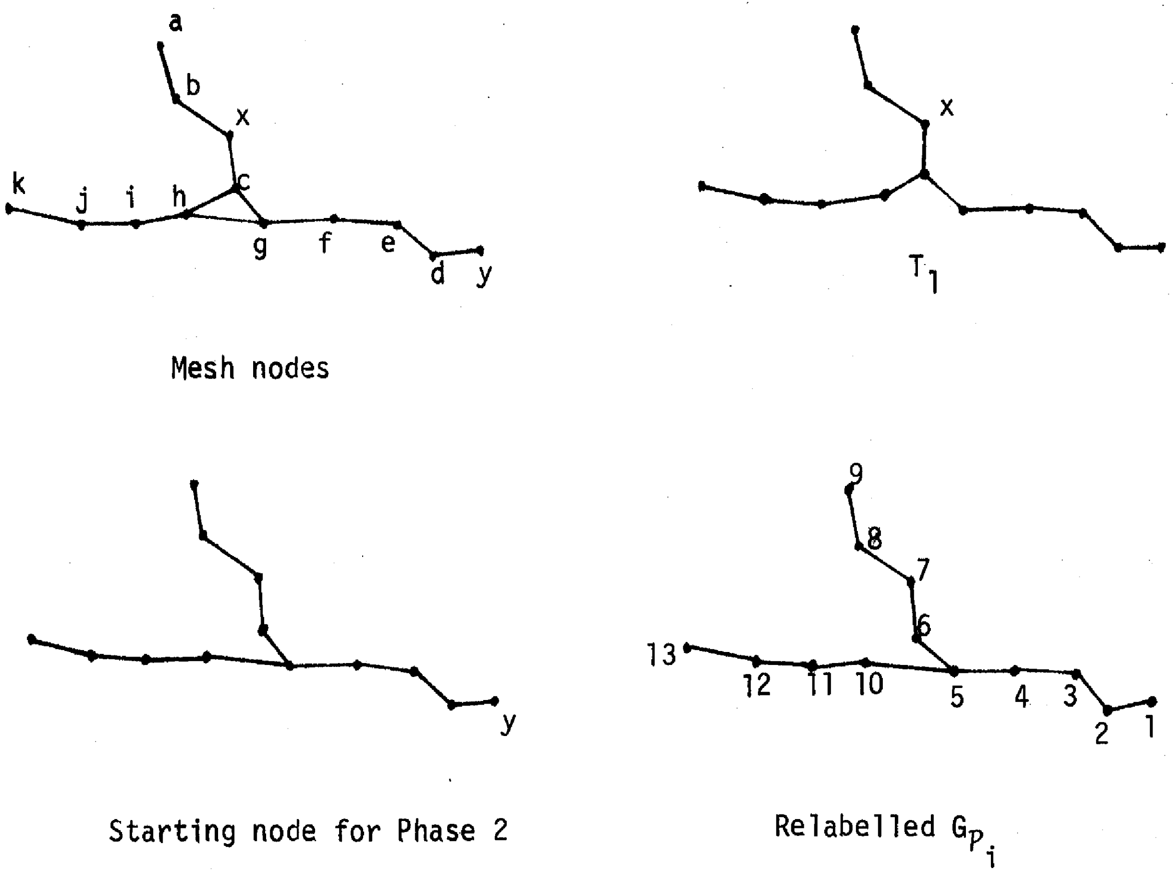


Figure 4.4 Relabelling of G_{P_i}

§5 Remarks on Implementation and Some Numerical Experiments

We saw in section 2 how cliques naturally arise during symmetric Gaussian elimination. In matrix problems associated with the use of the finite element method, cliques of size larger than one exist in G^A, and persist for some time during the elimination, typically growing in size by merging with other cliques before finally disappearing through elimination. Moreover, Theorem 3.5 operates for a considerable proportion of the total node numberings.

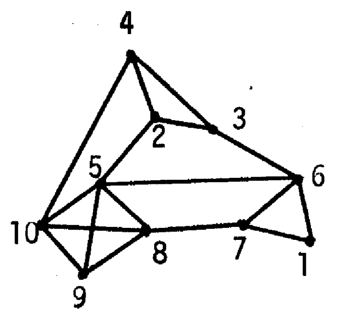
These observations make it natural to represent the elimination graph sequence through its clique structure, since elimination of variables typically leads to merging of two or more cliques into a new clique. Our approach then, is as follows. The graph $G_i = (X_i, E_i)$ is represented by the set of its cliques $\mathcal{C}_i = \{C_\ell^i\}$ along with a clique membership list for each node. An example appears in Figure 5.1.

Now our actual implementation does not represent the entire sequence of graphs G_i, $i = 0,1,2,\ldots,N-1$, during its execution. Only those graphs which would be obtained after each P_i is determined are actually created. That is, we repeatedly apply Theorems 3.3 and 3.5. The general step of our algorithm, described below, is executed p times, where $p = |\mathcal{P}|$ and $\mathcal{P} = \{P_1, P_2, \ldots, P_p\}$.

General Step r, r = 1,2,...,p

1) Find an unnumbered node x of minimum degree. If all nodes are numbered, stop.

2) Let $Q_r = \{\ell \mid x \in C_\ell^{r-1}\}$, and determine the set of nodes P_r in $\tilde{C}_r = \bigcup_{\ell \in Q_r} C_\ell^{r-1}$ which are connected only to nodes in $\tilde{C}_r$.



Graph G_0

Node	Clique Membership	Clique Set C_0
1	1	
2	2,5	C_1^0:{1,7,6}
3	2,6	C_2^0:{2,3,4}
4	2,4	C_3^0:{5,8,9,10}
5	3,5,7	C_4^0:{4,10}
6	1,6,7	C_5^0:{2,5}
7	1,8	C_6^0:{3,6}
8	3,8	C_7^0:{5,6}
9	3	C_8^0:{7,8}
10	3,4	

Figure 5.1 The graph G_0 represented by its clique set C_0 and clique membership list

3) Set $C_r = (C_{r-1} \setminus \bigcup_{\ell \in Q_r} \{C_\ell^{r-1}\}) \cup (\tilde{C}_r \setminus P_r)$.

4) Update the degrees of the nodes in $\tilde{C}_r \setminus P_r$ (the new clique), and their clique membership lists.

5) Increment r and go to step 1).

Our code consists of two phases, the first is simply the minimum degree algorithm, modified to exploit what we know about the behavior of the algorithm, as described by Theorems 3.3 and 3.5. The second phase performs the reordering of each partition member P_i as described in section 4. Although this splitting into two phases is not necessary (since each P_i could be reordered as it is generated), it was done to keep the code modular, and to ease maintenance and subsequent possible enhancements.

Our code accepts as initial input a collection of node sets corresponding to the <u>elements</u> (cliques) of the finite element mesh. This mesh changes as the algorithm proceeds, so its representation must be such that merging cliques (elements) is reasonably efficient and convenient. The data structure we used to represent the graphs is depicted in Figure 5.2. At any stage of the algorithm, the nodes of each clique along with some storage management information are stored in consecutive locations in a storage pool (POOL). A pointer array HDR of length $\leq$ NCLQS (the initial number of elements) is used to point to the locations of the elements in POOL. Finally, a rectangular array C is used to store the clique membership lists; row i of C contains pointers into HDR corresponding to cliques which have node i as a member.

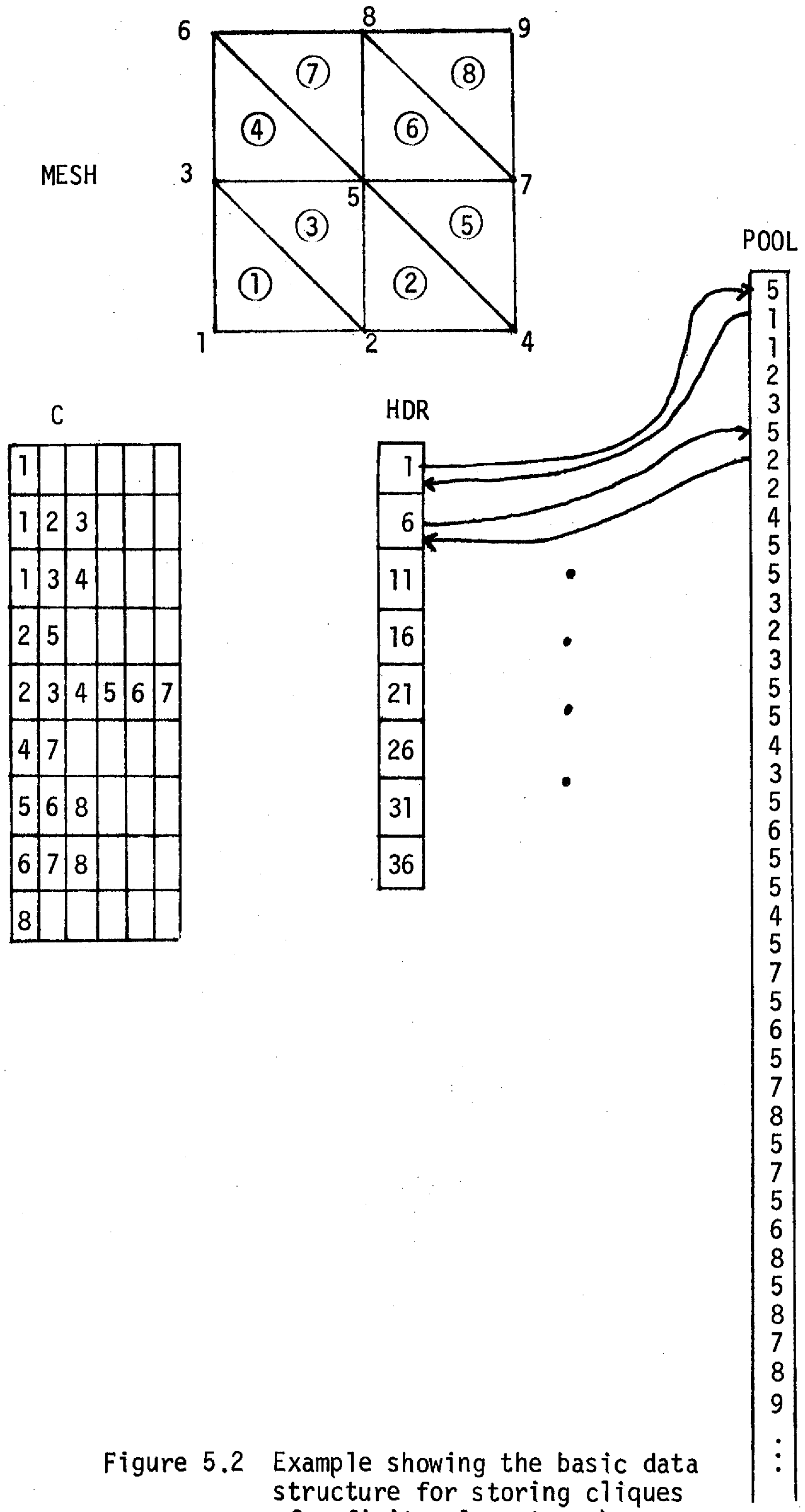


Figure 5.2 Example showing the basic data structure for storing cliques of a finite element mesh

Step 3) of the algorithm above obviously implies an updating operation of the arrays C, HDR and POOL to reflect the new clique structure of the graph which has seen some of its cliques coalesce into a single new one, along with the removal of some nodes. In general, the node-sets corresponding to each clique to be merged will be scattered throughout POOL, and none of them may occupy enough space so that the new clique to be created could overwrite them. To avoid excessive shuffling of data, we simply allocate space for the new clique from the last-used position in POOL, and mark the space occupied by the coalesced cliques as free. When space for a new element can no longer be found in POOL, a storage compaction is performed. See [8, pp.435-451] for a description of these standard storage management techniques.

Our first objective is to study the behavior of our ordering algorithm. We ran our code on N by N finite element matrix problems arising from $n \times n$ right-triangular meshes of the form shown in Figure 5.3. We ran our code for n = 5(5)35 to study the behavior of various quantities as a function of $N = (n+1)^2$.

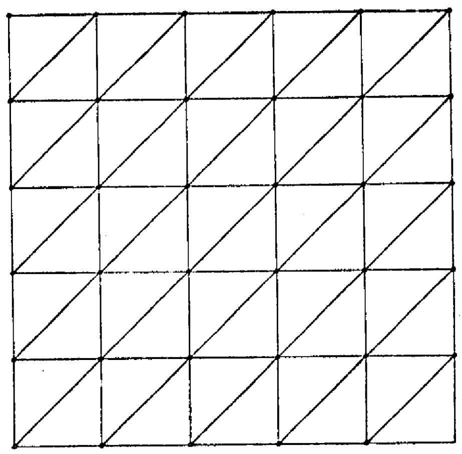


Figure 5.3 A 5 by 5 right-triangular mesh yielding N = 36

The results of our runs are summarized in Tables 5.1-5.3. The "overhead" column in Table 5.1 refers to the number of pointers etc. used by our data structure for L. In our implementation on an IBM 360/75, we used a 32 bit word for both pointers and data. On many machines with a larger wordlength, it would make sense to pack two or perhaps more pointers per word. Thus, in other implementations the overhead for our data structure compared to the storage required for the actual components of L would be much less that appears in Table 5.1.

The overhead and primary column entries in Table 5.1 do not quite add up to the corresponding entry in the total column because we included various other auxiliary vectors and space for the right side b in the total storage count.

The following observations are apparent from the data in Tables 5.1-5.3.

1) The overhead storage appears to grow linearly with N, and the total storage requirement for all data associated with solving the matrix problem grows as N log N. This has two important practical implications. First, it implies that (overhead storage)/(total storage) $\to 0$ as $N \to \infty$, in contrast to most sparse matrix solvers for which this ratio is some constant α, usually with $\alpha > 1$. The second implication is perhaps even more important. It is well known that for this problem, the use of bandmatrix methods (i.e., a banded ordering) implies that total storage requirements grow as $N^{3/2}$. Indeed, the best ordering known to the authors (the so-called diagonal dissection ordering [3]) would imply a storage requirement of O(N log N).

TABLE 5.1

Storage statistics for the ordering produced by the minimum degree algorithm followed by the improvement described in section 4

n	N	Overhead	Primary	Total	$\frac{\text{Overhead}}{\text{Total}}$	$\frac{\text{Overhead}}{\text{N}}$	$\frac{\text{Total}}{\text{N log N}}$
5	36	316	185	537	.59	8.78	4.16
10	121	1174	1039	2334	.50	9.70	4.02
15	256	2457	2899	5612	.44	9.60	3.95
20	441	4195	5959	10595	.40	9.51	3.95
25	676	6501	10092	17269	.38	9.62	3.92
30	961	9153	17190	27304	.34	9.52	4.14
35	1296	12425	24252	37973	.33	9.59	4.09

TABLE 5.2

Execution Time in Seconds on an IBM 360/75 for the ordering algorithm described in Sections 3 and 4

n	N	Time for Phase 1	Time for Phase 2	Total time	$\frac{\text{Time}}{N \log N}$	Fact. mult.	Fact. time	Back-solve mult.	Back-solve time	Total Soln. time	$\frac{\text{Soln. time}}{N\sqrt{N}}$[†]	Ordering plus solution time
5	36	.35	.20	.55	.0043	578	.04	370	.03	.07	.324	.62
10	121	1.15	.50	1.65	.0028	5739	.22	2078	.09	.31	.233	1.96
15	256	2.37	1.08	3.45	.0024	21919	.61	5798	.20	.81	.198	4.26
20	441	4.23	1.84	6.07	.0023	56501	1.30	11918	.34	1.64	.177	7.71
25	676	6.57	2.84	9.41	.0021	107474	2.36	20184	.54	2.90	.165	12.31
30	961	9.69	4.01	13.70	.0021	242548	4.36	34380	.77	5.13	.172	18.83
35	1296	13.23	5.39	18.62	.0020	360937	6.14	48504	1.06	7.20	.154	25.82

† Scaled by 10^{-3}

TABLE 5.3

Statistics on P as a function of N for the ordering produced by the algorithm described in Sections 3 and 4

n	N	No. of off-diagonal blocks	$\lvert P\rvert$	$\frac{\text{off-diagonal blocks}}{\lvert P\rvert}$	$\frac{\lvert P\rvert}{N}$
5	36	59	28	2.1	.777
10	121	242	82	3.0	.678
15	256	510	156	3.3	.609
20	441	871	256	3.4	.580
25	676	1362	384	3.5	.568
30	961	1912	531	3.6	.553
35	1296	2608	710	3.7	.548

2) The entries in Table 5.2 suggest rather strongly that the execution time of our ordering code for this problem grows no faster than N log N. Similar experiments with other mesh problems demonstrate the same behavior.

3) Table 5.3 contains some interesting statistics about P, the partitioning induced by the repeated application of Theorems 3.3 and 3.5 in our ordering algorithm. It appears that $|P|$ is approaching a "limit" near N/2, and that the number of off-diagonal blocks in each "block column" is approaching about 4. Again, similar experiments with other mesh problems indicate that this behavior is not unique to our test problem.

We now turn to a comparison of our ordering algorithm with an alternative. For comparison, we used the recently developed ordering algorithm due to Gibbs et al. [5], along with a solver which exploits the variation in the bandwidth of the matrix, as suggested by Jennings [7]. In our tables, we denote results for this ordering-solver combination by BAND, as opposed to the results of our ordering algorithm/linear equation solver package, which we denote by BMD (block-minimum-degree).

From Table 5.4 the total storage for the solution of the test problem, using the band ordering, appears to grow as $O(N\sqrt{N})$, as expected for these meshes. The storage overhead is only $\approx N$. However, in contrast, the total storage used for the solver which uses the BMD ordering appears to grow only as $O(N \ell n N)$, despite the larger overhead. Extrapolating the results of the tables suggests that the storage for the BMD ordering will be less than band storage for $N \gtrsim 2000$, with the saving reaching 50 percent by the time N is around 15000.

TABLE 5.4

Storage Statistics for Band Ordering

n	N	Overhead	Primary	Total	$\frac{\text{Overhead}}{\text{Total}}$	$\frac{\text{Overhead}}{N}$	$\frac{\text{Total}}{N\sqrt{N}}$
5	36	39	191	267	.15	1.08	1.236
10	121	124	1056	1302	.10	1.02	.978
15	256	259	3096	3612	.07	1.01	.882
20	441	444	6811	7697	.06	1.01	.831
25	676	679	12701	14057	.05	1.00	.800
30	961	964	21266	23192	.04	1.00	.778
35	1296	1299	33006	35602	.04	1.00	.763

It should be noted here that the BMD ordering algorithm is implemented in $\approx 30N$ storage (i.e., linear in N). This is important because for $N \gtrsim 1000$ the ordering can be done in the space used later for the factorization.

The entries in Table 5.5 suggest that the band ordering time is $O(N^{\rho})$, for $\rho \approx 1.05$, and the solution time is $O(N^2)$. A look at the operations for the factorization time for the BMD and band orderings in Tables 5.2 and 5.5 confirm that the apparent differences in factorization times are indeed due to differences in operation counts and not to program complexity.

Least squares approximations to the total execution times were found for the BMD and band algorithms using as basis functions the orders suggested in Tables 5.2 and 5.5. The results suggest that the BMD algorithm will execute faster than the band algorithm for $N \geq 20{,}000$. For $N \approx 60{,}000$ the results imply that the BMD algorithm is twice as fast. Thus, our ordering/solution package is unlikely to be attractive as a one shot scheme.

However, in many situations involving mildly nonlinear and/or time dependent problems, many matrix problems having the same structure, or even the same coefficient matrix, must be solved. In these situations it makes sense to ignore ordering time and compare the methods with respect to factorization time or solution time. If we do this we see that the cross-over point for factorization time is when $N \approx 1500$, and for solution time the cross-over point is about $N \approx 2200$.

TABLE 5.5

Execution Time in Seconds on an IBM 360/75
for Band Ordering

n	N	Ordering time	Ordering time / $N^{1.05}$	Fact. mult.	Fact. time	Back-solve mult.	Back-solve time	Total soln. time	Soln. time / N^2 †	Ordering plus solution time
5	36	.03	.697	610	.02	382	.01	.03	.0231	.06
10	121	.11	.715	5445	.09	2112	.03	.12	.082	.23
15	256	.23	.681	21880	.29	6192	.06	.35	.053	.58
20	441	.41	.686	61040	.75	13622	.13	.88	.045	1.29
25	676	.66	.705	137800	1.48	25402	.22	1.70	.037	2.36
30	961	.93	.686	270785	2.73	42532	.38	3.11	.034	4.04
35	1296	1.31	.706	482370	4.64	66012	.54	5.18	.031	6.49

† Scaled by 10^{-3}

TABLE 5.6

Ratio of BMD/BAND for Various Quantities

n	N	Total time	Total Store	Fact. time	Soln. time
5	36	10.33	2.01	2.00	3.00
10	121	8.52	1.79	2.44	3.00
15	256	7.34	1.55	2.10	3.33
20	441	5.98	1.38	1.73	2.62
25	676	5.22	1.23	1.59	2.45
30	961	4.66	1.18	1.60	2.03
35	1296	3.98	1.07	1.32	1.96

Concluding Remarks

In terms of execution time, our numerical experiments suggest that our ordering algorith/solution package is attractive for "one-of" problems only if N is extremely large. However, in terms of storage requirements, and if only factorization and solution time is considered, our scheme looks attractive compared to band oriented schemes if N is larger than a few thousand.

Our experiments suggest that for our class of finite element problems, the ordering code executes in $O(N \log N)$ time, and the ordering produced for this problem yields storage and operation counts of $O(N \log N)$ and $O(N^{3/2})$ respectively. For the square mesh problem, these counts are known to be optimal, in the order of magnitude sense [4]. It is interesting to observe that the partitioning produced by the minimum degree algorithm prescribes dissecting sets similar in flavor to those for dissection orderings [3,4]. This leads us to speculate whether the minimum degree algorithm generates asymptotically optimal orderings for general finite element matrix problems. Further research in this area seems appropriate.

§7 References

[1] Araldsen, P.O., "The application of the superelement method in analysis and design of ship structures and machinery components", National Symposium on Computerized Structural Analysis and Design, George Washington University, March 1972.

[2] C. Berge, The Theory of Graphs and its Applications, John Wiley & Sons Inc., New York, 1962.

[3] Garrett Birkhoff and Alan George, "Elimination by nested dissection" in Complexity of Sequential and Parallel Algorithms (J.F. Traub, editor), Academic Press, New York, 1973, pp.221-269.

[4] Alan George, "Nested dissection of a regular finite element mesh", SIAM J. Numer. Anal., 10(1973), pp.345-363.

[5] N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix", SIAM J. Numer. Anal., to appear.

[6] M.J.L. Hussey, R.W. Thatcher and M.J.M. Bernal, "Construction and use of finite elements", JIMA, 6(1970), pp.262-283.

[7] A. Jennings, "A compact storage scheme for the solution of symmetric linear simultaneous equations", Computer J. 9, 281-285.(1966).

[8] D.E. Knuth, The Art of Computer Programming, vol.I (Fundamental Algorithms), Addison Wesley, 1968.

[9] S.V. Parter, "The use of linear graphs in Gauss elimination", SIAM Rev., 3(1961), pp.364-369.

[10] D.J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations", in Graph Theory and Computing, edited by R.C. Read, Academic Press, New York, 1972.

[11] B. Speelpenning, "The generalized element method", unpublished manuscript.

[12] K.L. Stewart and J. Baty, "Dissection of structures", J. Struct. Div., ASCE, Proc. paper No.4665, (1966), pp.75-88

[13] James H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.

[14] O.C. Zienkiewicz, The Finite Element Method in Engineering Science, McGraw Hill, London, 1970.