This work was presented to the faculty of the Graduate School of Yale University in candidacy for the degree of Doctor of Philosophy. The author is presently in the Department of Computer Science at the University of Illinois, Urbana, Illinois 61801.

On the Efficient Solution of Sparse Systems of Linear and Nonlinear Equations

Andrew Harry Sherman

Research Report #46

December 1975

This work was partially supported by ONR grant N0014-67-A-0097-0016 NSF grant GJ-43157, and the Sloan Foundation.

ABSTRACT

ON THE EFFICIENT SOLUTION OF SPARSE SYSTEMS OF LINEAR AND NONLINEAR EQUATIONS

Andrew Harry Sherman
Yale University, 1975

This dissertation is primarily concerned with the efficient solution by Gaussian elimination of large, sparse, symmetric, positive definite systems of linear equations. We develop and analyze the costs of an efficient implementation of sparse symmetric Gaussian elimination, investigate ways of reducing the storage requirements of direct methods, and consider the efficient use of direct methods for sequences of sparse linear systems, specifically as they might arise in the solution of nonlinear equations.

Our implementation of sparse symmetric Gaussian elimination differs from previous implementations in two main respects. First, we employ a new storage scheme for sparse matrices which takes full advantage of both the available symmetry in the matrices and the regular way in which the zeroes fill in, or become nonzero, during the computation. The scheme allows the use of symbolic factorization techniques due to Chang, and we introduce an algorithm for symbolic factorization which is significantly more efficient than previous algorithms.

A major problem with direct methods is their large storage requirement. As a means of mitigating this difficulty, we present a way to trade off storage for work so as to obtain minimal storage Gaussian elimination methods. In particular, we develop two minimal storage methods which are well-suited to the numerical solution of partial differential equations on rectangular regions in the plane using either finite difference or finite element techniques.

Finally, as an application of our results for sparse linear systems, we introduce the Newton-Richardson methods for the solution of sparse systems of nonlinear equations. These methods are a class of Newton-iterative methods which retain the local convergence properties of Newton's method. They are computationally more efficient than the straightforward implementation of Newton's method when the Jacobian matrices are sparse, and they may offer substantial savings in practice. We also consider more general Newton-iterative methods and obtain a result which shows that it is possible to obtain local quadratic convergence with such methods.

PREFACE

The efficient solution of sparse systems of equations is a topic which intersects many of the subdisciplines of Computer Science. Certainly it is of great importance to the scientists, engineers, and numerical analysts who deal frequently with sparse systems. However, the goals and techniques of the research on the problem also make it of interest to others.

For instance, the analysis of the cost of solving sparse linear systems is very combinatorial in flavor, incorporating theoretical aspects of graph theory and concrete computational complexity. Moreover, since one of the practical goals of the research is to develop high quality software, one must be concerned with the aspects of computer architecture, operating systems, programming languages, and software design which affect the performance and generality of numerical software.

In this dissertation we try to cover both theoretical and practical issues and to treat the subject as one in Computer Science as a whole, rather than as one solely in numerical analysis, systems design, computational complexity, or any other single subdiscipline. Naturally, certain chapters will achieve this goal better than others, but the basic premise which underlies all of the research presented here is that the solution of difficult problems in numerical computation requires not only the application of the specific techniques of any single subdiscipline, but also consideration of the ways in which the subdisciplines interact with and reinforce each other.

Many people have aided me with this research. Most of all, of course, I am indebted to my advisers, Professors Martin H. Schultz and Stanley C. Eisenstat, for suggesting this subject area to me and for providing the necessary stimulus when my pace slackened over the past several years. Their insights and comments have had an immeasurable effect on the quality of the research and the form of its presentation here. I would also like to acknowledge Professor Donald J. Rose of Harvard University, with whom I have had many helpful discussions, Gail Beyer-Ohlson, who has done such an excellent job of typing this dissertation, and Robert Schreiber, who aided in proofreading.

I am grateful to a number of organizations for their financial support of my work. Yale University, the Sloan Foundation, and the Exxon Foundation have supported my graduate education with a combination of fellowships and assistantships. I have received summer support at Yale from the Sloan Foundation, the Office of Naval Research under Grant N0014-67-A-0097-0016, and the National Science Foundation under Grant GJ-43157. In addition, part of my research was performed while I was a Resident Student Associate at the Argonne National Laboratory in June 1974, part was performed under contract to the Chevron Oil Field Research Company, and part was performed while I acted as a consultant to the Computation Department of the Lawrence Livermore Laboratory during the 1974-1975 academic year.

Finally, I wish to thank the most important person in my life, my wife Martha. Her great devotion and support have kept me going during the past two years, and I dedicate this work to her.

ORGANIZATIONAL NOTE

This dissertation is divided in eight chapters numbered with Roman numerals. Except for Chapter I, each chapter is further divided into sections numbered with Arabic numerals, and within each section, the results (i.e. lemmas, theorems, and corollaries), algorithms, equations, figures, and tables are numbered sequentially. A "decimal" notation has been used to facilitate reference within the dissertation. Every section implicitly has a fully qualified number of the form "c.s", and every numbered item implicitly has a fully qualified number of the form "c.s.n", where "c" is a chapter number, "s" is a section number, and "n" is an item number within a section. Thus, for example, Section III.2 is the second section of the third chapter, and Lemma III.2.2 and Equation III.2.2 are, respectively, the second result and the second numbered equation of that section. For convenience, we will only use the chapter number in references to chapters other than the current chapter.

TABLE OF CONTENTS

		Page
PREFACE		iii
ORGANIZATIONAL NOTE		vi
TABLE OF CONTENTS		vii
LIST OF TABLES		ix
LIST OF ALGORITHMS		x
Chapter		
I	INTRODUCTION	1
II	PRELIMINARIES	9
	II.1 Mathematical Notation	9
	II.2 Elementary Graph Theory	11
	II.3 Model Problems	14
III	SYMMETRIC GAUSSIAN ELIMINATION	21
	III.1 Introduction	21
	III.2 Algorithms for U^TDU Factorization	22
	III.3 Analysis of Dense U^TDU Factorization	26
	III.4 A Graph Model for Sparse U^TDU Factorization	30
	III.5 An Element Model for Sparse Symmetric Gaussian Elimination	36
IV	ELIMINATION ORDERINGS	49
	IV.1 Introduction	49
	IV.2 Local Cost Minimization Algorithms	52
	IV.3 Nested Dissection Orderings	55
	IV.4 Orderings for the Model Problems	75
V	IMPLEMENTATION OF SPARSE SYMMETRIC FACTORIZATION	84
	V.1 Introduction	84
	V.2 Symbolic Factorization	86
	V.3 Numeric Factorization	92
	V.4 Storage of Sparse Matrices	97
VI	PRACTICAL RESULTS	105
	VI.1 Introduction	105
	VI.2 Band and Envelope Methods	106
	VI.3 Results for the Model Problems	111
	VI.4 Results for the Biharmonic Equation	115
	VI.5 Guidelines for Sparse Symmetric Linear Systems	118
VII	MINIMAL STORAGE METHODS	123
	VII.1 Introduction	123
	VII.2 Auxiliary Storage Band Methods	126
	VII.3 Minimal Storage Band Methods	129
	VII.4 Minimal Storage Band Methods for the Model Problems	134
	VII.5 Minimal Storage Sparse Methods for the Model Problems	142
VIII	SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS	153
	VIII.1 Introduction	153
	VIII.2 A Model Problem	157
	VIII.3 Newton-Richardson Methods for the Model Problem	159
	VIII.4 Convergence Results	163
	VIII.5 Ramifications of the Theory for the Model Problem	169
	VIII.6 Numerical Results for the Model Problem	174
	VIII.7 Numerical Results for the Minimal Surface Equation	179
	VIII.8 Proofs of Convergence Results	184
CONCLUDING REMARKS		189
REFERENCES		190

LIST OF TABLES

Table		Page
IV.4.1	Comparison of Minimum Degree and Diagonal Nested Dissection for the Five-Point Model Problem	77
IV.4.2	Comparison of Minimum Degree and Nested Dissection for the Nine-Point Model Problem	80
VI.3.1	Results for the Five-Point Model Problem	112
VI.3.2	Results for the Nine-Point Model Problem	113
VI.4.1	Results for the Biharmonic Model Problem	117
VI.5.1	Recommended Methods for Sparse Symmetric Linear Systems	119
VII.5.1	Comparison of In-core Gaussian Elimination Methods for the Nine-Point Model Problem	152
VIII.6.1	Results for Newton-Richardson Methods Reducing the Residual by a Factor of 10^{-11}	176
VIII.6.2	Results for Newton-Richardson Methods Reducing the Error by a Factor of h^2	177
VIII.6.3	Results for Nonlinear Conjugate Gradient	178
VIII.7.1	Results for Newton-Richardson Methods	182
VIII.7.2	Results for Block Nonlinear SOR	183

LIST OF ALGORITHMS

Algorithm		Page
III.2.1	Row-Oriented Dense U^TDU Factorization	24
III.2.2	Row-Oriented Sparse U^TDU Factorization (with zero testing)	25
III.2.3	Row-Oriented Sparse U^TDU Factorization (with pre-processing)	25
III.3.1	Outer-Product Form of Dense U^TDU Factorization	28
V.2.1	$O(\theta_A)$ Symbolic Factorization Algorithm	88
V.2.2	$O(\theta_S)$ Symbolic Factorization Algorithm	88
V.3.1	Numeric Factorization Algorithm	94

CHAPTER I: INTRODUCTION

In this dissertation we consider systems of linear equations

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

where $A = (a_{ij})$ is an $N \times N$ symmetric, positive definite, irreducible matrix.† Such systems occur throughout scientific computation, and often their solution comprises the bulk of the work required for the numerical calculations in which they arise.

When N is fairly small (e.g. $N < 50$) and A is dense (i.e. most $a_{ij} \neq 0$), one usually solves systems like (1.1) with some form of symmetric Gaussian elimination (cf. Forsythe and Moler [F1], Dahlquist and Björck [D1], p. 146). Equivalently, one can first factor A into the product

$$A = U^T D U \tag{1.2}$$

where U is unit upper triangular and D is a positive diagonal matrix and then forward- and back-solve to obtain $\underset{\sim}{x}$, i.e., successively solve the systems

† The matrix A is irreducible (cf. Blum [B4], p. 137) if there is no permutation matrix P such that PAP^T is of the form

$$PAP^T = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} .$$

$$U^T \underset{\sim}{y} = \underset{\sim}{b}, \quad D\,\underset{\sim}{z} = \underset{\sim}{y}, \quad \text{and} \quad U\,\underset{\sim}{x} = \underset{\sim}{z}. \tag{1.3}$$

However, the linear systems which arise in such computations as the numerical solution of ordinary or partial differential equations may be quite large and sparse (i.e. most $a_{ij} = 0$). Because methods which work well for small, dense linear systems are extremely inefficient for these systems, a great deal of research has been devoted to the development of efficient, practical methods for their solution.

Methods for the solution of a large, sparse linear system may be classified as either iterative or direct. Iterative methods (cf. Young [Y1]) approximate the solution of such a system by computing a sequence of iterates $\underset{\sim}{x}_1, \underset{\sim}{x}_2, \underset{\sim}{x}_3, \ldots$ which converges to the solution $\underset{\sim}{x}$. An iterative method requires an initial approximation $\underset{\sim}{x}_0$ to $\underset{\sim}{x}$ and computes $\underset{\sim}{x}_{k+1}$ from the previous iterates by a rule of the form

$$\underset{\sim}{x}_{k+1} = G_k(\underset{\sim}{x}_0, \underset{\sim}{x}_1, \ldots, \underset{\sim}{x}_k) ,$$

where G_k is a function depending on A, $\underset{\sim}{b}$, and k. The computed solution is taken to be the first iterate $\underset{\sim}{x}_k$ for which some error function (e.g. $\|A\,\underset{\sim}{x}_k - \underset{\sim}{b}\|$) is less than a specified tolerance ε that depends on the problem and the computing environment in which the iterates are computed.

The advantages of iterative methods are that they normally require only a small amount of storage (enough to store the nonzeroes of A, $\underset{\sim}{x}$, $\underset{\sim}{b}$, and perhaps several auxiliary vectors), that they can take advantage of a good initial guess $\underset{\sim}{x}_0$ when one is

known, and that they are very efficient for certain linear systems. In general, however, the determination of when to stop the iteration may be difficult; a particular iterative method may not be widely applicable; and obtaining good iteration parameters (based on the spectrum of A) and a good initial guess may be both extremely critical and difficult. Because of these serious drawbacks, we will investigate only direct methods for the solution of large, sparse systems of linear equations.

Sparse direct methods are usually variants of Gaussian elimination which exploit the sparseness of A and U. The matrix A is factored as in (1.2), and the resulting systems are solved as in (1.3) to obtain the solution $\underset{\sim}{x}$. However, special data structures and algorithms must be used to avoid storing or operating on zeroes in A and U.

The cost of a direct method for solving (1.1) may be measured in terms of the amount of storage or the number of arithmetic operations required to compute $\underset{\sim}{x}$. To simplify our analysis, we will usually consider only those components of the storage or work which dominate the cost. Thus in analyzing Gaussian elimination, we will ignore the storage required for the vectors $\underset{\sim}{x}$, $\underset{\sim}{y}$, $\underset{\sim}{z}$, and $\underset{\sim}{b}$, since the storage needed for the matrices A, D, and U will be far greater for most systems. In fact, we will ignore the storage for A when it is known to be much less than that required for D and U. Similarly, we will count only the arithmetic operations performed in factoring A, since the factorization usually requires far more operations than does the forward- and back-solution.

The cost of sparse symmetric Gaussian elimination for the solution of a sparse linear system depends on the ordering of the equations and the order in which the variables are eliminated. To reduce the cost, it may be desirable to preprocess the system prior to the factorization by reordering the equations and variables. Thus for any $N \times N$ permutation matrix P, we might choose to solve the permuted system

$$PAP^T P \underset{\sim}{x} = P \underset{\sim}{b}, \qquad (1.4)$$

rather than (1.1), if the storage or work required to factor PAP^T were less than that required to factor A. Under our assumptions on A, the permuted matrix PAP^T will also be symmetric, irreducible, and positive definite (cf. Young [Y1], p. 24), so (1.4) may be solved in the same way as (1.1).

The choice of a good ordering for variable elimination (or, equivalently, the choice of a good permutation matrix P) can be quite difficult. In fact, it has been conjectured by Rose and Tarjan [R7] that the problem of obtaining an optimal ordering (i.e. one that minimizes the cost) is NP-complete (cf. Karp [K1] for a discussion of the ramifications of this condition). However, as we shall see in Chapter IV, there are several (heuristic) ordering algorithms which seem to work fairly well in practice and which are not too costly to use.

The advantages of direct methods for solving sparse linear systems are that they produce the exact solution $\underset{\sim}{x}$ (ignoring possible roundoff error) in a finite number of arithmetic operations, that they work well for a wide variety of linear systems, and that

they do not depend on any a priori knowledge about the solution $\underset{\sim}{x}$ or the spectrum of A. However, the number of nonzeroes in U and the number of arithmetic operations required to factor A may be prohibitively large (e.g. in the solution of linear systems arising in the numerical solution of partial differential equations in three dimensions); Gaussian elimination does not take advantage of a good approximation to $\underset{\sim}{x}$ where one is known (e.g. in the solution of systems of nonlinear equations); and the special data structures and algorithms may require a large amount of overhead (i.e. extra storage for pointers in addition to that needed for the nonzeroes in A and U, and extra non-numeric operations in addition to the necessary arithmetic operations).

In this dissertation we investigate ways to mitigate the difficulties with sparse direct methods. In particular, we

(i) develop and analyze the costs of an efficient implementation of sparse symmetric Gaussian elimination;

(ii) investigate possible storage/work trade-offs for direct methods which would make them competitive storage-wise with iterative methods; and

(iii) consider the efficient use of direct methods for sequences of sparse linear systems, specifically as they might arise in the solution of systems of nonlinear equations.

We begin with Chapter II in which we present basic material used throughout the remainder of the thesis. We describe our mathematical notation, introduce the elementary graph theory which is necessary to develop the graph-theoretic and element models for sparse symmetric Gaussian elimination, and describe two model sparse linear systems which arise in the numerical solution of the Poisson equation in the unit square.

In Chapter III we present and analyze algorithms for both dense and sparse symmetric Gaussian elimination. Much of our analysis is given in terms of a graph model for symmetric Gaussian elimination proposed by Parter [P1] and extensively developed by Rose [R4]. We also introduce an element model for sparse symmetric Gaussian elimination which generalizes a view employed by George [G3] and is similar to a model used by Eisenstat [E1]. The graph model and the element model are equivalent in the sense that they both correctly model the evolving structure of U as the U^TDU factorization of A is carried out. However, the element model leads to a recursive form of analysis which may be easier to apply in certain cases.

In Chapter IV we discuss algorithms for finding good elimination orderings for sparse symmetric Gaussian elimination. For general sparse systems there are two well-known ordering algorithms: the minimum deficiency algorithm, which locally minimizes storage, and the minimum degree algorithm, which locally minimizes work (cf. Rose [R4], Duff and Reid [D7]). While neither algorithm produces good orderings for all problems, both have performed well in practice on a variety of sparse linear systems. For sparse linear systems like the model problems of Chapter II, it is possible to obtain near-optimal orderings using a nested dissection algorithm (cf. Birkhoff and George [B3], George [G3]). We examine these special orderings and compare them with the minimum degree ordering.

In Chapter V we describe an efficient implementation of sparse symmetric Gaussian elimination. We describe a new storage scheme for A and U that can take full advantage of both the available symmetry in the matrices and the regular way in which zeroes in A fill in, or become nonzero, in U. The scheme allows the use of symbolic factorization techniques due to Chang [C2] to obtain a practical and efficient software package for sparse symmetric Gaussian elimination.†

In Chapter VI we report on the results of several numerical experiments which illustrate the performance of programs based on the ideas of Chapter V. Of main interest are results for the model problems introduced in Chapter II, since they may be used to estimate the overhead and to compare sparse Gaussian elimination with other direct methods such as band Gaussian elimination (cf. Martin and Wilkinson [M1], Hindmarsh [H2]) and envelope (or profile) Gaussian elimination (cf. George [G1], Eisenstat and Sherman [E8]). In addition, we examine the use of sparse symmetric Gaussian elimination for linear systems arising in the numerical solution of the biharmonic equation in the unit square and give some rough guidelines as to the best ways to solve sparse symmetric linear systems which might arise in practice.

In Chapter VII we consider methods of reducing the storage requirements of direct methods for sparse linear systems, and present a way to trade off storage for work so as to obtain minimal

† In this dissertation we present only the algorithms required to implement these techniques, not the actual programs. For a detailed presentation of the programs, see Eisenstat and Sherman [E7].

storage Gaussian elimination methods which are competitive with iterative methods as far as storage is concerned. We also develop minimal storage variants of band and sparse Gaussian elimination which are particularly well-suited to the model problems of Chapter II.

Finally, in Chapter VIII, we examine the application of sparse symmetric Gaussian elimination to the solution of systems of nonlinear equations. We introduce the Newton-Richardson methods (cf. Eisenstat, Schultz, and Sherman [E4]), a class of Newton-iterative methods which retain the local convergence properties of Newton's method. These methods are computationally more efficient than the straightforward implementation of Newton's method when the Jacobian matrices are symmetric and sparse, and they may offer substantial savings in practice. We also consider more géneral Newton-iterative methods and obtain a result which shows that it is possible to obtain local quadratic convergence with such methods. As examples, we present the results of numerical experiments involving two semi-linear partial differential equations and the minimal surface equation.

CHAPTER II: PRELIMINARIES

I.1 Mathematical Notation

In general, standard mathematical notation is used throughout this dissertation. Those exceptions and extensions which we will make are presented in this section. In Section 2 we introduce some elementary definitions and notation from graph theory, and in Section 3 we describe two model problems which will be used extensively.

We use upper case Roman letters to denote sets and matrices and lower case Roman letters to represent set elements and matrix entries. Sets may be either ordered or unordered, and elements of ordered sets are subscripted to indicate order within the set. Matrix entries are subscripted to indicate row and column position. Vectors are denoted by lower case Roman letters with an under-tilde, and their components are represented with lower case Roman letters subscripted to indicate position within the vector. We denote the size of a set S by $|S|$ and the length of a vector $\underset{\sim}{y}$ by $|\underset{\sim}{y}|$.

A system of linear equations is written as

$$A \underset{\sim}{x} = \underset{\sim}{b}. \qquad (1.1)$$

We use N to denote the order of A (i.e. the number of equations and unknowns in (1.1)). Unless otherwise stated, the coefficient matrix A is assumed to be an $N \times N$ symmetric, positive definite, irreducible matrix, and the vectors $\underset{\sim}{x}$ and $\underset{\sim}{b}$ are of length N.

When the matrix A is factored into the product $U^T DU$, we assume that U is unit upper triangular (i.e. $u_{ij} = 0$ for $j < i$ and $u_{ii} = 1$) and D is a diagonal matrix with positive diagonal entries. The $k \times k$ identity matrix is denoted by I_k.

On occasion we reorder the variables and equations of a linear system to reduce the cost of solving it. Such a reordering leads to a permuted system of the form

$$PAP^T P \underset{\sim}{x} = P \underset{\sim}{b} ,$$

where P is an $N \times N$ permutation matrix (i.e. a matrix of ones and zeroes with exactly one one in each row and column). Under our assumptions on A, PAP^T is also an $N \times N$ symmetric, positive definite, irreducible matrix (cf. Young [Y1], p.24).

In the cost analysis of various algorithms, θ denotes the total time or total operations required, θ_M denotes the number of multiplications (and/or divisions) required, θ_A denotes the number of additions (and/or subtractions) required, and θ_S denotes the number of storage locations required. Throughout, we use $\log n$ to denote $\log_2 n$. In general our analyses are asymptotic; we write

$$c(k) = f(k) + O(g(k))$$

to mean that there is a constant $C < +\infty$ such that for all positive integers k,

$$c(k) - f(k) < Cg(k)$$

and

$$c(k) = f(k) + o(g(k))$$

to mean that

$$\lim_{k\to\infty} [(c(k) - f(k))/g(k)] = 0.$$

Where appropriate, we use "$\sim f(k)$" to mean "$f(k) + o(f(k))$" and "$c(k) \approx f(k)$" to mean "$c(k) = f(k) + o(f(k))$". Finally, following Rivest and Vuillemin [R3], we write

$$c(k) \geq \Omega(f(k))$$

to mean

$$c(k) \geq mf(k) + o(f(k)),$$

for some $m > 0$. This notation gives us a convenient way of specifying asymptotic lower bounds when we do not know the exact value of the leading coefficient m.

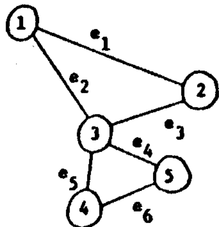
II.2 Elementary Graph Theory

In this section we introduce enough elementary graph theory to develop the models which we use to analyze sparse symmetric Gaussian elimination. Most of our notation is standard and follows Harary [H1] or Rose [R4]. Figure 2.1 illustrates the definitions presented here and should be used for reference.

An (undirected) graph G is a pair of sets $G = (X,E)$ where X is a finite ordered set of $|X|$ vertices and

$$E \subseteq \{\{x,y\}: x,y \in X \text{ and } x \neq y\}$$

G:



$G = (X,E)$ where

$X = \{x_1,x_2,x_3,x_4,x_5\}$

$E = \{e_1,e_2,e_3,e_4,e_5,e_6\}$

$= \{\{x_1,x_2\},\{x_1,x_3\},\{x_2,x_3\},\{x_3,x_5\},\{x_3,x_4\},\{x_4,x_5\}\}$

$Adj(x_1) = \{x_2,x_3\}$ $\quad$ $Deg(x_1) = 2$; $E/x_1 = \{e_1,e_2\}$

$Adj(x_3) = \{x_1,x_2,x_4,x_5\}$ $\quad$ $Deg(x_3) = 4$; $E/x_3 = \{e_2,e_3,e_4,e_5\}$

$G\langle\{x_1,x_2\}\rangle = (\{x_1,x_2\},\{e_1\})$ $\quad$ $G\langle\{x_2,x_3,x_5\}\rangle = (\{x_2,x_3,x_5\},\{e_3,e_4\})$

$G\langle\{x_1,x_2,x_3\}\rangle$ and $G\langle\{x_3,x_4,x_5\}\rangle$ are subcliques of G.

There are four paths between x_1 and x_5 in G:

(x_1,x_2,x_3,x_5), (x_1,x_2,x_3,x_4,x_5), (x_1,x_3,x_5), (x_1,x_3,x_4,x_5).

$Def(x_1) = \emptyset$; $Def(x_3) = \{\{x_1,x_4\},\{x_1,x_5\},\{x_2,x_4\},\{x_2,x_5\}\}$

FIGURE 2.1

is a set of $|E|$ unordered pairs of vertices called edges. If every pair of distinct vertices in X is connected by an edge in E, so that

$$E = \{\{x,y\}: x,y \in X \text{ and } x \neq y\},$$

then G is said to be a complete graph or clique.

For any vertex $x \in X$, we define the set of vertices which are adjacent to x as

$$Adj(x) = \{y \in X: \{x,y\} \in E\},$$

and the degree of x as

$$Deg(x) = |Adj(x)|.$$

The edges in the set

$$E/x = \{\{x,y\} \in E: y \in Adj(x)\}$$

are said to be incident on x.

A subgraph of G is a graph $G' = (X',E')$, where $X' \subseteq X$ and

$$E' \subseteq \{\{x,y\}: x,y \in X' \text{ and } \{x,y\} \in E\}.$$

For any set of vertices $Y \subseteq X$, the Y-induced section subgraph $G<Y>$ is the graph $G<Y> = (Y,E<Y>)$ where

$$E<Y> = \{\{x,y\} \in E: x,y \in Y\}.$$

A subclique of G is a maximal complete subgraph of G, i.e., a complete Y-induced subgraph $G<Y> = (Y,E<Y>)$ where for any $x \in X - Y$, the subgraph of G induced by $Y \cup \{x\}$ is not complete.

A path of length ℓ from vertex x to vertex y is an ordered set of distinct vertices

$$P(x,y) = (v_1,v_2,\ldots,v_{\ell+1}),$$

such that $v_1 = x$, $v_{\ell+1} = y$, and $v_{i+1} \in Adj(v_i)$ for $i = 1,2,\ldots,\ell$. If there is a path between every pair of distinct vectices x and y in X, then G is said to be connected. Corresponding to our assumption that the matrix A of (1.1) is irreducible, we always assume that graphs are connected (cf. Young [Y1], pp. 38-39).

Finally, given a vertex $x \in X$, the deficiency of x is the set

$$Def(x) = \{\{y,z\} \notin E: y,z \in Adj(x) \text{ and } y \neq z\},$$

(i.e. the deficiency of x is the set of edges which must be added to G to make the vertices in $Adj(x)$ pairwise adjacent).

II.3 Model Problems

In this section we derive two model linear systems which will be used extensively to illustrate our results. The two systems arise in the numerical solution of Poisson's equation on the unit square:

$$\begin{aligned} -\Delta v &= f \quad \text{in } D = (0,1)\times(0,1) \\ v &= 0 \quad \text{on } \partial D, \end{aligned} \qquad (3.1)$$

where ∂D denotes the boundary of D. This equation is particularly simple, and special purpose methods may often be used to

solve it more efficiently than we can with Gaussian elimination (cf. Dorr [D5]). However, the results which we obtain for this model problem apply equally well to more general problems to which the special purpose methods are not applicable.

To obtain a numerical solution to (3.1), we cover D with a regular $n \times n$ square grid or mesh D_n with boundary ∂D_n and compute an approximation V_{ij} to $v(ih,jh)$ at each of the $N = n^2$ interior points (ih,jh) in D_n. (Here $h = 1/n+1$ is the vertical or horizontal distance between two adjacent mesh points; see Figure 3.1.). In particular, we use finite difference techniques to replace the differential operator $-\Delta$ at each interior point (ih,jh) with a linear combination of the values V_{ij} at surrounding mesh points. This leads to a system of linear equations which can be solved to obtain the desired approximation (cf. Forsythe and Wasow [F2], pp. 190-195).

A five-point finite difference approximation is obtained by approximating $-\Delta$ at (ih,jh) in terms of the values V_{ij} at (ih,jh) and the four points adjacent to (ih,jh) along vertical or horizontal mesh lines (see Figure 3.2). This leads to the system of equations

$$-V_{i-1,j} - V_{i,j-1} - V_{i,j+1} - V_{i+1,j} + 4V_{ij} = h^2 f(ih,jh), \quad \text{for } (ih,jh) \in D_n \qquad (3.2)$$

$$V_{ij} = 0, \quad \text{for } (ih,jh) \in \partial D_n.$$

This system could also be obtained by using the Rayleigh-Ritz-Galerkin method with linear right-triangular elements (cf. Courant [C5],

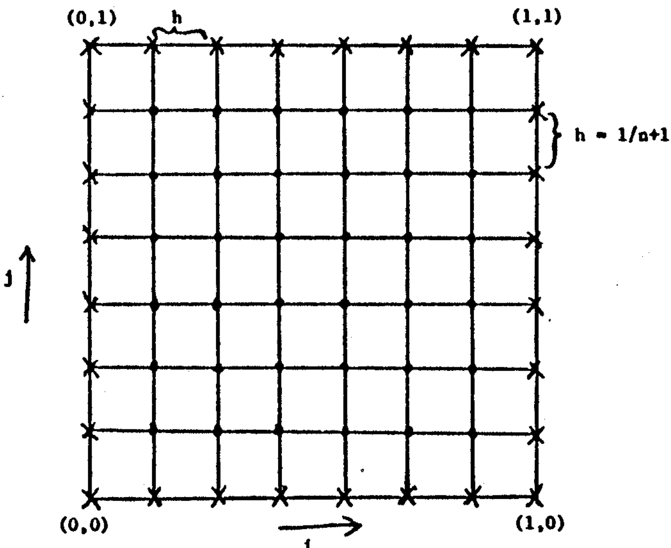


FIGURE 3.1

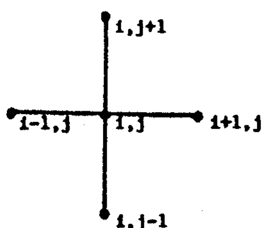


FIGURE 3.2

Strang and Fix [S4], p. 77).

A nine-point finite difference approximation is obtained by approximating $-\Delta$ at (ih,jh) in terms of the values V_{ij} at (ih,jh) and the eight points which border on the four small squares of D_n containing (ih,jh) (see Figure 3.3). This leads to the system of equations

$$- V_{i-1,j-1} - V_{i-1,j} - V_{i-1,j+1} - V_{i,j-1} - V_{i,j+1} - V_{i+1,j-1}$$

$$- V_{i+1,j} - V_{i+1,j+1} + 8V_{ij} = h^2 f(ih,jh), \quad \text{for} \quad (ih,jh) \in D_n$$

$$V_{ij} = 0, \quad \text{for} \quad (ih,jh) \in \partial D_n . \qquad (3.3)$$

This system could also be obtained by using the Rayleigh-Ritz-Galerkin method with a basis of tensor products of piecewise linear functions (cf. Strang and Fix [S4], pp. 77, 87).

Ordering the points of D_n in the natural or row-by-row order (see Figure 3.4), we may write the two systems more concisely as $N \times N$ systems of linear equations

$$A \underset{\sim}{V} = \underset{\sim}{F}, \qquad (3.4)$$

where

$$\underset{\sim}{V} = \{V_{ij}: (ih,jh) \in D_n\},$$

$$\underset{\sim}{F} = \{h^2 f(ih,jh): (ih,jh) \in D_n\},$$

and A is a sparse, symmetric, positive definite and irreducible matrix derived from either (3.2) or (3.3) (cf. Forsythe and Wasow [F2], pp. 190-195).

Specifically, if we use the five-point difference equations to

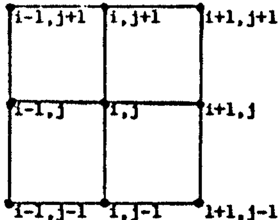


FIGURE 3.3

	1	2	3	4	5	6
	7	8	9	10	11	12
	13	14	15	16	17	18
	19	20	21	22	23	24
	25	26	27	28	29	30
	31	32	33	34	35	36

Natural Ordering of D_6

FIGURE 3.4

derive (3.4), then we obtain what we shall call the $n \times n$ five-point model problem

$$A_5 \underset{\sim}{V} = \underset{\sim}{F}. \tag{3.5}$$

A_5 is the $n \times n$ block tridiagonal matrix

$$A_5 = \begin{bmatrix} B & -I_n & & & \\ -I_n & B & -I_n & & \\ & -I_n & \ddots & \ddots & \\ & & \ddots & \ddots & -I_n \\ & & & -I_n & B \end{bmatrix},$$

where B is the $n \times n$ tridiagonal matrix

$$B = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{bmatrix}$$

and I_n is the $n \times n$ identity matrix.

Alternatively, if we use the nine-point difference equations to derive (3.4), then we obtain what we shall call the $n \times n$ nine-point model problem

$$A_9 \underset{\sim}{V} = \underset{\sim}{F}. \tag{3.6}$$

A_9 is the $n \times n$ block tridiagonal matrix

$$A_9 = \begin{bmatrix} B & -C & & & \\ -C & B & -C & & \\ & -C & \ddots & \ddots & \\ & & \ddots & \ddots & -C \\ & & & -C & B \end{bmatrix},$$

where B and C are the $n \times n$ tridiagonal matrices

$$B = \begin{bmatrix} 8 & -1 & & & \\ -1 & 8 & -1 & & \\ & -1 & 8 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 8 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 & & & \\ 1 & 1 & 1 & & \\ & 1 & 1 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 1 \end{bmatrix}.$$

CHAPTER III: SYMMETRIC GAUSSIAN ELIMINATION

III.1 Introduction

In this chapter we present and analyze symmetric Gaussian elimination for the solution of the system of linear equations

$$A \underset{\sim}{x} = \underset{\sim}{b}, \qquad (1.1)$$

where A is an $N \times N$ symmetric, positive definite matrix. Symmetric Gaussian elimination is equivalent to the square-root-free Cholesky method, i.e., first factoring A into the product $U^T DU$ and then forward- and back-solving to obtain $\underset{\sim}{x}$, and we will often talk interchangeably about these two methods of solving (1.1). Furthermore, since almost the entire cost is connected with the factorization of A, we restrict most of our attention to that portion of the solution process.

In Section 2 we present a row-oriented $U^T DU$ factorization algorithm for dense matrices A and consider two modifications of it which take advantage of sparseness in A and U. We also discuss the type of information about A and U which is required to allow sparse symmetric factorization to be implemented efficiently.

In Section 3 we present the outer product or Crout form of the

$U^T DU$ factorization and use it to determine the costs of the dense row-oriented factorization algorithm presented in Section 2. The two algorithms are numerically equivalent, but when A is sparse, the outer product form is easier to analyze, while the row-oriented form is easier to implement.

In Section 4 we associate an undirected graph with A and present a graph model for sparse symmetric Gaussian elimination. The model is based on the outer product form of the $U^T DU$ factorization and is similar to one suggested by Parter [P1] and extensively developed by Rose [R4]. We use it to analyze the costs of the sparse symmetric factorization process in terms of the zero structure of A.

Finally, in Section 5 we introduce an element model for sparse symmetric Gaussian elimination based on an examination of the structure of the graphs which arise in the graph model. The model generalizes a view taken by George [G3], and it is similar to a model used by Eisenstat [E1] in his analysis of lower bounds for the costs of sparse symmetric factorizations for the five- and nine-point model problems.

III.2 Algorithms for $U^T DU$ Factorization

In this section we present a row-oriented algorithm for the $U^T DU$ factorization of a dense symmetric matrix A and show how to modify it in order to take advantage of sparseness in A and U. In so doing we will determine the information about A and U that is required to make the sparse factorization process effi-

cient, although we defer until Chapter V a detailed discussion of the implementation of sparse symmetric factorization.

Algorithm 2.1 is a row-oriented algorithm for U^TDU factorization. The diagonal entries d_{kk} of D are stored in an array D in which $D(k) = d_{kk}$, and in practice, all of the computations would be performed on the upper triangular matrix U. However, to avoid notational confusion in this discussion, we present the algorithm using the upper triangular array M instead. At any time during the execution of Algorithm 2.1, part of M contains entries of U, part contains entries of DU (the matrix product of D and U), and part is unspecified. Figure 2.1a shows the contents of M just prior to the start of the k-th step of the factorization. During the k-th step the algorithm computes the k-th column of U in the k-th column of M (line 8), the k-th row of DU in the k-th row of M (lines 10-11), and d_{kk} (line 9). At the conclusion of the k-th step, the contents of M have been modified as shown in Figure 2.1b. At the end of the algorithm, M contains exactly the entries of U.

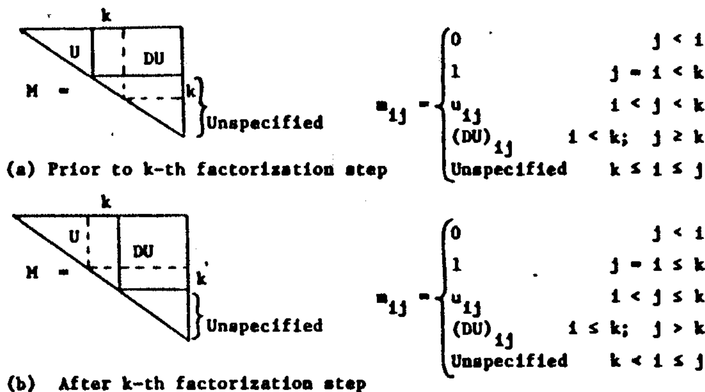
When A is dense, Algorithm 2.1 may be implemented efficiently, since it stores and operates on just the upper triangle of A, D, and U. However, when A is sparse, Algorithm 2.1 fails to exploit the zeroes in A and U to reduce the storage or work.

Conceptually, at least, it is possible to avoid arithmetic operations on zeroes by testing the operands prior to using them. One such modification of Algorithm 2.1 is given as Algorithm 2.2. Unfortunately, there are two serious problems with this approach. First, all of the entries of the upper triangles of A and M


```
line
 1   For  k ← 1 to N  do
 2      [m_kk ← 1;
 3       d_kk ← a_kk;
 4       For  j ← k+1 to N  do
 5          [m_kj ← a_kj];
 6       For  i ← 1 to k-1  do
 7          [t ← m_ik;
 8           m_ik ← m_ik/d_ii;
 9           d_kk ← d_kk - t·m_ik;
10           For  j ← k+1 to N  do
11              [m_kj ← m_kj - m_ik·m_ij]]];
     Comment: Now  M = U
```

Row-oriented Dense U^TDU Factorization

ALGORITHM 2.1



(a) $m_{ij} = \begin{cases} 0 & j < i \\ 1 & j = i < k \\ u_{ij} & i < j < k \\ (DU)_{ij} & i < k;\ j \ge k \\ \text{Unspecified} & k \le i \le j \end{cases}$

(b) $m_{ij} = \begin{cases} 0 & j < i \\ 1 & j = i \le k \\ u_{ij} & i < j \le k \\ (DU)_{ij} & i \le k;\ j > k \\ \text{Unspecified} & k < i \le j \end{cases}$

Structure of array M during a step of Algorithm 2.1

FIGURE 2.1

line	
1	For $k \leftarrow 1$ to N do
2	$\{m_{kk} \leftarrow 1;$
3	$d_{kk} \leftarrow a_{kk};$
4	For $j \in \{n\colon u_{kn} \neq 0\}$ do
5	$\{m_{kj} \leftarrow 0\};$
6	For $j \in \{n > k\colon a_{kn} \neq 0\}$ do
7	$\{m_{kj} \leftarrow a_{kj}\};$
8	For $i \in \{n < k\colon u_{nk} \neq 0\}$ do
9	$\{t \leftarrow m_{ik};$
10	$m_{ik} \leftarrow m_{ik}/d_{ii};$
11	$d_{kk} \leftarrow d_{kk} - t \cdot m_{ik};$
12	For $j \in \{n > k\colon u_{in} \neq 0\}$ do
13	$\{m_{kj} \leftarrow m_{kj} - m_{ik} \cdot m_{ij}\}\}\};$
	Comment: Now $M = U$

Row-oriented Sparse U^TDU Factorization
(with zero testing)

ALGORITHM 2.2

---

line	
1	For $k \leftarrow 1$ to N do
2	$\{m_{kk} \leftarrow 1;$
3	$d_{kk} \leftarrow a_{kk};$
4	For $j \in ru_k$ do
5	$\{m_{kj} \leftarrow 0\};$
6	For $j \in \{n \in ra_k\colon n > k\}$ do
7	$\{m_{kj} \leftarrow a_{kj}\};$
8	For $i \in cu_k$ do
9	$\{t \leftarrow m_{ik}$
10	$m_{ik} \leftarrow m_{ik}/d_{ii};$
11	$d_{kk} \leftarrow d_{kk} - t \cdot m_{ik};$
12	For $j \in \{n \in ru_i\colon n > k\}$ do
13	$\{m_{kj} \leftarrow m_{kj} - m_{ik} \cdot m_{ij}\}\}\};$
	Comment: Now $M = U$

Row-oriented Sparse U^TDU Factorization
(with pre-processing)

ALGORITHM 2.3

must be stored, since any of them could be tested in lines 4, 6, 8, or 12, or used as m_{kj} in line 13. And second, there are more testing operations than arithmetic operations, so that the running time of Algorithm 2.2 could be proportional to the amount of testing rather than the amount of arithmetic.

To see how to avoid these problems, let us assume that for each k, $1 \le k \le N$, we have precomputed

(i) the set ra_k of columns $j \ge k$ for which $a_{kj} \neq 0$;

(ii) the set ru_k of columns $j > k$ for which $u_{kj} \neq 0$;

(iii) the set cu_k of rows $i < k$ for which $u_{ik} \neq 0$.

We can then modify Algorithm 2.2 to obtain Algorithm 2.3 in which the only entries of M which are used are those corresponding to nonzeroes in A or U. In Chapter V, we will describe an implementation for Algorithm 2.3. We will show that the total storage required is proportional to the number of nonzeroes in A and U and that the total running time is proportional to the number of arithmetic operations on nonzeroes. In addition, we will show that the preprocessing required to compute the sets $\{ra_k\}$, $\{ru_k\}$, and $\{cu_k\}$ can be performed in time proportional to the total storage.

III.3 Analysis of Dense U^TDU Factorization

In this section we analyze the storage and arithmetic costs of Algorithm 2.1. For dense matrices A in which all entries are assumed to be nonzero, we could perform this analysis by examining only the index bounds on the loops. However, to lay groundwork for

the analysis of Algorithm 2.3 in Section 4, we instead present and analyze the outer product or Crout form of dense U^TDU factorization. This form of factorization corresponds to the usual presentation of Gaussian elimination (cf. Isaacson and Keller [I1], pp. 29-31) and to the graph model which we use to analyze the costs of sparse factorization in Section 4. The row-oriented and outer product factorization algorithms require the same number of arithmetic operations, but they perform them in different orders.

An algorithmic statement of the outer product form of dense symmetric factorization is given as Algorithm 3.1. In order to analyze its cost, it is useful to give a more intuitive matrix formulation (cf. Rose [R4]). At the first step of the algorithm, we compute d_{11} and the entries of the first row of the strict upper triangle of U. We write A as

$$A = A^{(1)} = \left[\begin{array}{c|c} a_1 & r_1^T \\ \hline r_1 & C_1 \end{array}\right],$$

where a_1 is 1×1, r_1 is $(N-1)\times1$, and C_1 is $(N-1)\times(N-1)$. We then set $d_{11} = a_1$, replace the first row of the strict upper triangle of U with r_1^T/a_1, and compute $A^{(2)}$ as

$$A^{(2)} = \left[\begin{array}{c|c} a_1 & r_1^T \\ \hline r_1 & C_1 - \frac{r_1 r_1^T}{a_1} \end{array}\right].$$

At the k-th step of the algorithm $(1 \le k \le N-1)$, we compute d_{kk} and the k-th row of the strict upper triangle of U. We write


```
line
 1   For  k ← 1 to N-1  do
 2      [d_kk ← a_kk;
 3       For  j ← k+1 to N  do
 4          [u_kj ← a_kj/a_kk;
 5           For  i ← k+1  to N  do
 6              [a_ij ← a_ij - a_ki·u_kj]]];
```

Outer-Product Form of Dense U^TDU Factorization

ALGORITHM 3.1

$A^{(k)}$ as

$$A^{(k)} = \left[\begin{array}{c|cc} A_k & \multicolumn{2}{c}{B_k^T} \\ \hline B_k & a_k & r_k^T \\ & r_k & C_k \end{array}\right],$$

where A_k is $(k-1)\times(k-1)$, B_k is $(N-k+1)\times(k-1)$, a_k is 1×1, r_k is $(N-k)\times1$, and C_k is $(N-k)\times(N-k)$. We then set $d_{kk} = a_k$ and replace the k-th row of the strict upper triangle of U with r_k^T/a_k. Finally, we compute $A^{(k+1)}$ as

$$A^{(k+1)} = \left[\begin{array}{c|cc} A_k & \multicolumn{2}{c}{B_k^T} \\ \hline B_k & a_k & r_k^T \\ & r_k & C_k - \frac{r_k r_k^T}{a_k} \end{array}\right].$$

Since A and the matrices $A^{(k)}$ are symmetric, we need only

store and operate on their upper triangles. Thus at the k-th step, we compute the k-th row of the strict upper triangle of U and those entries of $r_k r_k^T/a_k$ and $A^{(k+1)}$ which lie in the upper triangle. We can do so by computing

(i) r_k^T/a_k, at a cost of $N-k$ divisions,

(ii) the upper triangle of $r_k r_k^T/a_k$, at a cost of $(N-k)(N-k+1)/2$ multiplications, and

(iii) the upper triangle of $A^{(k+1)}$, at a cost of $(N-k)(N-k+1)/2$ subtractions.

Let $\theta_M^{(k)}$ and $\theta_A^{(k)}$ denote the number of multiplication/divisions and addition/subtractions, respectively, required at the k-th step. Then

$$\theta_M^{(k)} = (N-k)(N-k+3)/2$$

and

$$\theta_A^{(k)} = (N-k)(N-k+1)/2,$$

so the total costs $\theta_M(A)$ and $\theta_A(A)$ of the U^TDU factorization of A are given by

$$\theta_M(A) = \sum_{k=1}^{N-1} \theta_M^{(k)} = \sum_{k=1}^{N-1} (N-k)(N-k+3)/2 \qquad (3.1)$$

$$\theta_A(A) = \sum_{k=1}^{N-1} \theta_A^{(k)} = \sum_{k=1}^{N-1} (N-k)(N-k+1)/2. \qquad (3.2)$$

The factorization can be computed in place (destroying A), and using (3.1) and (3.2), we can obtain the following result (cf. Isaacson and Keller [I1], pp. 54-55).

Theorem 3.1: The costs of the U^TDU factorization of a matrix A are

$$\theta_S(A) = 1/2\ N^2 + O(N),$$

$$\theta_M(A) = 1/6\ N^3 + O(N^2), \text{ and}$$

$$\theta_A(A) = 1/6\ N^3 + O(N^2).$$

III.4 A Graph Model for Sparse U^TDU Factorization

It is evident that the costs given in Theorem 3.1 depend only on the fact that r_k contains $N-k$ nonzero entries, not on the actual values of the nonzeroes. In this section we associate an undirected graph with the matrix A and introduce a graph-theoretic interpretation of the sparse symmetric factorization process. The graph model, which is essentially identical to a model used previously by Parter [P1] and Rose [R4], takes account of the number and position of the nonzero entries in the matrices $A^{(k)}$, but ignores their values. The cost analysis based on it is actually an analysis of the costs of Algorithm 2.3 for sparse matrices A.

Since the graph model interprets the outer product algorithm for U^TDU factorization as a non-numeric operation on a graph representing the zero structure of A, it cannot exploit any accidental creation of zeroes due to exact arithmetic cancellation in computing the matrices $A^{(k)}$. Once a position in a matrix $A^{(k)}$ has become nonzero, it will never again be treated as a zero in a later matrix. Ignoring accidental creation of zeroes is quite reasonable, since exact arithmetic cancellation almost never occurs

in practice (cf. Brayton, Gustavson, and Willoughby [B5] for a discussion of this point).

With an $N \times N$ symmetric matrix A, we associate the undirected graph $G(A) = (X(A), E(A))$, where $X(A) = \{x_1, x_2, \ldots, x_N\}$ is the set of vertices and

$$E(A) = \{\{x_i, x_j\} : i \neq j \text{ and } a_{ij} \neq 0\}$$

is the set of edges. The vertex x_i corresponds to row i (or column i) of A, and the edges in $E(A)$ correspond to the nonzero entries of A (see Figure 4.1). If A is irreducible, then $G(A)$ is connected (cf. Blum [B4], p. 137).

Given a graph $G = (X,E)$ and a vertex $x \in X$, we define the x-elimination graph of G as the graph $G_x = (X_x, E_x)$ which is formed from G by deleting x from X and the edges incident on x from E and adding those edges necessary to make the vertices in $Adj(x)$ in G pairwise adjacent in G_x (see Figure 4.2). Thus $X_x = X - \{x\}$ and

$$E_x = E \cup Def(x) - E/x.$$

We now turn to the graph interpretation of Algorithm 3.1. At the k-th step of the algorithm, we compute $A^{(k+1)}$ from $A^{(k)}$. Ignoring exact arithmetic cancellation, $a_{ij}^{(k+1)}$ $(k < i \leq j)$ is nonzero if and only if either $a_{ij}^{(k)}$ was nonzero or both $a_{ki}^{(k)}$ and $a_{kj}^{(k)}$ were nonzero. To formulate this condition in terms of graphs, we define

$$G^{(k)} = (X^{(k)}, E^{(k)}) = G(A^{(k)}) \langle\{x_i : i \geq k\}\rangle .$$

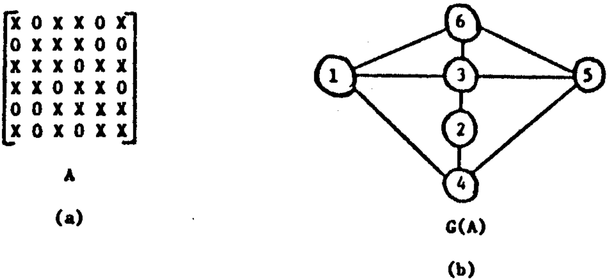


FIGURE 4.1

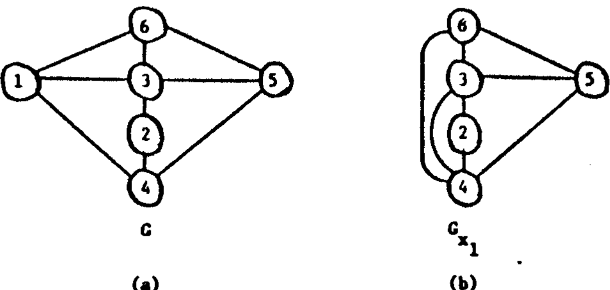


FIGURE 4.2

Then for $k < i, j \le N$, $\{x_i,x_j\} \in E^{(k+1)}$ if and only if either $\{x_i,x_j\} \in E^{(k)}$ or both x_i and x_j were adjacent to x_k in $G^{(k)}$. But this is exactly the condition which characterizes the structure of $G_{x_k}^{(k)}$, so we have $G^{(k+1)} = G_{x_k}^{(k)}$. Thus there is a simple relationship between the operation of forming an elimination graph and that of performing a step of symmetric factorization.

The graph elimination process defined by the sequence $G^{(1)},G^{(2)},\ldots,G^{(N)}$ models the entire sparse symmetric Gaussian elimination process on A as successive vertex eliminations on the undirected graph $G(A)$. This graph elimination process is related to the outer product formulation of sparse symmetric Gaussian elimination in several ways:

(i) $G^{(k)} = G(A^{(k)}) <\{x_i: i \ge k\}>$;

(ii) the fill-in caused by eliminating the k-th variable is represented by $Def(x_k)$ in $G^{(k)}$; and

(iii) at the k-th step, the zero structure of r_k is represented by the edges in $E^{(k)}/x_k$, and the number of nonzeroes in r_k is equal to $Deg(x_k)$ in $G^{(k)}$.

We can analyze the cost of computing the U^TDU factorization of a sparse symmetric matrix A with Algorithm 2.3 by examining the graph elimination process on $G(A)$. To simplify our notation somewhat, we will define $d_k = Deg(x_k)$ in $G^{(k)}$. Then analyzing the outer product algorithm as in Section 3, we find that we can perform the k-th step of sparse symmetric Gaussian elimination by

computing

(i) r_k^T/a_k, at a cost of d_k divisions,

(ii) the upper triangle of $r_k r_k^T/a_k$, at a cost of $d_k(d_k+1)/2$ multiplications, and

(iii) the upper triangle of $A^{(k+1)}$, at a cost of $d_k(d_k+1)/2$ subtractions.

Again letting $\theta_M^{(k)}$ and $\theta_A^{(k)}$ denote the number of multiplication/divisions and addition/subtractions, respectively, required at the k-th step, we have

$$\theta_M^{(k)} = d_k(d_k+3)/2 \quad \text{and}$$

$$\theta_A^{(k)} = d_k(d_k+1)/2.$$

So the total costs $\theta_M(A)$ and $\theta_A(A)$ of the U^TDU factorization of A are given by

$$\theta_M(A) = \sum_{k=1}^{N-1} \theta_M^{(k)} = \sum_{k=1}^{N-1} d_k(d_k+3)/2 \quad \text{and}$$

$$\theta_A(A) = \sum_{k=1}^{N-1} \theta_A^{(k)} = \sum_{k=1}^{N-1} d_k(d_k+1)/2.$$

The vectors r_k are such that $u_{kj} \ne 0$ iff and only if r_k^T contains a nonzero in the corresponding column position, so the number of offdiagonal nonzeroes in the k-th row of U is equal to d_k. Hence we obtain the following result similar to Theorem 3.1 (cf. Rose [R4], p. 203).

<u>Theorem 4.1</u>: The costs of the sparse U^TDU factorization of the matrix A are

$$\theta_S(A) = N + \sum_{k=1}^{N-1} d_k$$

$$\theta_M(A) = \sum_{k=1}^{N-1} d_k(d_k+3)/2, \quad \text{and}$$

$$\theta_A(A) = \sum_{k=1}^{N-1} d_k(d_k+1)/2 .$$

<u>Examples</u>: The following examples, similar to those used by Bunch [B6], illustrate the graph-theoretic model of the sparse symmetric Gaussian elimination process. In each case a matrix A is given, and the first step of elimination is illustrated for both the matrix and its associated undirected graph.

1.

$$A = A^{(1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & 0 \\ a_{13} & 0 & a_{33} \end{bmatrix} \qquad G(A) = G^{(1)}:$$

$$A^{(2)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} - a_{12}^2/a_{11} & -a_{12}a_{13}/a_{11} \\ a_{13} & -a_{12}a_{13}/a_{11} & a_{33} - a_{13}^2/a_{11} \end{bmatrix} \qquad G^{(2)}:$$

Here the fill-in edge $\{x_2,x_3\}$ is added to $G^{(2)}$ since $Adj(x_1) = \{x_2,x_3\}$.

2.

$$A = A^{(1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \qquad G(A) = G^{(1)}:$$

$$A^{(2)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} - a_{12}^2/a_{11} & a_{23} - a_{12}a_{13}/a_{11} \\ a_{13} & a_{23} - a_{12}a_{13}/a_{11} & a_{33} - a_{13}^2/a_{11} \end{bmatrix} \qquad G^{(2)}:$$

Here the edge $\{x_2,x_3\}$ is added to $G^{(2)}$, but it is not a fill-in edge since $\{x_2,x_3\} \in E^{(1)}$. It should be noted that $\{x_2,x_3\}$ is still included in $E^{(2)}$ even if

$$a_{23}^{(2)} = a_{23}^{(1)} - (a_{13}^{(1)}/a_{11}^{(1)})a_{12}^{(1)} = 0$$

(i.e., the accidental creation of a zero in $a_{23}^{(2)}$ is ignored).

III.5 <u>An Element Model for Sparse Symmetric Gaussian Elimination</u>

In this section we introduce an element model for sparse symmetric Gaussian elimination which we will use later in our discussions of matrix reordering (Chapter IV), implementation (Chapter V), and minimal storage methods (Chapter VII). The model leads to a recursive form of analysis which is sometimes simpler to use than the graph-theoretic analysis, and it has the additional advantage that it is an extremely natural model for sparse linear systems arising in the use of finite element methods in the numerical solu-

tion of partial differential equations. (In fact, models similar to ours have been used by George [G3] and Eisenstat [E1] in their analyses of the costs of sparse symmetric factorization for the nine-point model problem.)

The element model is an alternative view of the graph-theoretic model which emphasizes the subcliques of a graph rather than its edges. Since it can be shown (cf. Harary [H1]) that an undirected graph is uniquely characterized by its subcliques as well as by its edges, this view of a graph is well-defined.

For a graph $G = (X,E)$, an <u>element</u> is a set of vertices $s \subseteq X$ such that $G\langle s\rangle$ is a subclique, and we associate with G the unique set

$$S(G) = \{s_1, s_2, \ldots, s_q\}$$

of elements which correspond to its distinct subcliques. If $G = G(A)$ is the graph associated with an $N \times N$ symmetric matrix A, then the elements of $S(G)$ correspond to dense symmetric minors of A. Figure 5.1 illustrates the relationships among matrices, graphs, and elements.

Let $G = (X,E)$ and $S = S(G)$. For any vertex $x \in X$, we denote the <u>set of elements containing x</u> by $S\langle\{x\}\rangle$. If x is contained in exactly one element, then it is said to be an <u>interior vertex</u> (of that element); otherwise it is said to be a <u>boundary vertex</u> of each element in $S\langle\{x\}\rangle$.

In the x-elimination graph G_x of G, all the members of $\mathrm{Adj}(x)$ in G have become pairwise adjacent, and $G\langle \mathrm{Adj}(x)\rangle$ is

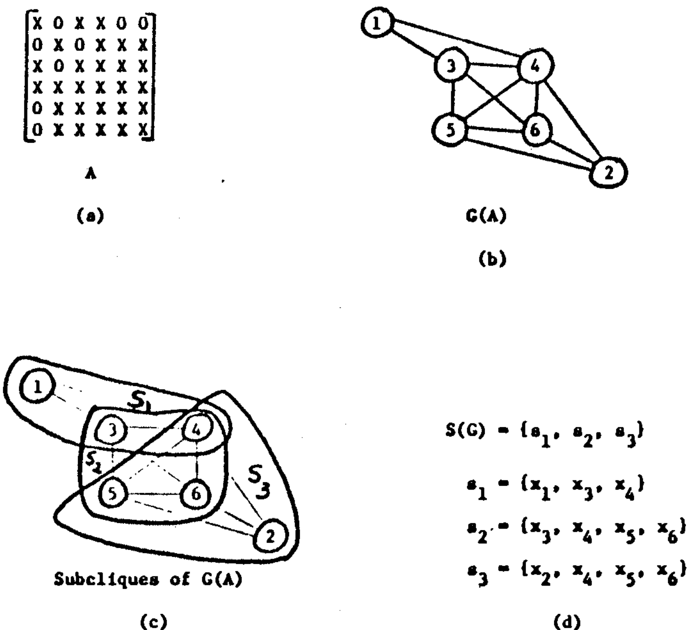


FIGURE 5.1

a complete graph. Hence there must be some subclique of G_x which contains $G \langle Adj(x) \rangle$, i.e., there must be an element $s \in S(G_x)$ such that $Adj(x) \subseteq s$. All of the elements in $S(G) \langle \{x\} \rangle$ have been merged into a single superelement s in $S(G_x)$.† Eliminating the vertex x causes the formation of at least one element in $S(G_x)$ which was not an element of $S(G)$ (i.e. s), and for convenience, we number any newly-formed or modified elements consecutively starting with s_{q+1}, where s_q was the highest-numbered element in $S(G)$. Examples of the relationship between $S(G)$ and $S(G_x)$ are shown in Figure 5.2.

We can now model the entire sparse symmetric Gaussian elimination process on a symmetric matrix A with an element merging process, just as we previously modelled it with a graph elimination process. If $G^{(1)}, G^{(2)}, \ldots, G^{(N)}$ is the graph elimination process for A, then the element merging process for A is the sequence of element sets $S^{(1)}, S^{(2)}, \ldots, S^{(N)}$ in which $S^{(k)} = S(G^{(k)})$.

One way of representing an element merging process is to define an element merge tree which reflects the essential characteristics of the merges that were performed. We construct the tree in a bottom-up manner, placing the original elements of $S(G)$ on the bottom level. If eliminating vertex x_k caused the formation of the element s_q in $S^{(k+1)}$, then a tree node labelled s_q is added to the tree. s_q is placed on the level of the tree which is

† Depending on the structure of G, other elements of $S(G)$ may have also been modified or deleted (see Figure 5.2).

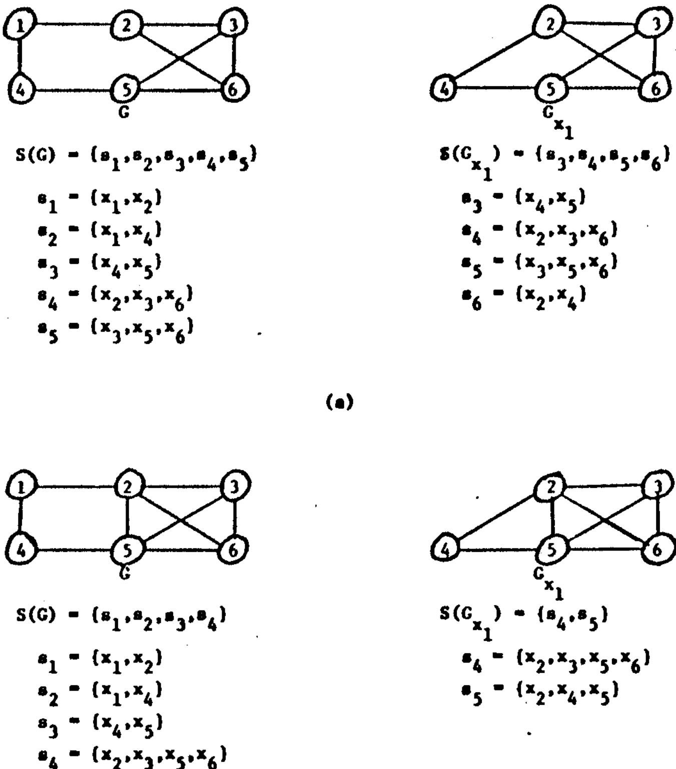


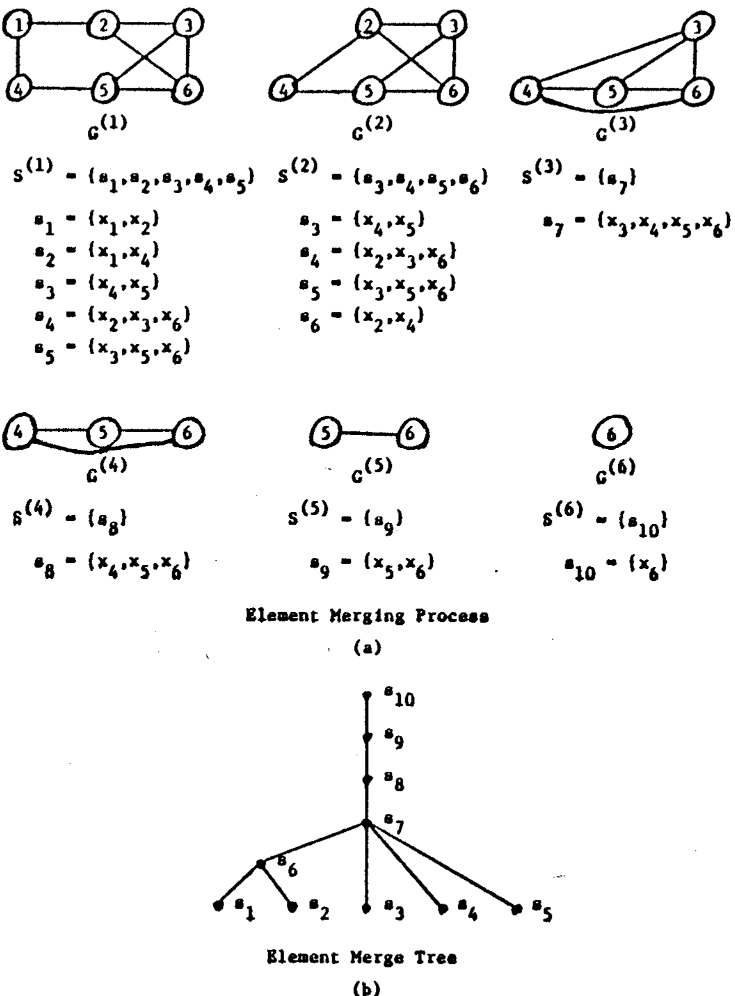
FIGURE 5.2

one higher than the highest level among elements of $S^{(k)}$ which were modified or deleted to form s_q, and all these elements become sons of s_q in the tree. At the top of the tree is the single element of $S^{(N)}$. Figure 5.3 gives an example of an element merging process and an element merge tree. The levels of the element merge tree are numbered consecutively starting with level 0 at the root (top).

The reason that the element model is so natural for linear systems arising from the use of the finite element method is that simple finite element meshes often have the property that they can also serve as <u>element diagrams</u>, i.e., they represent geometrically the element structure of the associated graph. Each element s corresponds to a planar region in the mesh, and the vertices of s correspond to points either in the interior or on the boundary of that region. When a vertex x is in several elements, the corresponding point is on a common boundary of the regions corresponding to the elements of $S\langle\{x\}\rangle$.

Element diagrams provide a second representation of an element merging process. Associating an element diagram with each element set of the element merging process, we find that the regions containing x_k in the diagram corresponding to $S^{(k)}$ are merged into a single region in the diagram corresponding to $S^{(k+1)}$ (see Figure 5.7).

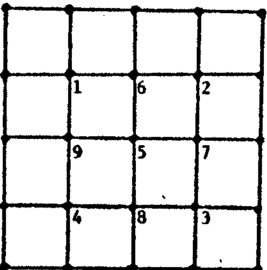
<u>Example</u>: To illustrate the ideas introduced in this section, we consider the square nine-point finite element mesh. Figure 5.4 shows a sample mesh D_n with boundary ∂D_n, the zero structure



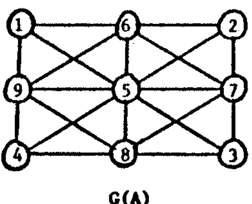
Element Merging Process
(a)

Element Merge Tree
(b)

FIGURE 5.3



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



$S(G(A)) = \{s_1, s_2, s_3, s_4\}$

$s_1 = \{x_1, x_5, x_6, x_9\}$
$s_2 = \{x_2, x_5, x_6, x_7\}$
$s_3 = \{x_3, x_5, x_7, x_8\}$
$s_4 = \{x_4, x_5, x_8, x_9\}$

FIGURE 5.4

of a matrix A corresponding to D_n, the graph $G(A)$, and the element set $S(G(A))$. (The vertices and elements have been numbered arbitrarily.)

In constructing an element diagram for a nine-point finite element mesh D_n, it is convenient to include the mesh points of ∂D_n in "pseudo-elements," in order to avoid any special handling of the mesh points of D_n near the boundary. The mesh squares along the boundary of the mesh geometrically represent these "pseudo-elements," just as the interior mesh squares represent the elements in the graph associated with D_n. This convention has no effect on the matrix or graph associated with D_n or on the sparse Gaussian elimination process; it simply adds $4n+4$ extra elements to the element set corresponding to D_n and makes it easier to bound the storage cost of Algorithm 2.3 in terms of the structure of the element merge tree. Figure 5.5 shows the element diagram for the sample mesh.

Figures 5.6 and 5.7 show the element merge tree and sequence of element diagrams which correspond to the element merging process for the sample problem. As we pointed out before, the diagram corresponding to $S^{(k+1)}$ is obtained from the diagram corresponding to $S^{(k)}$ by geometrically merging the regions corresponding to the elements in $S^{(k)}\langle\{x_k\}\rangle$ and deleting x_k.

An examination of $S^{(6)} - S^{(9)}$ in the example shows little change — only a sequence of interior vertices of a single element have been eliminated, leaving unchanged the overall number of elements and boundary vertices. This corresponds to a procedure

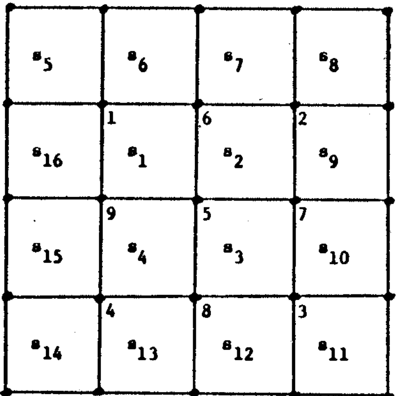


FIGURE 5.5

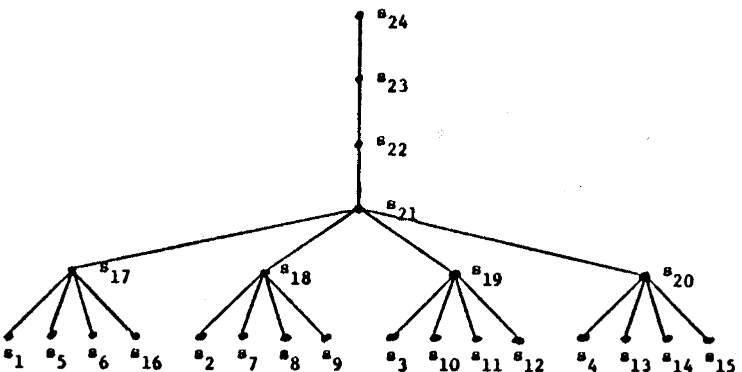
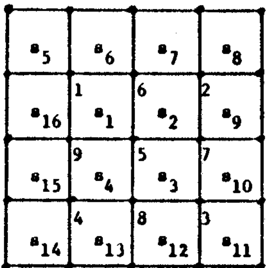
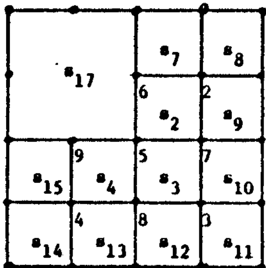


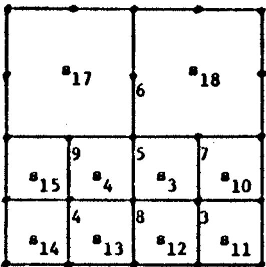
FIGURE 5.6



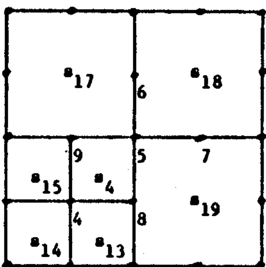
$S^{(1)} = \{s_1, s_2, \ldots, s_{16}\}$
(a)



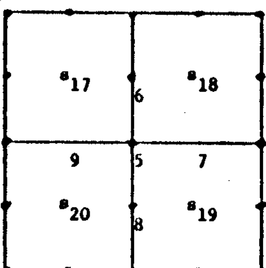
$S^{(2)} = \{s_2, s_3, s_4, s_7, s_8, \ldots, s_{15}, s_{17}\}$
(b)



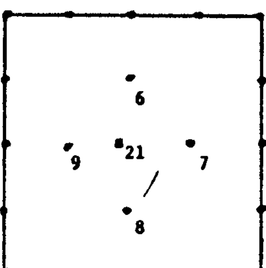
$S^{(3)} = \{s_3, s_4, s_{10}, \ldots, s_{15}, s_{17}, s_{18}\}$
(c)



$S^{(4)} = \{s_4, s_{13}, s_{14}, s_{15}, s_{17}, s_{18}, s_{19}\}$
(d)

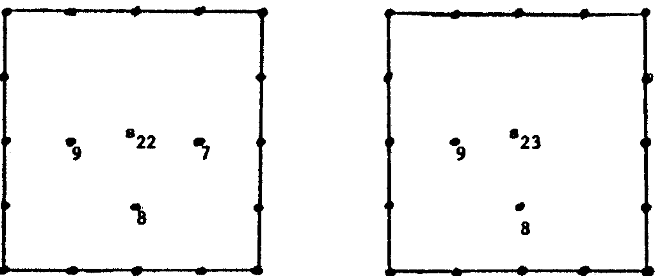


$S^{(5)} = \{s_{17}, s_{18}, s_{19}, s_{20}\}$
(e)



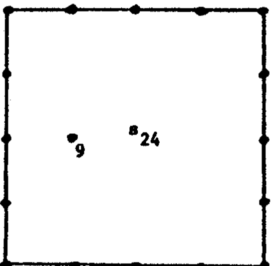
$S^{(6)} = \{s_{21}\}$
(f)

FIGURE 5.7



$S^{(7)} = \{s_{22}\}$
(g)

$S^{(8)} = \{s_{23}\}$
(h)



$S^{(9)} = \{s_{24}\}$
(i)

FIGURE 5.7 (continued)

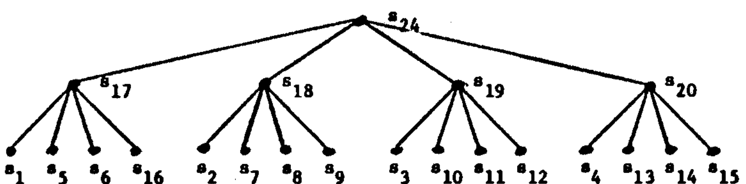


FIGURE 5.8

called static condensation which is often used in practice to eliminate all interior nodes as soon as they are created, thus simplifying the elimination process (cf. Strang and Fix [S4], p. 81). To simplify our notation in this case, we introduce the notion of a compacted element merge tree in which we represent the sequence of consecutive interior vertex eliminations as a single element merge. Figure 5.8 shows the compacted element merge tree for the example. For $k \geq 1$ in the example, level k of the tree contains 4^k elements of $\sim 4n/2^k$ vertices each.† The number of elements of the element merge tree which do not appear in the compacted element merge tree equals the number of interior nodes eliminated with static condensation.

† This regularity is due to the inclusion of the "pseudo-elements" in the tree.

CHAPTER IV: ELIMINATION ORDERINGS

IV.1 Introduction

As we have pointed out before, the storage and work required for sparse symmetric Gaussian elimination depend largely on the ordering of the equations and variables. Thus for some $N \times N$ permutation matrix P, it may be less costly to solve the permuted (or reordered) system

$$PAP^T P \underset{\sim}{x} = P \underset{\sim}{b} \tag{1.1}$$

than it is to solve the original system

$$A \underset{\sim}{x} = \underset{\sim}{b}. \tag{1.2}$$

In this chapter we consider a number of algorithms for choosing a permutation matrix P so that the costs of solving (1.1) are reduced.

Since the costs of Gaussian elimination depend solely on the zero structure of the coefficient matrix, it is quite natural to consider the reordering problem in the context of the graph elimination process introduced in Section III.4. The problem then becomes that of reordering the vertices of the graph $G(A)$ to

reduce the costs given in Theorem III.4.1. Viewing the problem in this way is advantageous because algorithms for graph reordering can be stated easily and because for any permutation matrix P, $G(PAP^T)$ and $G(A)$ are identical up to the numbering of the vertices.

In general the ordering problem is very difficult. To begin with, it is not normally possible to minimize both the work and storage simultaneously. Letting $\mathrm{Deg}(x_k^{(k)})$ denote the degree of x_k in $G^{(k)}$, minimizing work is equivalent to minimizing

$$\sum_{k=1}^{N-1} \mathrm{Deg}(x_k^{(k)})^2,$$

while minimizing storage is equivalent to minimizing

$$\sum_{k=1}^{N-1} \mathrm{Deg}(x_k^{(k)}) .$$

Furthermore, since the costs of sparse Gaussian elimination depend directly on the graphs $G^{(k)}$, rather than on $G(A)$, it is necessary to examine the entire graph elimination process on $G(A)$ in reordering $X(A)$. Thus it is ordinarily far too costly to obtain an optimal ordering (i.e. an ordering of $X(A)$ that actually minimizes either the storage or the work). Only for perfect elimination graphs (i.e. graphs for which there is an ordering which causes no fill-in) can optimal orderings (with respect to storage) be obtained at reasonable cost (cf. Rose and Tarjan [R6]).

In Section 2 we consider ordering algorithms which use local optimization criteria to select x_k from $X^{(k)}$, given

$x_1, x_2, \ldots, x_{k-1}$. We discuss the minimum deficiency algorithm, which locally minimizes the fill-in or storage cost, and the minimum degree algorithm, which locally minimizes the work. The algorithms might be called "greedy" algorithms, since x_k is always chosen as a vertex $x \in X^{(k)}$ which has minimum cost associated with it (i.e. minimum $|Def(x)|$ or minimum $Deg(x)^2$). Both of these local minimization algorithms may be used to reorder the graphs corresponding to arbitrary sparse symmetric positive definite matrices, and practical experience has indicated that the resulting costs will often be quite small (cf. Woo, Roberts, and Gustavson [W3]).

In Section 3 we consider special orderings for matrices like A_5 or A_9 which arise from an underlying geometric mesh. Lower bounds on the costs of sparse Gaussian elimination are known for such matrices (cf. Hoffman, Martin, and Rose [H3], George [G3], and Eisenstat [E1]), and we define near-optimal orderings to be those orderings for which the costs are within a constant factor of the lower bounds as $N \to \infty$. It is possible to efficiently obtain near-optimal orderings by exploiting the structure of the underlying mesh using a "divide-and-conquer" strategy. For the five- and nine-point model problems this leads to the nested dissection orderings of Birkhoff and George [B3] and George [G3], respectively. More generally, the divide-and-conquer idea may be applied to any two- or three-dimensional mesh, and Eisenstat [E1] has proved that the resulting ordering is near-optimal.

Finally, in Section 4 we report the results of a number of experiments which compare the minimum degree and nested dissection orderings for the model problems. While our results neither prove

nor disprove the conjecture (due to George [G4]) that minimum degree algorithms produce near-optimal orderings for the model problems, they do support the view that the minimum degree algorithm may often be a reasonable choice in practice, particularly for the five-point model problem (cf. Woo, Roberts, and Gustavson [W3]).

IV.2 Local Cost Minimization Algorithms

In this section we present two reordering algorithms which may be applied to arbitrary sparse symmetric positive definite matrices. The algorithms reorder the vertices in $X(A)$ by examining the graph elimination process on $G(A)$. In particular, they locally minimize the costs of sparse symmetric Gaussian elimination by selecting as x_k that vertex $x \in X^{(k)}$ which adds the least to the overall cost of Gaussian elimination as measured in terms of storage or work. In this sense the algorithms are "greedy" algorithms, similar in approach to Kruskal's algorithm for generating the minimum spanning tree of a graph (cf. Aho, Hopcroft, and Ullman [A1], pp. 172-176).

The ordering algorithm which locally minimizes storage is the minimum deficiency algorithm. At the k-th step of the algorithm, the remaining unordered vertex with smallest deficiency is ordered as x_k, and $G^{(k+1)}$ is computed as $G^{(k)}_{x_k}$. When several vertices all have smallest deficiency, a tie-breaking

strategy must be used to select one of the tied vertices, and different tie-breaking strategies lead to different minimum deficiency algorithms.

The major problem with the minimum deficiency algorithm is speed. At the k-th step it is necessary to examine several of the vertices in $X^{(k)}$ in order to find the one with smallest deficiency.† For each examined vertex x, each distinct pair of vertices u and v adjacent to x must be checked to see if the edge (u,v) is present in $G^{(k)}$. (There are $\sim Deg(x)^2/2$ such pairs, and $|Def(x)|$ is equal to the number of edges which are not present.) The computation at the k-th step is equivalent to computing the structure of $G_x^{(k)}$ for each examined vertex, and since most steps of the minimum deficiency algorithm examine more than one vertex, the overall cost of the algorithm is usually much greater than the cost of sparse Gaussian elimination using the resulting variable ordering (cf. Duff and Reid [D7]).

Locally minimizing the number of arithmetic operations, on the other hand, is much easier, and the minimum degree algorithm (cf. Rose [R4]) is used for this purpose. At the k-th step, x_k is chosen from among the vertices having minimum degree in $G^{(k)}$, and $G^{(k+1)}$ is computed as $G_{x_k}^{(k)}$. (The minimum degree algorithm actually locally minimizes work since minimizing $Deg(x)$ in $G^{(k)}$ is equivalent to minimizing $Deg(x)^2$ in $G^{(k)}$.) As with the minimum deficiency algorithm, a tie-breaking strategy must be used when

† By keeping track of the deficiencies of the vertices remaining as the algorithm progresses, it is possible to avoid examining every vertex in $X^{(k)}$.

several vertices in $X^{(k)}$ have minimum degree, and different versions of the minimum degree algorithm are obtained by varying the tie-breaking strategy.

At the k-th step of the minimum degree algorithm, the degrees of the vertices in $X^{(k)}$ must be examined to find the vertex of minimum degree, and $G^{(k+1)}$ must be computed. It is possible to avoid looking at all the remaining vertices at any step by keeping lists of the vertices having degree i, $1 \le i < N$, and updating the lists at the end of each step when the new elimination graph is computed. If this is done, then finding the vertex of minimum degree at the k-th step requires no searching, and the amount of work needed to execute the entire minimum degree algorithm is essentially equal to that needed to perform graph elimination on $G(A)$ with the computed ordering. Hence the minimum degree algorithm costs far less than the minimum deficiency algorithm.

We know of no general theoretical results concerning the quality of the local ordering algorithms discussed above. In large part this is due to the fact that the orderings produced are greatly affected by tie-breaking strategies which are difficult to analyze theoretically. Thus most of the results on minimum deficiency and minimum degree orderings have been obtained by testing them on a number of examples. It has been observed experimentally (cf. Duff and Reid [D7]) that the two algorithms produce almost equally good elimination orderings. Therefore, because of its lower cost, the minimum degree algorithm is usually preferred. That algorithm appears to work quite well in practice (cf. Woo, Roberts

Gustavson [W3]), although there are examples for which it produces elimination orderings which are arbitrarily worse than optimal orderings (cf. Duff and Reid [D7], Rose [R4]). In Section 4 we will look more closely at the quality of the minimum degree algorithm for the five- and nine-point model problems.

IV.3 Nested Dissection Orderings

In this section we consider the generation of good orderings for the five- and nine-point model problems and similar problems which are derived from an underlying planar mesh. For these special problems there are ordering algorithms based on the geometric properties of the mesh which can generate near-optimal orderings at very low cost. For the five- and nine-point model problems, these are the nested dissection algorithms (cf. Birkhoff and George [B3], George [G3], Rose and Whitten [R8], Woo, Roberts, and Gustavson [W3]). Most of the published results on mesh-based problems concern the two model problems, but recently Eisenstat [E1] has considered extensions of the divide-and-conquer principle of nested dissection to more general meshes in two and three dimensions.

In order to evaluate the absolute quality of an ordering, we must have lower bounds on the cost of sparse Gaussian elimination. We obtain this information for the model problems from the following theorem due to Hoffman, Martin, and Rose [H3]. (The bounds for the arithmetic work for the nine-point model problem were obtained independently by George [G3].) In fact, the result for the five-

point model problem implies the results for the nine-point model problem, since for any permutation matrix P, PA_5P^T is a submatrix of PA_9P^T.

Theorem 3.1: Let D_n be an $n \times n$ mesh, and let A be any symmetric, positive definite coefficient matrix corresponding to the use of either a five- or nine-point finite difference operator in D_n. Then the costs of factoring A with Algorithm III.2.3 satisfy

$$\theta_S \geq \Omega(n^2 \log n);$$

$$\theta_M \geq \Omega(n^3);$$

$$\theta_A \geq \Omega(n^3).$$

The proof for Theorem 3.1 given by Hoffman, Martin, and Rose [H3] is based on the graph-theoretic model for Gaussian elimination. Eisenstat [E1] has obtained a new proof of the theorem using an element model for Gaussian elimination, and his proof can be extended to cover the corresponding cases in both one and three dimensions (cf. Theorem 3.2).

Theorem 3.2: Let C_n be an $n \times n \times n$ cubic mesh in three dimensions, and let A be any symmetric, positive definite coefficient matrix corresponding to the use of either a seven- or twenty-seven-point finite difference operator in C_n (cf. Ames [A2], p. 253). Then the costs of factoring A with Algorithm III.2.3 satisfy

$\theta_S \geq \Omega(n^4)$;

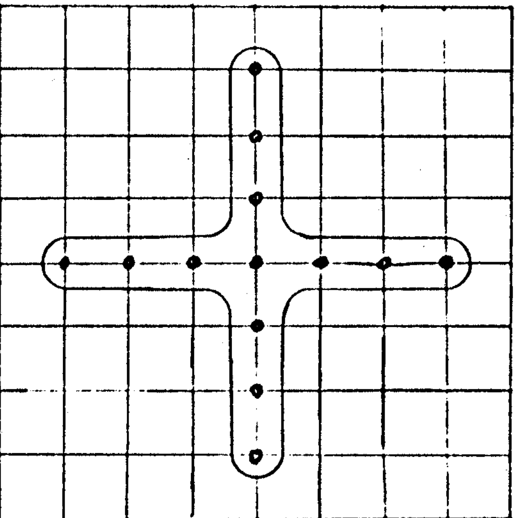
$\theta_M \geq \Omega(n^6)$;

$\theta_A \geq \Omega(n^6)$.

We now turn to near-optimal orderings for the model problems, considering first the nine-point model problem. For this problem, we present the breadth-first and depth-first nested dissection algorithms, two algorithms for generating a nine-point nested dissection ordering similar to that suggested by George [G3].†
The two algorithms are equivalent in the sense that the computed orderings give the same costs for sparse Gaussian elimination; but they are derived from different views of the ordering problem and lead to different element merging processes.

Let D_n be an $n \times n$ mesh. For convenience we assume that $n = 2^t - 1$ for some positiva integer t, but extension to other values of n is not difficult (cf. Duff, Erisman, and Reid [D6], Eisenstat [E2], Rose and Whitten [R8, R9]). Both of our nested dissection algorithms order the point of D_n in reverse order from $N = n^2$ to 1 by successively applying a basic ordering step to square submeshes of D_n. The basic step consists of first choosing a separating cross which divides the submesh into quadrants (see Figure 3.1a) and then ordering the vertices on the separating

† The ordering is identical to George's except in the detailed numbering of the vertices on the separating crosses (see Figure 3.1b). George's version gives slightly lower costs, but to present it here would obscure the underlying principles of the nested dissection process, which are independent of the detailed numbering.



Separating Cross in D_7

(a)

k

k-1

k-2

k-12 k-11 k-10 k-3 k-9 k-8 k-7

k-4

k-5

k-6

Ordering of Vertices in Separating Cross in D_7

(b)

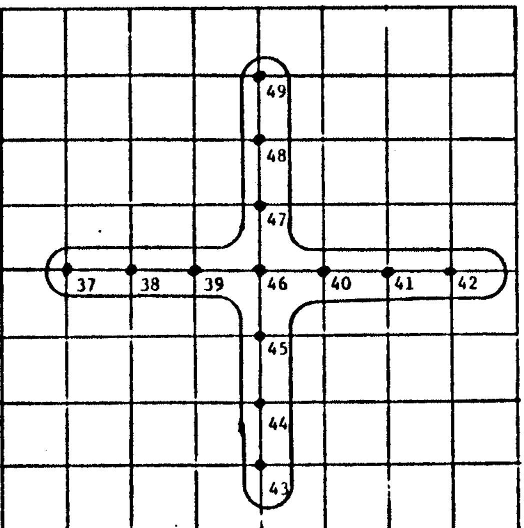
FIGURE 3.1

cross (see Figure 3.1b). If $x_{k+1}, x_{k+2}, \ldots, x_N$ have been determined, then the basic ordering step applied to an $m \times m$ submesh D_m orders as $x_{k-2m+2}, x_{k-2m+3}, \ldots, x_k$ the $2m-1$ vertices along the separating cross in D_m (see Figure 3.1). Essentially, the basic step applies a divide-and-conquer technique to D_m, i.e., it divides the mesh into identical $\frac{m-1}{2} \times \frac{m-1}{2}$ submeshes which are independent in the sense that elimination of variables corresponding to one of them cannot cause fill-in affecting the variables corresponding to another.

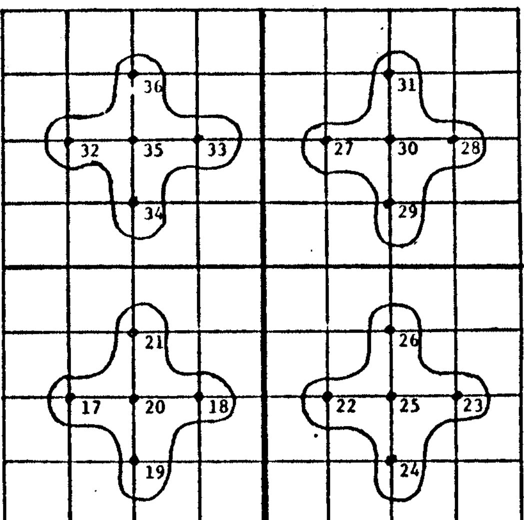
During any single step of either the breadth-first or depth-first nested dissection algorithm, the basic step is applied to either D_n or a submesh of D_n which was generated by a previous step. The two algorithms differ only in the order in which they process the submeshes; they both eventually generate the same set of separating crosses.

The breadth-first nested dissection algorithm follows the ideas of George [G3]. It starts by applying the basic step to D_n, thus dividing the mesh into four identical square $m \times m$ submeshes D_m with $m = 2^{t-1} - 1$ (see Figure 3.2a). The basic step is then applied to each of the generated D_m's in turn to obtain sixteen smaller, square submeshes $D_{m'}$ with $m' = 2^{t-2} - 1$ (see Figure 3.2b). The algorithm next subdivides each of these submeshes (see Figure 3.2c), and it continues in this way until all of the vertices in D_n have been numbered. Figure 3.3 illustrates this ordering for $n = 7$. The algorithm is called breadth-first nested dissection because the basic ordering step is applied to

(a)



(b)



Breadth-First Nested Dissection of D_7

FIGURE 3.2

(c)

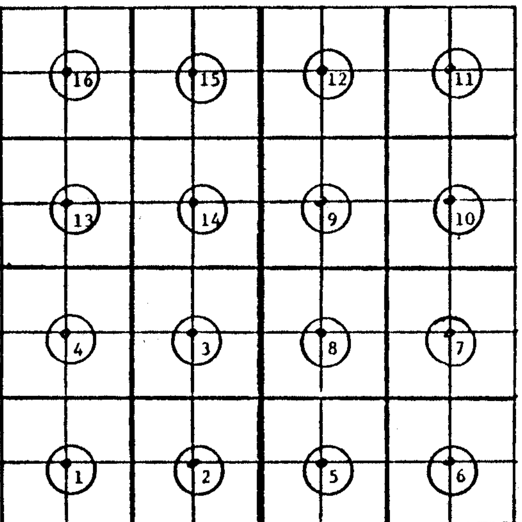
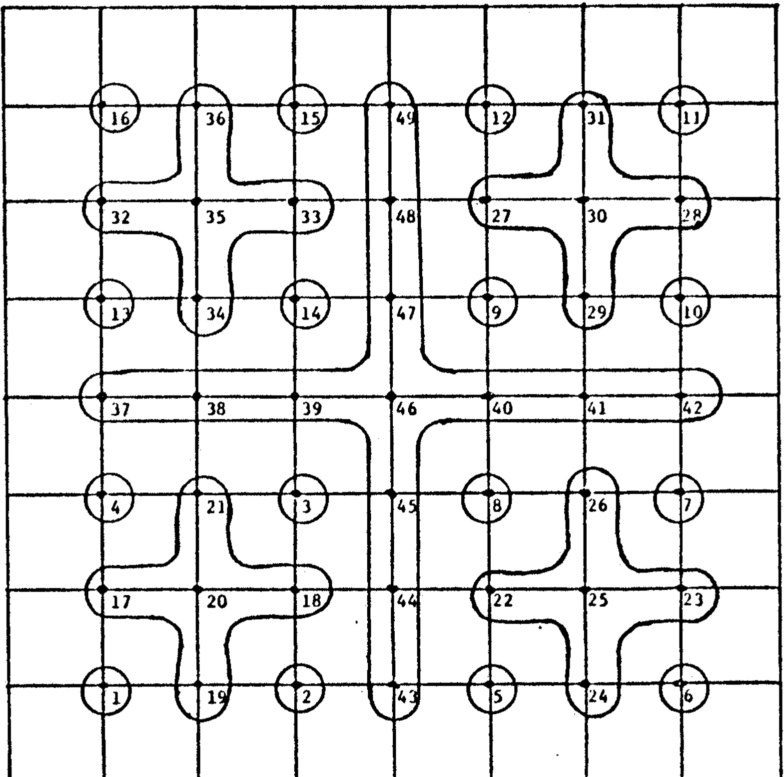


FIGURE 3.2 (continued)



Breadth-First Nested Dissection Ordering for D_7

FIGURE 3.3

all the submeshes of a given size before treating any smaller submeshes and because the ordering it generates corresponds to a breadth-first ordering of the merges in the element merge tree (see Figure 3.4).†

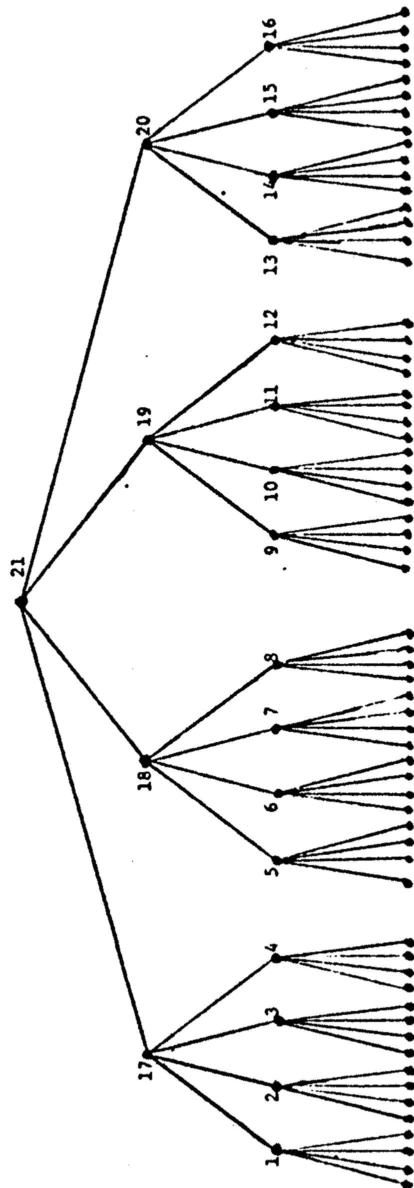
The depth-first nested dissection algorithm takes a recursive view of nested dissection as described by Rose and Whitten [R8]. At the first step it applies the basic step to D_n to obtain four identical m×m submeshes D_m with $m = 2^{t-1} - 1$ (see Figure 3.5a). Next, the basic step is applied to one of the generated D_m's to obtain four smaller, square submeshes $D_{m'}$ with $m' = 2^{t-2} - 1$ (see Figure 3.5b). At the third step one of these $D_{m'}$ is further subdivided (as in Figure 3.5c). The algorithm continues with this depth-first or recursive means of processing the submeshes to order the entire mesh D_n. Figure 3.5 illustrates this idea by showing the order in which the submeshes are processed, and Figure 3.6 shows the resulting ordering for n = 7. The algorithm is called depth-first nested dissection because it processes the generated submeshes in a depth-first or recursive way and because the ordering it generates corresponds to a depth-first ordering of the merges in the element merge tree (see Figure 3.7).††

Each separating cross generated by a nested dissection algorithm corresponds to an element merge in the compacted element

---

† I.e. the merges in the tree are performed level-by-level, so that the elements in the tree are numbered as in a breadth-first numbering of the tree.

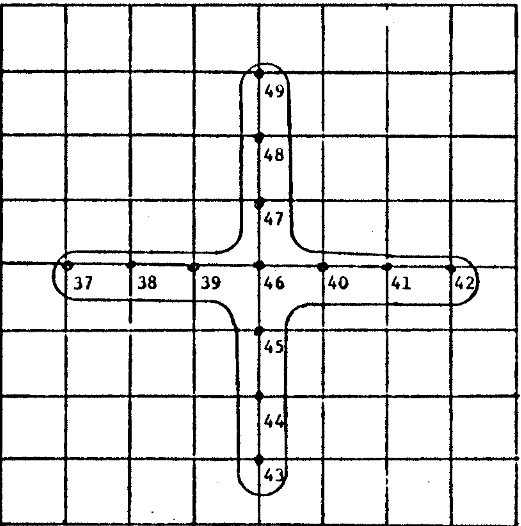
†† I.e. at any time the merge that is performed is as far to the top and left of the tree as possible, so that the elements in the tree are numbered as in a depth-first numbering of the tree.



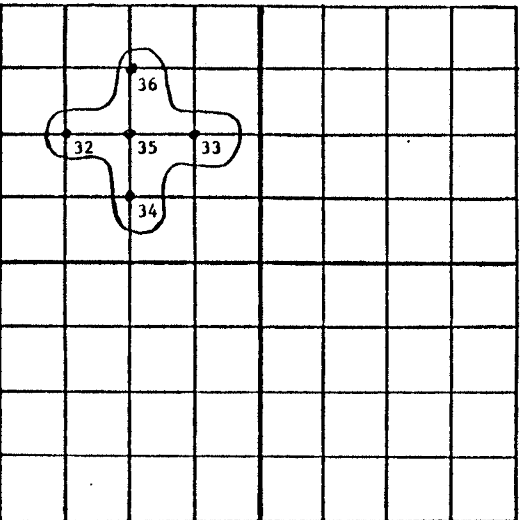
Ordering of the Merges in the Compacted Merge Tree for the Breadth-First Nested Dissection of D_7

FIGURE 3.4

(a)



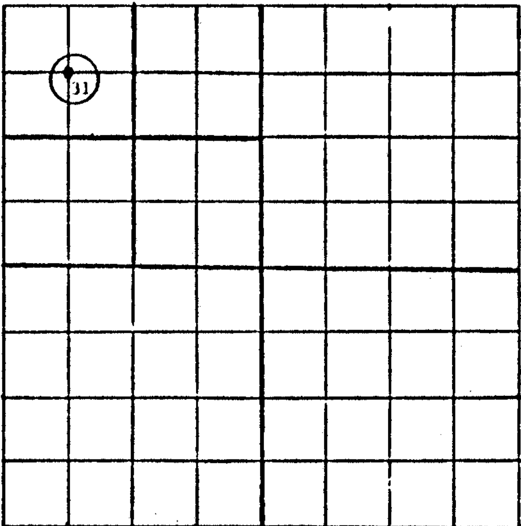
(b)



Depth-First Nested Dissection of D_7

FIGURE 3.5

(c)



(d)

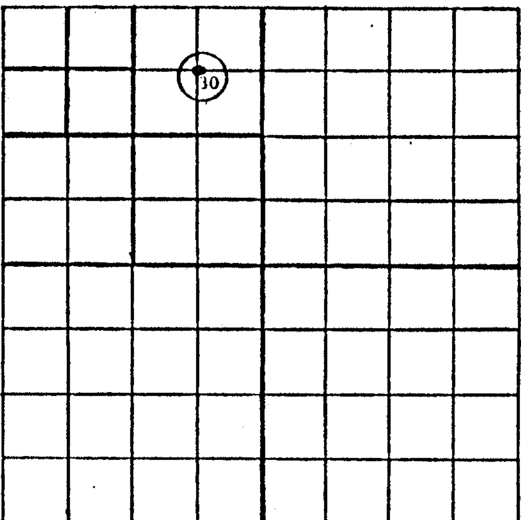
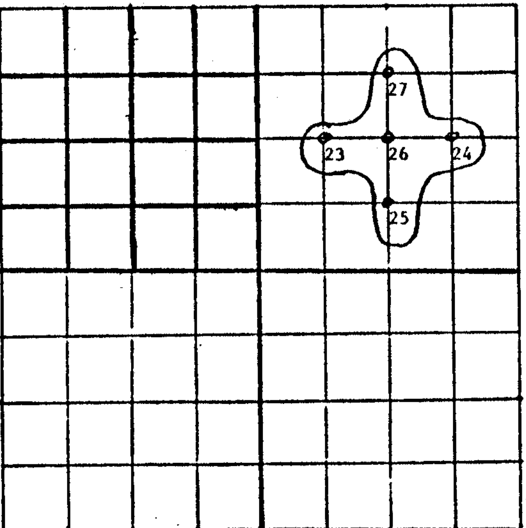


FIGURE 3.5 (continued)

(e)



(f)

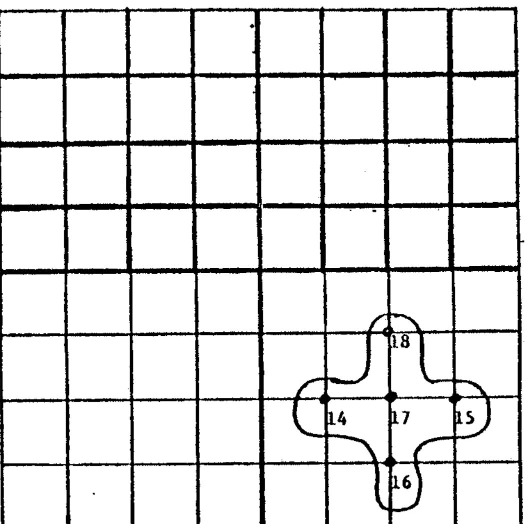
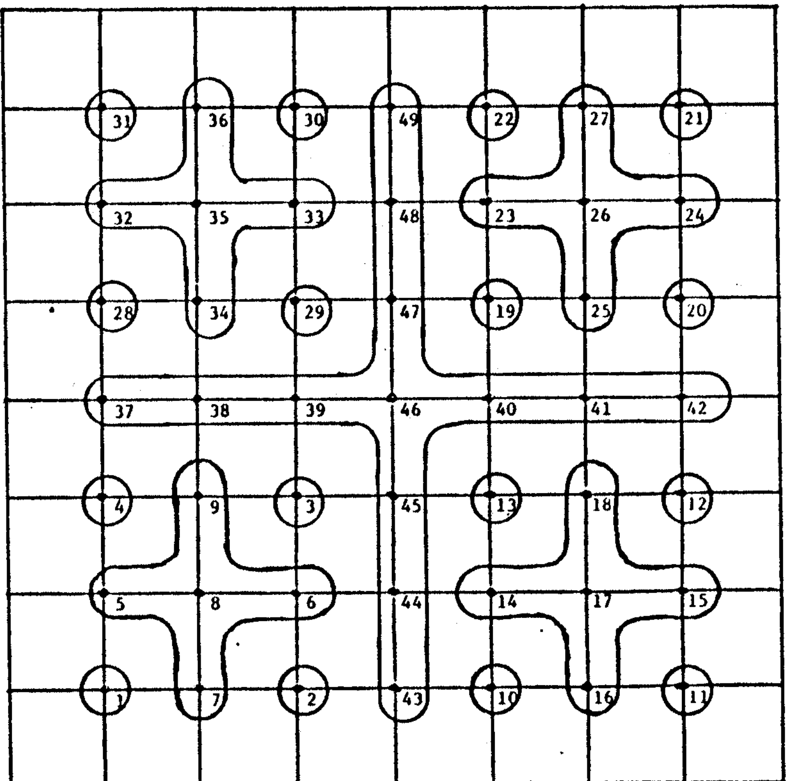
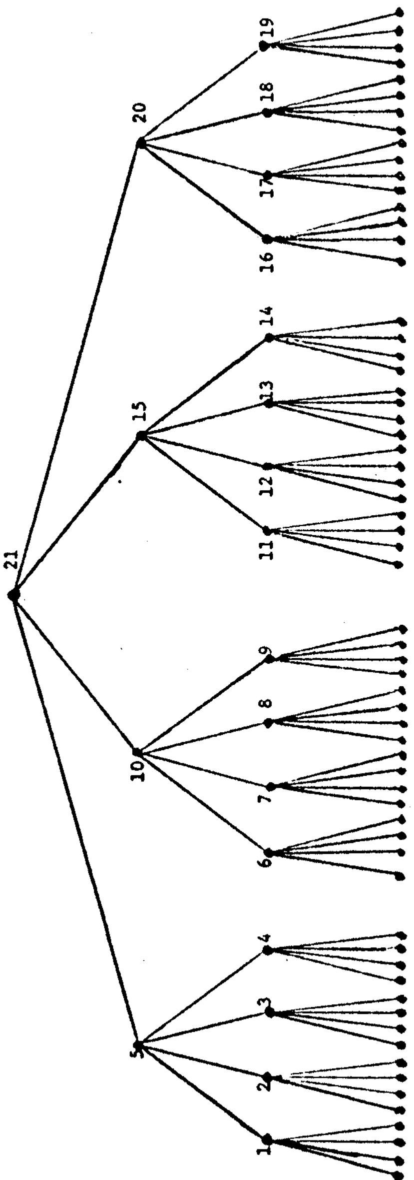


FIGURE 3.5 (continued)



Depth-First Nested Dissection Ordering for D_7

FIGURE 3.6



Ordering of the Merges in the Compacted Merge Tree for the Depth-First Nested Dissection of D_7

FIGURE 3.7

merge tree for the model problem. Since the breadth-first and depth-first algorithms find the same separating crosses in different orders, we see that the associated compacted element merge trees must have identical structures. Furthermore, the cost of sparse symmetric factorization is reflected in the structure of the element merge tree, not in the order in which the compacted merges are performed. Hence the two nested dissection orderings are equivalent in that they lead to sparse Gaussian elimination processes having identical costs. It then follows from Theorem 3.3 (due to George [G3]) that both are near-optimal.

<u>Theorem 3.3:</u> Let D_n be an $n{\times}n$ mesh ordered with the breadth-first nested dissection algorithm using an optimal numbering of the vertices on each separating cross. Let A be a symmetric, positive definite coefficient matrix corresponding to the use of a nine-point finite difference operator on the ordered mesh D_n. Then the costs of factoring A with Algorithm III.2.3 are

$$\theta_S \approx \frac{31}{4}n^2\log n - \frac{73}{3}n^2 + O(n\log n);$$

$$\theta_M \approx \frac{267}{28}n^3 - 17\,n^2\log n + O(n^2);$$

$$\theta_A \approx \frac{267}{28}n^3 - \frac{99}{4}n^2\log n + O(n^2).$$

Since A_5 is a submatrix of A_9, Theorem 3.3 certainly gives upper bounds on the costs of symmetric Gaussian elimination for the five-point model problem using a nine-point nested dissection ordering. Hence the breadth-first and depth-first nested

dissection orderings are also near-optimal orderings for the five-point model problem. However, a somewhat better elimination ordering for the five-point model problem is the diagonal nested dissection ordering proposed by Birkhoff and George [B3] (also cf. Woo, Roberts, and Gustavson [W3]). They derive this ordering in a complex manner akin to performing nested dissection using diagonal, rather than vertical and horizontal separating crosses. Their version does not appear to lead to a very convenient method for generating the ordering for arbitrary values of n, so we present instead a simple algorithm which generates a slightly different (and actually slightly better) diagonal nested dissection ordering.

Let D_n be an $n \times n$ mesh. We assume that n is odd, although the ordering can be generated as easily for even values of n. For $p = \frac{n^2+1}{2}$, we define $x_1, x_2, \ldots, x_p$ to be the points on the alternate diagonals of D_n, starting at a corner. The particular ordering of these p points does not matter, but a simple one, known as the alternate diagonal or red-black ordering (cf. Price and Coats [P2], Woo, Roberts, and Gustavson [W3]), is shown in Figure 3.8a. Figure 3.8b shows the element diagram corresponding to the graph $G^{(p+1)}$ obtained by eliminating the first p vertices from $G^{(1)} = D_n$. Noting that it depicts a diamond shaped mesh which has an element structure similar to that of a nine-point mesh, we embed it in an $(n-1)\times(n-1)$ nine-point mesh D' as shown in Figure 3.8c. D' is ordered with a nine-point nested dissection algorithm, inducing a relative ordering

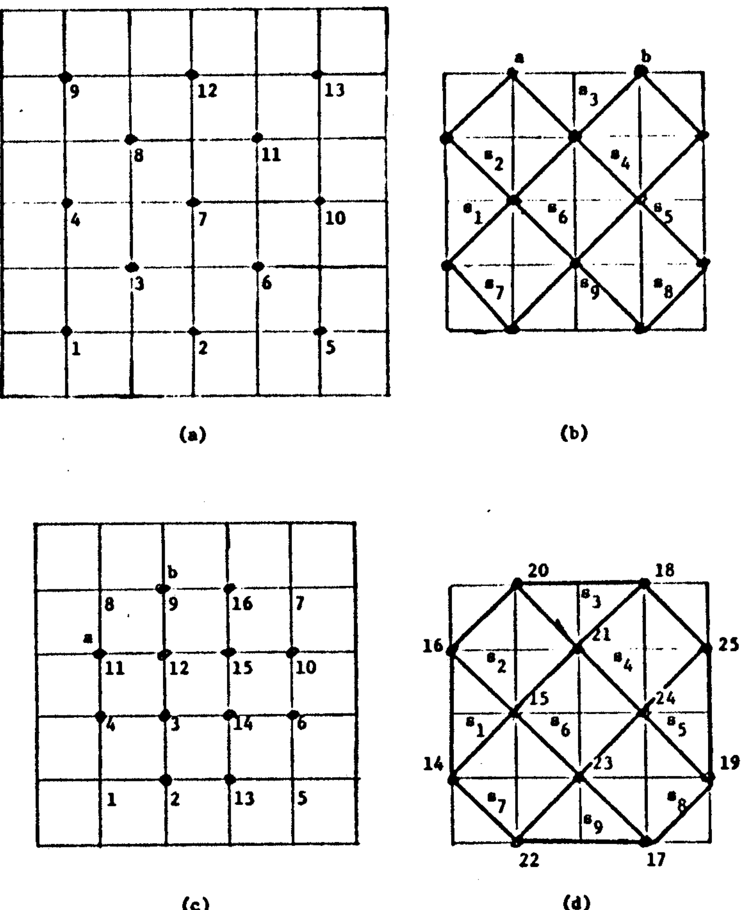
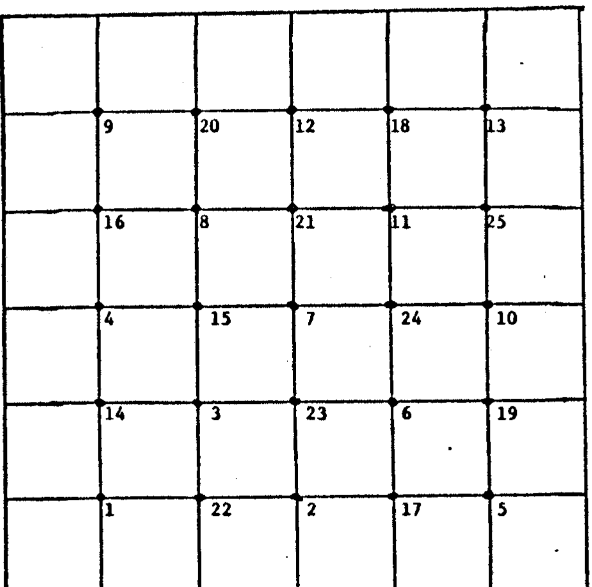


FIGURE 3.8



(e)

FIGURE 3.8 (continued)

on the points of D' which correspond to vertices in $G^{(p+1)}$ (see Figure 3.8c). We then order these vertices starting with x_{p+1}, putting them in the same relative order as the corresponding points in D' (see Figure 3.8d). Figure 3.8e shows the final ordering which is obtained in this way.

Birkhoff and George [B3] and Woo, Roberts, and Gustavson [W3] have given the following theorem for the diagonal nested dissection ordering developed by Birkhoff and George [B3]. Although we will not prove it, a similar result holds for the ordering presented here.

<u>Theorem 3.4</u>: Let D_n be an $n \times n$ mesh ordered with a diagonal nested dissection ordering, and let A be a symmetric, positive definite coefficient matrix corresponding to the use of a five-point finite difference operator in the ordered mesh D_n. Then the costs of factoring A with Algorithm III.2.3 are

$$\theta_S \approx \frac{31}{8} n^2 \log n - \frac{67}{8} n^2 + O(n \log n);$$

$$\theta_M \approx \frac{251}{48} n^3 - \frac{17}{2} n^2 \log n + O(n^2);$$

$$\theta_A \approx \frac{251}{48} n^3 - \frac{99}{8} n^2 \log n + O(n^2).$$

The extension of either ordinary or diagonal nested dissection orderings to three dimensions and to more general meshes in the plane can be accomplished by generalizing the basic step to apply the divide-and-conquer technique to arbitrary geometric regions in two or three dimensions. (For example, in an $n \times n \times n$

cubic mesh C_n, three mutually perpendicular planes of mesh points could be used to divide the mesh into eight $\frac{n-1}{2} \times \frac{n-1}{2} \times \frac{n-1}{2}$ submeshes.) Eisenstat has studied the general orderings which result from such divide-and-conquer strategies, and he has proved that they are always near-optimal (cf. Eisenstat [E1]).

IV.4 Orderings for the Model Problems

George [G4] has pointed out that the nested dissection orderings presented in Section 3 are actually minimum degree orderings with appropriate tie-breaking strategies. Because of this and the popularity of minimum degree orderings, several experimental studies have compared the minimum degree and nested dissection orderings for the five- and nine-point model problems (cf. Duff, Erisman, and Reid [D6], Woo, Roberts, and Gustavson [W3]). The results have been inconclusive, but George [G4] has conjectured that all minimum degree orderings are near-optimal for these problems. If it should be true, the conjecture would give a firm foundation to the widespread use of minimum degree orderings for problems derived from mesh discretizations of geometric regions in two or three dimensions. To this date, minimum degree orderings have been used because of the difficulty of applying the divide-and-conquer ideas to arbitrary irregular regions, not because they are known a priori to be good orderings.

In this section we present the results of a series of ex-

periments performed jointly with Eisenstat. Unfortunately, our results are rather inconclusive, since we have not been able to obtain results for large enough values of n.

Our results for the minimum degree ordering were fairly sensitive to the initial ordering of the mesh points, since our minimum degree algorithm employed a more-or-less random tie-breaking strategy dependent on the initial ordering and the mechanics of implementation. We found that the alternate diagonal ordering was the best initial ordering for the five-point model problem, while the natural ordering was best for the nine-point model problem. Using these initial orderings, we compared minimum degree with nine-point nested dissection for the nine-point model problem and with our version of diagonal nested dissection for the five-point model problem (cf. Section 3 for a description of these nested dissection orderings). The resulting storage costs θ_S and arithmetic costs θ_M ($\approx \theta_A$) are shown in Tables 4.1 and 4.2.

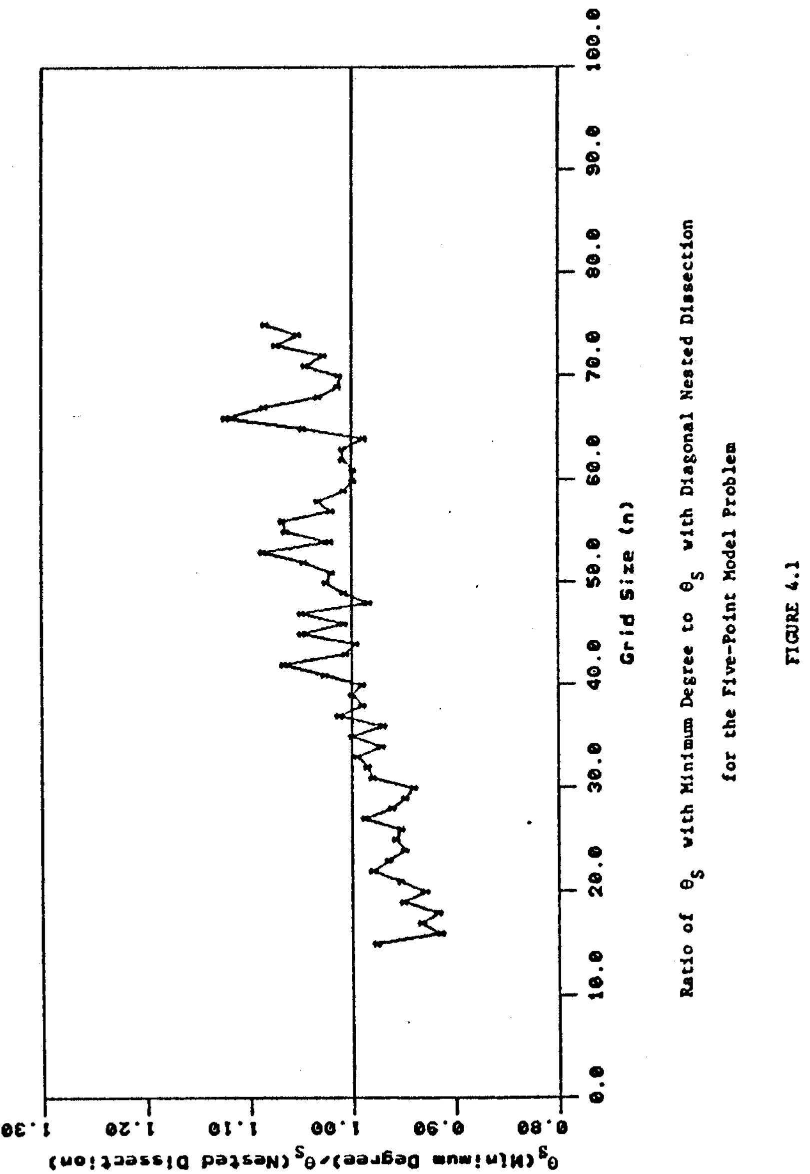
To check for the near-optimality of the minimum degree orderings, we divided the costs with minimum degree by the costs with nested dissection and plotted the ratios (see Figures 4.1 - 4.4). The convergence of these ratios to a constant limit would indicate the near-optimality of minimum degree, since nested dissection is near-optimal. Unfortunately, however, it is unclear as to whether or not the ratios are converging, so our results neither prove nor disprove George's conjecture. On the other hand, the ratios are always less than two, and for five-point model problems with

n	Minimum Degree		Diagonal Nested Dissection	
	θ_S	θ_M	θ_S	θ_M
15	1802	10706	1845	10872
16	2047	11893	2239	14445
20	3613	25271	3890	29851
25	6341	53844	6620	58574
30	9815	93738	10444	109263
31	10880	112237	11105	116597
32	12031	130869	12207	134475
35	14868	176122	14871	171684
40	20679	271389	20885	273483
45	28553	439338	27234	383116
50	36148	578668	35287	546560
55	46583	837406	43797	722282
60	53960	930906	54007	966202
63	60669	1095335	60141	1104982
64	62313	1124122	63027	1187143
65	67998	1317641	64884	1222058
70	78428	1538362	77494	1562798
75	98070	2339382	90475	1903173

Comparison of Minimum Degree and Diagonal Nested Dissection

for the Five-Point Model Problem

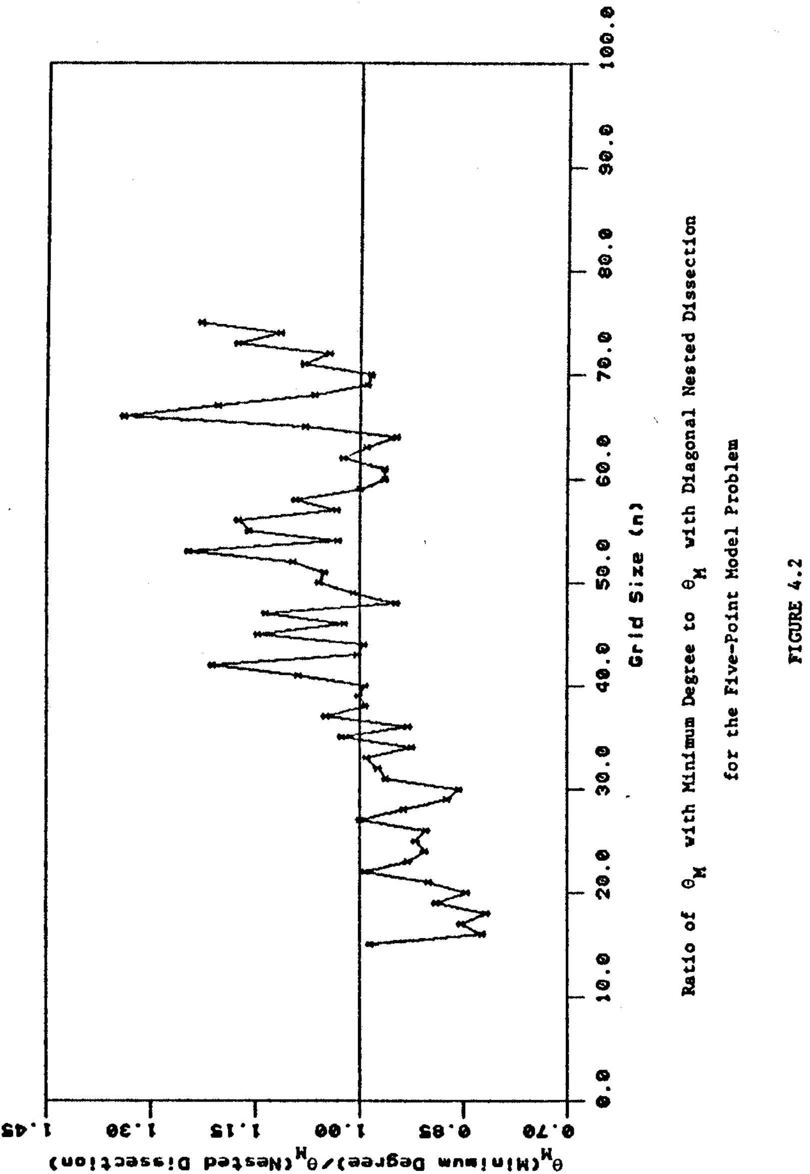
TABLE 4.1



Ratio of θ_S with Minimum Degree to θ_S with Diagonal Nested Dissection

for the Five-Point Model Problem

FIGURE 4.1



Ratio of θ_M with Minimum Degree to θ_M with Diagonal Nested Dissection

for the Five-Point Model Problem

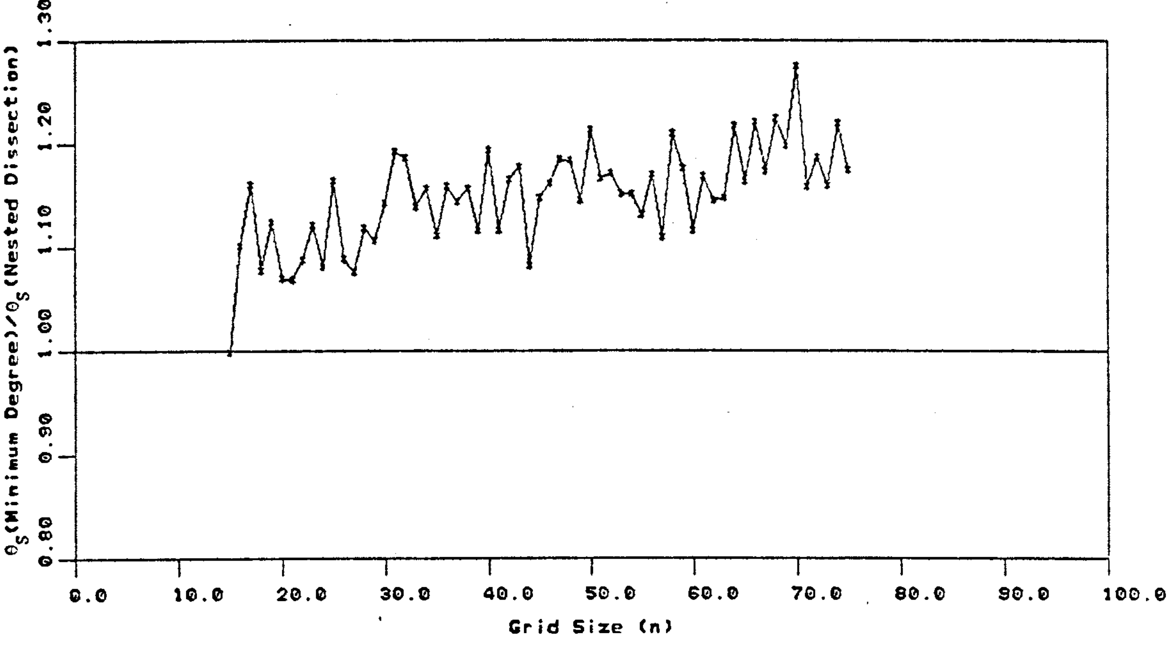
FIGURE 4.2

n	Minimum Degree		Nested Dissection	
	θ_S	θ_M	θ_S	θ_M
15	2774	21634	2778	21327
16	3533	33065	3209	25259
20	6174	65053	5768	53616
25	11830	168273	10169	109728
30	18496	289958	16195	201905
31	21056	371274	17666	227351
32	22377	383724	18855	245346
35	26254	426053	23614	330777
40	39384	831704	32998	504062
45	50595	1092173	44069	731764
50	68777	1735611	56676	1014984
55	81267	1926915	71826	1392127
60	98329	2435689	88063	1806903
63	114057	3121037	99450	2127959
64	124806	3688660	102517	2206085
65	123220	3413619	105841	2299096
70	162060	5844642	127167	2927638
75	176834	5764890	150430	3643881

Comparison of Minimum Degree and Nested Dissection

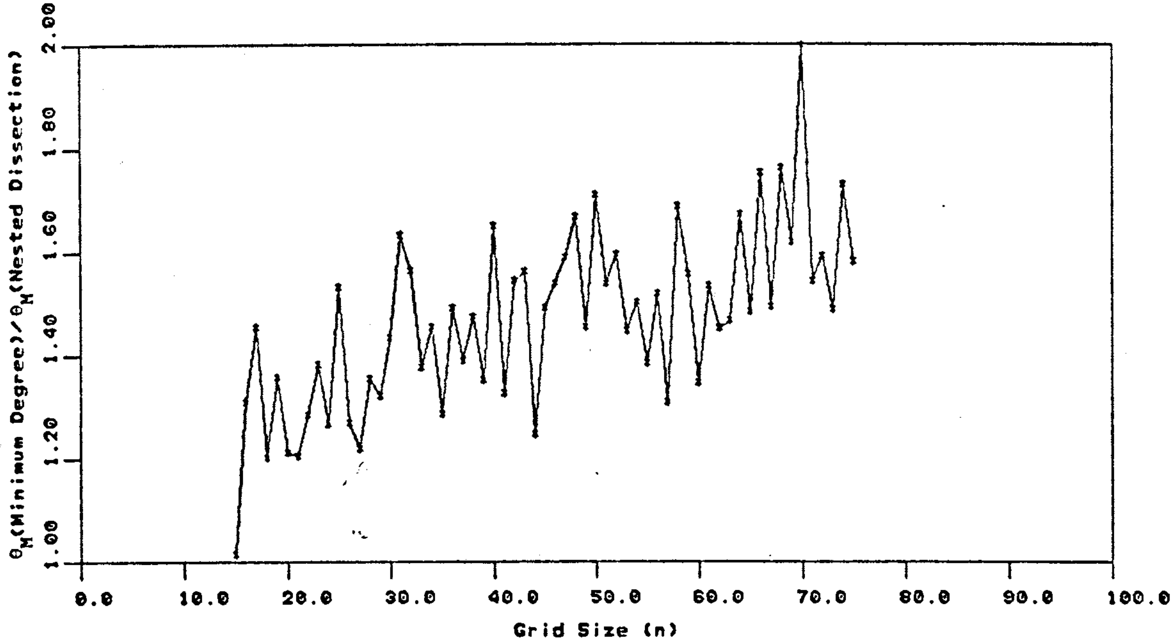
for the Nine-Point Model Problem

TABLE 4.2



Ratio of Θ_S with Minimum Degree to Θ_S with Nested Dissection
for the Nine-Point Model Problem

FIGURE 4.3



Ratio of Θ_M with Minimum Degree to Θ_M with Nested Dissection
for the Nine-Point Model Problem

FIGURE 4.4

$n < 50$, they are quite near one. Thus it appears that minimum degree may be a reasonable alternative in practice, especially for small to moderate size problems using five-point approximations (cf. Woo, Roberts, and Gustavson [W3]).

CHAPTER V: IMPLEMENTATION OF SPARSE SYMMETRIC FACTORIZATION

V.1 Introduction

Algorithm III.2.3 is a row-oriented algorithm for sparse symmetric factorization which can be used to factor the $N \times N$ symmetric, positive definite coefficient matrix of the system

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

In this chapter we consider the practical aspects of that algorithm and describe an efficient implementation for it (cf. Eisenstat, Schultz, and Sherman [E6]). Our presentation emphasizes the underlying principles of the implementation, not the specific details required to write a computer program. (For a more detailed discussion and a program, cf. Eisenstat, Schultz, and Sherman [E7].)

The basic goal of any implementation is to factor A into the product $U^T D U$ without storing or operating on the zeroes in A and U. To do this requires a certain amount of overhead, i.e., extra storage for pointers in addition to that needed for the nonzeroes in A and U, and extra non-numeric operations in addition to the required arithmetic operations. In the implemen-

tation presented here the amount of overhead is fairly small in practice (cf. Chapter VI, Eisenstat, Schultz, and Sherman [E3], Woo, Eisenstat, Schultz, and Sherman [W2]), although the exact amount depends on the particular linear system (1.1).

Following the ideas of Chang [C2] and Gustavson [G7], we break the implementation of Algorithm III.2.3 into two parts: symbolic factorization and numeric factorization. The symbolic factorization determines the structure of the rows of U from the known structure of the rows of A, and the numeric factorization uses the structure information to efficiently compute the U^TDU factorization of A. By splitting up the computation in this way we gain important flexibility. If several systems have coefficient matrices with identical zero structure but different nonzero entries, then only one symbolic factorization with separate numeric factorizations is required to solve them. (Such a situation might arise in the use of Newton's method to solve a system of nonlinear equations, cf. Chapter VIII.)

In Section 2 we present the symbolic factorization algorithm and show that it runs in $O(\theta_S)$ time. In Section 3 we present the numeric factorization algorithm and show that it runs in $O(\theta_A)$ time. Finally, in Section 4 we describe the data structures which are used to store sparse symmetric matrices efficiently.

Our implementation differs significantly from previous implementations (e.g. Gustavson [G7]) in the symbolic factorization algorithm and the storage scheme used for U. By exploiting the particular way in which U fills in, we are able to reduce the

time required for the symbolic factorization from $O(\theta_A)$ to $O(\theta_S)$,† and by taking advantage of certain regularity of structure in U, we are able to substantially reduce the storage overhead. Although the storage scheme may require as many as θ_S pointers to store U, it often requires many fewer. For instance, for the model nine-point problem, only $O(n^2)$ pointers are required to store U, even though $\theta_S \geq \Omega(n^2 \log n)$.

V.2 Symbolic Factorization

In this section we present the symbolic factorization algorithm and analyze its cost. As in Section III.2,

ra_k denotes the set of column indices $j \geq k$ for which $a_{kj} \neq 0$,

ru_k denotes the set of column indices $j > k$ for which $u_{kj} \neq 0$, and

cu_k denotes the set of row indices $i < k$ for which $u_{ik} \neq 0$.

The sets $\{ra_k\}$, which describe the structure of A, are input parameters, and the symbolic factorization algorithm computes the sets $\{ru_k\}$ from them. The sets $\{cu_k\}$ could be computed from the sets $\{ru_k\}$ (see Section V.3), but as we shall see later, they are not actually needed for symbolic factorization.

† Rose and Tarjan [R6] have developed a similar technique for use in computing the fill-in which occurs in symmetric Gaussian elimination.

At the k-th step of the symbolic factorization algorithm, ru_k is computed from ra_k and the sets ru_i for $i < k$. An examination of Algorithm III.2.3 shows that for $j > k$, $u_{kj} \neq 0$ if and only if either

(i) $a_{kj} \neq 0$ or

(ii) $u_{ij} \neq 0$ for some $i \in cu_k$.

Thus letting

$$ru_i^k = \{j \in ru_i : j > k\},$$

we have $j \in ru_k$ if and only if either

(iii) $j \in ra_k$ or

(iv) $j \in ru_i^k$ for some $i \in cu_k$.

Algorithm 2.1 is a symbolic factorization algorithm based directly on Algorithm III.2.3. At the k-th step, ru_k is formed by combining ra_k with the sets $\{ru_i^k\}$ for $i \in cu_k$. However, it is not necessary to examine ru_i^k for all rows $i \in cu_k$. Let ℓ_k be the set of rows $i \in cu_k$ for which k is the minimum column index in ru_i. Then we have the following result which expresses a type of "transitivity" condition for the fill-in in symmetric Gaussian elimination. The import of the theorem is that in forming ru_k, it suffices to examine ru_i^k for $i \in \ell_k$ (cf. Rose and Tarjan [R6] for a similar result).

<u>Theorem 2.1</u>: Let $i \in cu_k$. Then either $i \in \ell_k$ or there is an m_p, $i < m_p < k$, such that $m_p \in \ell_k$ and $ru_i^k \subseteq ru_{m_p}^k$.


```
line
 1   For  k ← 1 to N  do
 2      [ru_k ← ∅];
 3   For  k ← 1 to N-1  do
     Comment:  Form  ru_k  by set unions
 4      [For  j ∈ {n ∈ ra_k: n > k}  do
 5         [ru_k ← ru_k ∪ {j}];
 6       For  i ∈ cu_k  do
 7          [For  j ∈ ru_i^k  do
 8             [If  j ∉ ru_k  then
 9                 [ru_k ← ru_k ∪ {j}]]]];

 O(θ_A)  Symbolic Factorization Algorithm
```

ALGORITHM 2.1

```
line
 1   For  k ← 1 to N  do
 2      [ru_k ← ∅;
 3       ℓ_k ← ∅];
 4   For  k ← 1 to N-1  do
     Comment:  Form  ru_k  by set unions
 5      [For  j ∈ {n ∈ ra_k: n > k}  do
 6         [ru_k ← ru_k ∪ {j}];
 7       For  i ∈ ℓ_k  do
 8          [For  j ∈ ru_i^k  do
 9             [If  j ∉ ru_k  then
10                 [ru_k ← ru_k ∪ {j}]]];
11       m ← min {j: j ∈ ru_k ∪ {N+1}};
12       If  m < N+1  then
13          [ℓ_m ← ℓ_m ∪ {k}]];

 O(θ_S)  Symbolic Factorization Algorithm
```

ALGORITHM 2.2

<u>Proof</u>: Assume that $i \notin \ell_k$. We define a sequence $M = \{m_0, m_1, m_2, \ldots\}$ by $m_0 = i$ and

$$m_{n+1} = \min \{j \in ru_{m_n}\}.$$

The sequence terminates if ru_{m_n} is empty for some n, and it is clear that this must occur, since $m_n < m_{n+1} < N$. We now let m_p denote the last element of M which satisfies $m_p < k$ and consider only those $m_n < m_p < k$. Certainly $m_n \in \ell_{m_{n+1}} \subseteq cu_{m_{n+1}}$, and hence from (iv) above, it follows that

$ru_{m_n}^{m_p} \subseteq ru_{m_{n+1}}^{m_p}$ and $ru_{m_n}^{k} \subseteq ru_{m_{n+1}}^{k}$. Then since $m_{n+1} \leq m_p < k$, we have by transitivity that

$$ru_i^{m_p} = ru_{m_0}^{m_p} \subseteq ru_{m_p}^{m_p} = ru_{m_p}$$

and

$$ru_i^k = ru_{m_0}^k \subseteq ru_{m_p}^k.$$

Furthermore, $k \in ru_i^{m_p}$, so $k \in ru_{m_p}$. Thus, since $m_{p+1} \geq k$, we must have $m_{p+1} = k$ and $m_p \in \ell_k$. □

Using the theorem we see that $j \in ru_k$ if and only if either

(v) $j \in ra_k$ or

(vi) $j \in ru_i^k$ for $i \in \ell_k$.

Algorithm 2.2 is a symbolic factorization algorithm based on this observation. The sets $\{\ell_k\}$ are formed during the execution of

the algorithm by adding each row k to the proper set ℓ_m as soon as ru_k has been computed. Since each row is a member of at most one set ℓ_k, the sets $\{\ell_k\}$ combined can have at most N entries, and only $O(N)$ storage locations are required to store them.

The cost of Algorithm 2.2 is determined by the costs of its innermost loop (lines 7-10) and of lines 5-6 and 11-13, since the remainder of the algorithm requires only $O(N)$ operations. We will analyze the cost of the algorithm by separately analyzing the time spent in each of these program segments. At the k-th step, for $i \in \ell_k$, lines 9-10 are executed once for each entry $j \in ru_i^k$, and since each row i is a member of at most one set ℓ_k, lines 9-10 can be executed overall at most once for each entry of the sets $\{ru_i\}$ combined (i.e. once for each nonzero entry in U). If the characteristic vector of ru_k (cf. Aho, Hopcroft, and Ullman [A1], p. 49) is computed along with ru_k, then the test in line 9 requires constant time, since it can be done by examining one entry of the characteristic vector. Furthermore, the union operation in line 10 also requires only constant time since j is simply appended to the end of ru_k and the j-th entry of the characteristic vector is set to 1. Hence we see that a total of $O(\theta_S)$ total time is spent in the innermost loop of the algorithm.

At the k-th step, line 6 is executed once for each entry in ra_k, so that overall, it can be executed just once for each entry of the sets $\{ra_k\}$ combined (i.e. once for each nonzero entry of

A). Again the union operation requires only constant time, and since $\sum_{k=1}^{N} |ra_k| \le \theta_S$, the algorithm can spend a total of at most $O(\theta_S)$ time in lines 5-6.

Finally, we note that for each k, lines 11-13 require $O(|ru_k|)$ time, since the minimum operation of line 11 requires $O(|ru_k|)$ time, while the union operation of line 13 requires constant time. Thus overall, lines 11-13 require $O(\theta_S)$ time, and, combining this with our analyses of lines 5-6 and 7-10, we see that the entire algorithm requires only $O(\theta_S)$ time.

In the next two sections, we will find it necessary to have each set ru_k ordered in increasing order. There are two ways to obtain the ordered versions of the sets $\{ru_k\}$. One is to modify Algorithm 2.2 to compute ordered sets $\{ru_k\}$ directly. This requires changing the set union operations to list insertion operations so that each new entry is inserted in its proper place in the ordered set ru_k at each stage. Unfortunately, however, the time required for a list insertion operation depends on the length of the list into which the insertion is made, so that except in special cases (like the five- and nine-point model problems), the modified algorithm will require more than $O(\theta_S)$ time.

The alternative means of obtaining ordered sets $\{ru_k\}$ is to first compute the unordered sets and then to sort them. If we view the set $R = \bigcup_{k=1}^{N} ru_k$ as a set of ordered pairs (k,j) where $j \in ru_k$, we wish to order R in lexicographic order, i.e.,

so that (k,j) precedes (k',j') in R if and only if either

(i) $k < k'$ or

(ii) $k = k'$ and $j < j'$.

Since all the entries of the sets $\{ru_k\}$ are integers between 1 and N, this sorting operation can be accomplished efficiently by using a lexicographic sort (cf. Aho, Hopcroft, and Ullman [A1], pp. 77-84). This requires time proportional to the number of entries in the combined sets, i.e., $O(\theta_S)$ time, and if we combine the lexicographic sort with Algorithm 2.2, we obtain an algorithm for computing the ordered sets $\{ru_k\}$ in $O(\theta_S)$ time (cf. Rose and Whitten [R9]).†

V.3 Numeric Factorization

In this section we consider the numeric factorization process. After the symbolic factorization has been completed, the sets $\{ra_k\}$ and $\{ru_k\}$†† are available, and the only difficulty in the straightforward implementation of Algorithm III.2.3 is the generation of the sets $\{cu_k\}$. If we were willing to use θ_S

---

† Surprisingly, however, our experiments have shown that, in practice, it may often be faster to compute the ordered sets directly with list insertions. This is especially true for linear systems arising in the use of finite difference or finite element methods for the solution of partial differential equations, since it can be proved that computing the ordered sets directly requires only $O(\theta_S)$ time for such systems.

†† As noted in Section 2, we will assume that we have ordered sets $\{ru_k\}$.

storage locations for the sets $\{cu_k\}$, we could pre-compute them efficiently by sorting the sets $\{ru_k\}$ with a lexicographic sort. However, it is often desirable to reduce the storage as much as possible, so we present an implementation which uses only $O(N)$ storage locations to compute the information contained in the sets $\{cu_k\}$ as it is needed.

Algorithm 3.1 is a numeric U^TDU factorization algorithm based on Algorithm III.2.3 which uses the sets $\{p_k\}$ to compute the sets $\{cu_k\}$ as it proceeds. The sets $\{p_k\}$ are generated in such a way that during the k-th step, $p_k = cu_k$ and for $j > k$, p_j contains those indices $i < k$ which are in cu_j but not in cu_ℓ for $k \le \ell < j$. Thus during the k-th step, each row i can be in at most one set p_j $(j \ge k)$, and only $O(N)$ locations are required to store the sets $\{p_j\colon j \ge k\}$, since these sets combined can have a total of at most N entries.

The computation performed during the k-th step of Algorithm 3.1 consists of two parts: arithmetic operations identical to those of Algorithm III.2.3 and updating operations for the sets $\{p_k\}$. Since we have discussed the arithmetic operations in Section III.2, we will consider only the set-updating operations here.

During the k-th step we require that $p_k = cu_k$ and that for $j > k$, p_j contains those row indices $i < k$ such that $i \in cu_j$ and j is the minimum column index in ru_i^{k-1}. We now use an inductive illustration to describe the way in which the sets $\{p_k\}$ must be updated so that they satisfy this condition (see Figures 3.1 and 3.2). As shown in Figure 3.1a, all the sets $\{p_k\}$ are

line	
1	For $k \leftarrow 1$ to N do
2	$\quad [p_k \leftarrow \emptyset];$
3	For $k \leftarrow 1$ to N do
4	$\quad [m_{kk} \leftarrow 1;$
5	$\quad d_{kk} \leftarrow a_{kk};$
6	$\quad$ For $j \in ru_k$ do
7	$\qquad [m_{kj} \leftarrow 0];$
8	$\quad$ For $j \in \{n \in ra_k\colon n > k\}$ do
9	$\qquad [m_{kj} \leftarrow a_{kj}];$
10	$\quad$ For $i \in p_k$ do
11	$\qquad [t \leftarrow m_{ik};$
12	$\qquad m_{ik} \leftarrow m_{ik}/d_{ii};$
13	$\qquad d_{kk} \leftarrow d_{kk} - t \cdot m_{ik};$
14	$\qquad$ For $j \in ru_i^k$ do
15	$\qquad\quad [m_{kj} \leftarrow m_{kj} - m_{ik} \cdot m_{ij}]];$
16	$\quad$ For $i \in p_k$ do
17	$\qquad [p_k \leftarrow p_k - \{i\};$
18	$\qquad j \leftarrow \min\{m\colon m \in ru_i^k \cup \{N+1\}\};$
19	$\qquad$ If $j < N+1$ then
20	$\qquad\quad [p_j \leftarrow p_j \cup \{i\}]];$
21	$\quad j \leftarrow \min\{m\colon m \in ru_k \cup \{N+1\}\};$
22	$\quad$ If $j < N+1$ then
23	$\qquad [p_j \leftarrow p_j \cup \{k\}]];$
	<u>Comment</u>: Now $M = U$

Numeric Factorization Algorithm

ALGORITHM 3.1

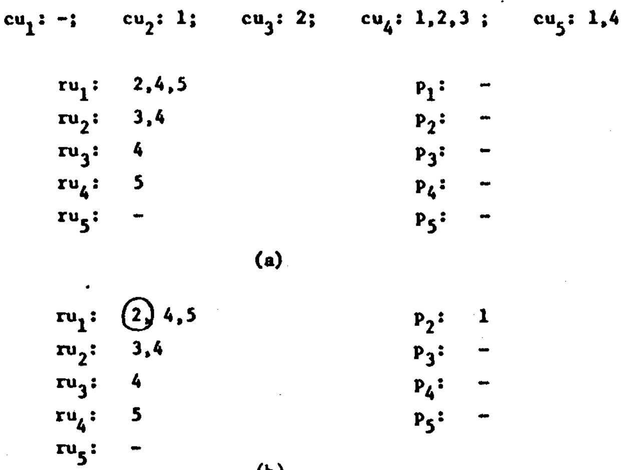


FIGURE 3.1

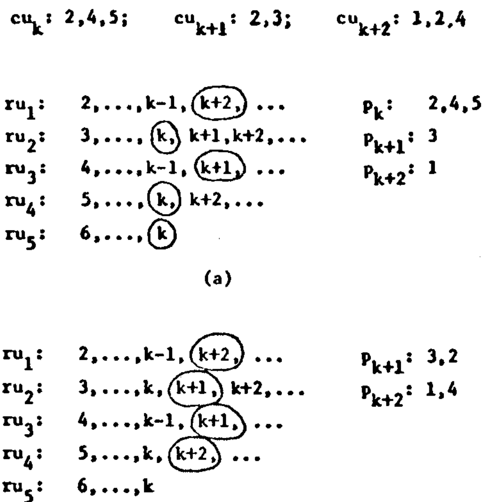


FIGURE 3.2

initially empty. Since cu_1 is also empty, $p_1 = cu_1$ during the first step of Algorithm 3.1. At the end of the first step, row 1 is added to p_j, where j is the minimum column index in ru_1 (see Figure 3.1b). Now if $cu_2 = 1$, then $2 \in ru_1$ and we will have $p_2 = 1$. Otherwise both cu_2 and p_2 will be empty. In either case, however, $p_2 = cu_2$, and the second step of the algorithm can be performed.

The more general situation during the k-th step of Algorithm 3.1 is illustrated in Figure 3.2. At the beginning of the step (Figure 3.2a), $p_k = cu_k$, and for $j > k$, p_j contains those row indices $i < k$ such that $i \in cu_j$ and j is the minimum column index in ru_i^{k-1}. Hence $p_k = 2,4,5$, $p_{k+1} = 3$, and $p_{k+2} = 1$. In particular, p_{k+1} includes all the indices in cu_{k+1}, except for those which are also in cu_k. At the end of the k-th step, the sets $\{p_j: j > k\}$ must be updated for the $k+1$-st step. This is done by moving each entry of p_k from p_k into the proper set p_j. For each $i \in p_k$, row i is appended to p_j, where j is the minimum column index in ru_i^k. If $ru_i^k = \emptyset$, then row i is not added to any set (e.g. row 5 in Figure 3.2b). Finally, row k is appended to the set p_j, where j is the smallest column index in ru_k. Figure 3.2b shows the sets $\{p_k\}$ which remain after the k-th step is completed. Now $p_{k+1} = cu_{k+1}$, so that the $k+1$-st step can be performed.

To determine the cost of Algorithm 3.1, we determine the times required for the arithmetic operations and the set-updating operations separately. There are $O(\theta_A)$ arithmetic operations, and

each one requires constant time; so $O(\theta_A)$ total time is required for the arithmetic operations. To determine the cost of the set updating operations, we note that at the k-th step, one updating operation is performed for each entry of $cu_k \cup \{k\}$, so that overall, $O(\theta_S)$ updating operations are performed. Making use of the fact that the sets $\{ru_k\}$ are ordered, the minimum operation in line 18 can be performed in constant time, since the first entry of ru_1^k will always be its smallest one. Thus the set-updating operations cost a total of only $O(\theta_S)$ time, and the entire algorithm runs in $O(\theta_A)$ time, since $\theta_S \le \theta_A$.

V.4 Storage of Sparse Matrices

In this section we describe the data structures used to efficiently store the matrices A and U (or equivalently the array M).[†] The storage schemes which we present are designed for use with sparse Gaussian elimination, and they take advantage of the row-by-row nature of the algorithms of Sections 2 and 3.

In general, our algorithms operate on entire rows of A and M at once. Hence it is important to be able to easily access rows of the matrix rather than individual entries. For this reason, the storage schemes described here are row-oriented, i.e., all of the nonzero entries of a matrix row are stored consecutively. In order to locate and identify the entries of a row, we

† The diagonal matrix D is stored in a one-dimensional array D so that $D(k) = d_{kk}$.

need to know where the row starts in storage, how long it is, and in what column each entry of the row lies. The extra storage for this information is the overhead storage mentioned in Section 1, and it is important to reduce it as much as possible.

Since A is symmetric, we need only store its upper triangle. We do this with the row-by-row storage scheme used in previous implementations of sparse symmetric Gaussian elimination (cf. Gustavson [G7] and the references therein). As shown in Figure 4.1, the scheme requires three one-dimensional arrays: IA, JA, and A. The nonzero entries of the upper triangle of A are stored row-by-row in the array A. The array JA contains the column indices which correspond to the entries of A, i.e., if $A(k) = a_{IJ}$, then $JA(k) = J$. If a_{IJ} is the first stored nonzero entry of the I-th row of the upper triangle of A, and if $A(k) = a_{IJ}$, then $IA(I) = k$. $IA(N+1)$ is defined so that $IA(I+1) - IA(I)$ is the number of nonzero entries in the I-th row of the upper triangle of A for $1 \le I \le N$. Thus the nonzero entries of the I-th row of the upper triangle of A are stored in A(IA(I)) through A(IA(I+1)-1), and the column indices corresponding to the I-th row of the upper triangle of A are stored in JA(IA(I)) through JA(IA(I+1)-1). In terms of our previous notation, the set of column indices stored for the I-th row is ra_I, and the input to the symbolic factorization algorithm is the arrays IA and JA.

The overhead in the row-by-row storage scheme for A is the storage required for IA and JA. Since IA has N+1 entries and JA has one entry for each nonzero stored in A, the

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & 0 \\ 0 & a_{32} & a_{33} & 0 & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & 0 & a_{46} \\ 0 & a_{52} & 0 & 0 & a_{55} & 0 \\ 0 & 0 & 0 & a_{64} & 0 & a_{66} \end{bmatrix}$$

	row 1		row 2				row 3	row 4		row 5	row 6
A:	a_{11}	a_{12}	a_{22}	a_{23}	a_{24}	a_{25}	a_{33}	a_{44}	a_{46}	a_{55}	a_{66}
k	1	2	3	4	5	6	7	8	9	10	11
JA	1	2	2	3	4	5	3	4	6	5	6
IA	1	3	7	8	10	11	12				

FIGURE 4.1

total storage is approximately twice the number of nonzeroes in the upper triangle of A.

The matrix U can also be stored with the row-by-row storage scheme just described (cf. Gustavson [G7]), since only its upper triangle needs to be stored. This storage scheme requires the arrays IU, JU, and U corresponding to IA, JA, and A (see Figure 4.2a). However, using the row-by-row storage scheme ignores certain features of U which can be exploited to reduce the overhead. Most obvious is the fact that all the diagonal entries of U are known to be 1, so that they need not be stored. Taking advantage of this leads to the storage scheme shown in Figure 4.2b in which the strict upper triangle of U is stored in row-by-row form.

An examination of Figure 4.2b now reveals that the column indices for certain rows of U in JU are actually final subsequences of the column indices for previous rows. In our previous notation, this is the situation in which the ordered set ru_k is a final subsequence of the ordered set ru_i, for some $i < k$, and in terms of the element model, it corresponds to the use of static condensation, i.e., the elimination of vertices interior to an element. In Figure 4.2b, for example, the column indices for row 3 are 4,5, while those for row 2 are 3,4,5. There is really no need to store the column indices for row 3 separately from those of row 2, provided that we know where to find them as a subsequence of the column indices for row 2.

Using this observation, we can compact the column indices in

$$U = \begin{bmatrix} 1 & u_{12} & 0 & 0 & 0 & 0 \\ 0 & 1 & u_{23} & u_{24} & u_{25} & 0 \\ 0 & 0 & 1 & u_{34} & u_{35} & 0 \\ 0 & 0 & 0 & 1 & u_{45} & u_{46} \\ 0 & 0 & 0 & 0 & 1 & u_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a)

U:	1	u_{12}	1	u_{23}	u_{24}	u_{25}	1	u_{34}	u_{35}	1	u_{45}	u_{46}	1	u_{56}	1
k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
JU:	1	2	2	3	4	5	3	4	5	4	5	6	5	6	6
IU:	1	3	7	10	13	15	16								

(b)

U:	u_{12}	u_{23}	u_{24}	u_{25}	u_{34}	u_{35}	u_{45}	u_{46}	u_{56}
k	1	2	3	4	5	6	7	8	9
JU:	2	3	4	5	4	5	5	6	6
IU:	1	2	5	7	9	10	10		

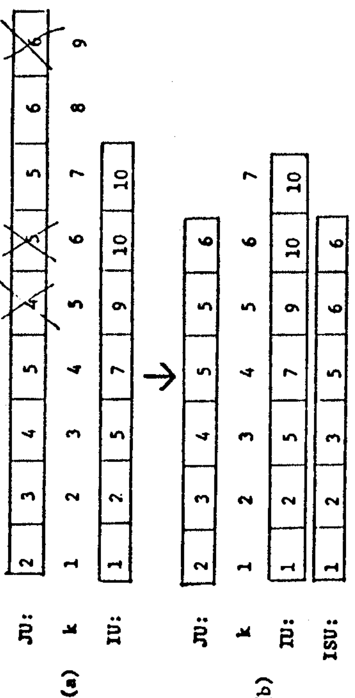
(c)

U:	u_{12}	u_{23}	u_{24}	u_{25}	u_{34}	u_{35}	u_{45}	u_{46}	u_{56}
k	1	2	3	4	5	6	7	8	9
JU:	2	3	4	5	5	6			
IU:	1	2	5	7	9	10	10		
ISU:	1	2	3	5	6	6			

FIGURE 4.2

JU by deleting the column indices for row k if they appear as a final subsequence of the column indices for row i, i < k (see Figure 4.3). (In practice this is done in the symbolic factorization algorithm by comparing $|ru_k|$ with $\max\{|ru_i^k|: i \in \ell_k\}$ and compacting the column indices for the k-th row of U if they are equal.) In order to find the column indices for any row in the compacted JU array, we must know where to look and how many column indices there are. We use the array ISU to locate the first column index for each row, and we determine the number of indices by using IU to compute the number of nonzeroes in the row (since there is one index per nonzero). Thus the compacted row-by-row storage scheme for U requires the four arrays IU, JU, ISU, and U, as shown in Figure 4.2c. U contains the nonzero entries of the strict upper triangle of U stored row-by-row, and the N+1 entries of IU delimit the rows in U as before. JU is the compacted array of column indices, and the entries of ISU locate the first column index in JU for each row. The nonzeroes of the I-th row of the strict upper triangle of U are stored in U(IU(I)) through U(IU(I+1)-1), and the corresponding column indices are stored in JU(ISU(I)) through JU(ISU(I) + IU(I+1) - IU(I) - 1). In our earlier notation, the set of indices stored for the I-th row in JU is the ordered set ru_I, and the output of the symbolic factorization algorithm is the arrays IU, JU, and ISU.

The overhead for the compacted row-by-row storage scheme is the storage required for IU, JU, and ISU. Clearly IU and ISU together require 2N+1 storage locations, so the bulk of the



Row	Locations of Column Indices: Before Compaction (a)	After Compaction (b)
1	JU(1)	JU(1)
2	JU(2) - JU(4)	JU(2) - JU(4)
3	JU(5) - JU(6)	JU(3) - JU(4)
4	JU(7) - JU(8)	JU(5) - JU(6)
5	JU(9)	JU(6)
6	-	-

FIGURE 4.3

overhead is the storage for JU (i.e. $|JU|$). It is not usually possible to determine a priori the amount of overhead for the compacted row-by-row storage scheme, but it is often far less than that of the ordinary row-by-row storage scheme (cf. Chapter VI, Eisenstat, Schultz, and Sherman [E3], Woo, Eisenstat, Schultz, and Sherman [W2]). In fact, we have the following theorem which shows that only $O(n^2)$ locations of overhead storage are required to solve the nine-point model problem. (Rose and Whitten [R9] and George [G4] have independently obtained similar results.) Although we will not give the proof here, it follows from a recursive analysis of the number of column indices required for the compacted row-by-row storage of the rows of U corresponding to each separating cross in the nested dissection ordering.

Theorem 4.1: Let D_n be an $n \times n$ mesh ordered with a nine-point nested dissection algorithm. Let A be a symmetric, positive definite coefficient matrix corresponding to the use of a nine-point finite difference operator on the ordered mesh D_n, and let U be such that $A = U^T DU$. If IU, JU, and ISU are the pointer arrays for the compacted row-by-row storage scheme for U, then

$$|U| \sim \frac{31}{4} n^2 \log n - \frac{73}{3} n^2 + O(n \log n);$$

and

$$|JU| \le 12n^2.$$

CHAPTER VI: PRACTICAL RESULTS

VI.1 Introduction

In this chapter we present the results of several numerical experiments involving the solution of the Poisson and biharmonic problems in the unit square. These experiments illustrate the use of sparse symmetric Gaussian elimination in practice and compare it with the band and envelope (or profile) algorithms for symmetric Gaussian elimination.

We begin in Section 2 by giving brief descriptions of band elimination (cf. Martin and Wilkinson [M1], Hindmarsh [H2]) and envelope elimination (cf. Jennings [J1], George [G1], Eisenstat and Sherman [E8]). We describe the matrix structures which are exploited to reduce the storage and arithmetic costs from those of dense elimination, and we discuss good orderings of the model problems for use with the two algorithms.

In Section 3 we present results for the five- and nine-point model problems. We give θ_S and θ_M for each of the three elimination algorithms and compare the actual running times and storage requirements (including overhead) for the corresponding

programs written in FORTRAN-IV. We find that sparse Gaussian elimination is faster and requires less storage than either of the other two algorithms, provided that the mesh points are ordered with an appropriate ordering (i.e. minimum degree or nested dissection).

In Section 4 we investigate the solution of the biharmonic equation in the unit square. Again we compare band, envelope, and sparse symmetric Gaussian elimination algorithms, but here we find that for moderate size problems, it may be better to use envelope elimination than sparse elimination. The reason for this conclusion is that we have been unable to find an efficient ordering for the points of the mesh used to approximate the biharmonic operator.

Finally, in Section 5 we consider techniques for the solution of sparse symmetric systems of linear equations which might arise in practice. We do not have any hard-and-fast rules, but by combining the known theory with practical experience, we are able to give some reasonable guidelines as to the most appropriate methods for certain classes of problems.

VI.2 Band and Envelope Methods

In this section we outline the ideas underlying the band and envelope (or profile) algorithms for symmetric Gaussian elimination, concentrating mainly on the application of these methods to the model problems. For more details see Martin and Wilkinson [M1] or Hindmarsh [H2] on band methods and Jennings [J1], George

[G1], or Eisenstat and Sherman [E8] on envelope methods.

A symmetric or upper triangular matrix M is said to be a band matrix with bandwidth m if all its nonzero entries lie in a band about the diagonal, i.e.,

$$m_{ij} \neq 0 \text{ implies } |i - j| \leq m.$$

The band of M is defined by

$$B(M) = \{(i,j): i \leq j \leq i+m\}.$$

When a band matrix A is factored into the product U^TDU, U is also a band matrix with the same bandwidth and band as A (see Figure 2.1). The band algorithm for symmetric Gaussian elimination assumes that only the entries of A and U which lie in $B(A)$ or $B(U)$ are nonzero and need be stored or operated on; other entries of A and U are ignored.

Envelope methods are a generalization of band methods which exploit zeroes at the edge of the band. For a symmetric or upper triangular matrix M, if we let f_j^M denote the row index of the first nonzero in the j-th column of M, then

$$m_{ij} \neq 0 \text{ and } i \leq j \text{ imply } f_j^M \leq i.$$

The envelope of M is defined by

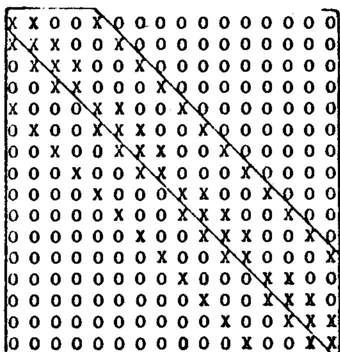
$$\text{Env}(M) = \{(i,j) : f_j^M \leq i \leq j\},$$

and when a symmetric matrix A is factored into the product U^TDU, we have $\text{Env}(U) = \text{Env}(A)$ (see Figure 2.2; cf. Jennings

	1	2	3	4
	5	6	7	8
	9	10	11	12
	13	14	15	16

Natural Ordering of D_4

(a)

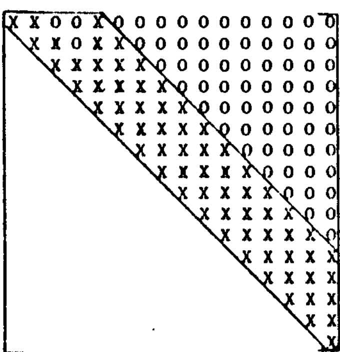


A

$m = 4$

$|B(A)| = 70$

(b)



U

67 nonzeroes

$m = 4$

$|B(U)| = 70$

(c)

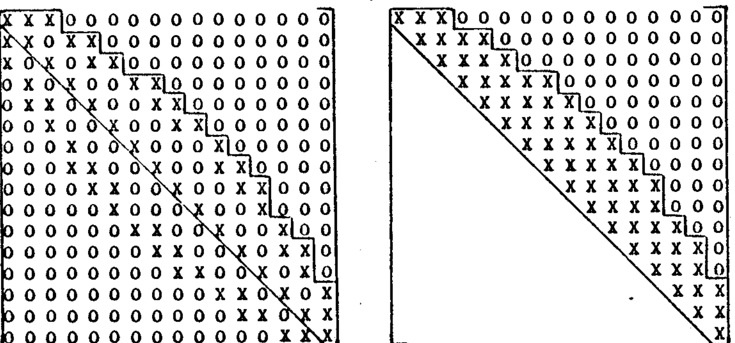
Zero Structures of A and U for the 4×4 Five-point Model Problem with the Natural Ordering

FIGURE 2.1

	1	2	4	7
	3	5	8	11
	6	9	12	14
	10	13	15	16

Diagonal Ordering of D_4

(a)



Zero Structures of A and U for the 4×4 Five-point Model Problem with the Diagonal Ordering

FIGURE 2.2

[J1] or George [G1]). The envelope algorithm for symmetric Gaussian elimination is similar to the band algorithm, except that only entries of A and U which lie within their respective envelopes are assumed to be nonzero and are stored or operated on.

In comparing the band and envelope methods, we note that $Env(A) \subseteq B(A)$, so that we would expect the envelope method to be more efficient, since it stores and operates on fewer matrix entries. As we will see in the next two sections, this intuitive comparison is substantiated in practice, at least for problems based on regular two-dimensional meshes.

Like the costs of the sparse algorithm, the costs of both the band and envelope algorithms for symmetric Gaussian elimination depend on the ordering of the equations and unknowns. (Cf. George [G1] for a discussion of how these costs may be computed.) Recently, there has been a great deal of research into good general ordering algorithms for use with these methods (cf. Cuthill [C6], Gibbs, Poole, and Stockmeyer [G5], Liu and Sherman [L1]). For the model problems, however, simple orderings of the mesh points are near-optimal for these methods (cf. Eisenstat [E1]), so we will not consider any general ordering techniques here.

With band methods, we use the natural ordering for both the five- and nine-point model problems. For the five-point model problem, the resulting coefficient matrix has bandwidth n, and the costs of the band algorithm for symmetric Gaussian elimination are $\theta_S = |B(A)| \approx n^3$ and $\theta_A \approx \theta_M \approx \frac{1}{2} n^4$. For the nine-

point model problem, the resulting coefficient matrix has band-width $n+1$, and the costs of the band algorithm for symmetric Gaussian elimination are again $\theta_S = |B(A)| \approx n^3$ and $\theta_A \approx \theta_M \approx \frac{1}{2} n^4$. In both cases these costs are optimal (cf. George [G3]).

With envelope methods, we use different orderings for the two model problems. For the five-point model problem we use the diagonal ordering of the mesh points (see Figure 2.2a), giving costs of $\theta_S = |Env(A)| \approx \frac{2}{3} n^3$ and $\theta_A \approx \theta_M \approx \frac{1}{4} n^4$. For the nine-point model problem, we use the natural ordering, giving costs of $\theta_S = |Env(A)| \approx n^3$ and $\theta_A \approx \theta_M \approx \frac{1}{2} n^4$. For both problems, Eisenstat [E1] has shown that these costs are near-optimal.

VI.3 Results for the Model Problems

In this section we present numerical results for the five- and nine-point model problems. These results are intended to compare the band, envelope, and sparse algorithms for symmetric Gaussian elimination and to determine the overhead involved with each of the three methods. They are similar to results reported previously by Eisenstat, Schultz, and Sherman [E4].

Our results are summarized in Tables 3.1 and 3.2. For each of the three methods, we give the following data:

θ_S The number of entries of U which must be stored;

θ_S^T Total storage (including all auxiliary vectors and pointers);

Problem/Method (Ordering)		θ_S	θ_S^T	σ_S	θ_M	t_M	σ_m	t_S
$h = \frac{1}{16}$	Band (Natural)	3480	3585	1.03	29240	.297	10.1	—
	Envelope (Diagonal)	2570	2795	1.09	17410	.220	12.6	—
	Sparse (Minimum Degree)	1577	3740	2.37	10931	.177	16.2	.067
	Sparse (Diagonal Nested Dissection)	1620	3766	2.32	11097	.177	15.9	.070
$h = \frac{1}{32}$	Band (Natural)	30256	30721	1.02	496496	5.263	10.6	—
	Envelope (Diagonal)	21266	22227	1.05	271746	2.893	11.2	—
	Sparse (Minimum Degree)	9919	19600	1.98	113198	1.587	14.0	.350
	Sparse (Diagonal Nested Dissection)	10144	19830	1.95	117558	1.633	13.9	.350

Results for Five-point Model Problem

TABLE 3.1

TABLE 3.2

Results for Nine-point Model Problem

Problem/Method (Ordering)		θ_S	θ_S^T	O_S	θ_M	t_M	O_M	t_S
$h = \frac{1}{16}$	Band (Natural)	3689	3809	1.03	32793	.360	11.0	—
	Envelope (Natural)	3585	3810	1.06	31389	.373	11.9	—
	Sparse (Minimum Degree)	2694	5021	1.86	24981	.357	14.3	.113
	Sparse (Nested Dissection)	2553	4873	1.91	21552	.317	14.7	.093
$h = \frac{1}{32}$	Band (Natural)	31185	31681	1.02	527153	5.530	10.5	—
	Envelope (Natural)	30721	31682	1.03	515901	5.313	10.3	—
	Sparse (Minimum Degree)	17603	28376	1.61	263544	3.380	12.8	.543
	Sparse (Nested Dissection)	16705	27449	1.64	228312	2.930	12.8	.517

O_S Measure of storage overhead (θ_S^T/θ_S);

θ_M Number of multiplications required to factor A;

t_M Time (in seconds) for numeric factorization of A;

O_M Measure of operational overhead (t_M/θ_M).

In addition, for the sparse algorithm, we give

t_S Time (in seconds) for symbolic factorization of A.

The times are in seconds and reflect execution times of FORTRAN-IV programs on an IBM 370/158 using the FORTRAN-IV Level H-Extended optimizing compiler. Despite our efforts to obtain accurate timings, the times may be subject to a variation of up to 10% due to system load.

From the tables we draw several conclusions for the model problems:

(i) The envelope method is usually preferable to the band method, since it requires no more storage or arithmetic operations and its overhead is of similar size. This superiority is particularly noticeable for the five-point model problem where the envelope method can exploit the structure of the coefficient matrix associated with the diagonal ordering;

(ii) Sparse symmetric Gaussian elimination is the fastest of the three methods, since it performs many fewer arithmetic operations and its operational overhead is only slightly larger;

(iii) The sparse method is usually the best of the three with respect to storage, although its storage overhead is the

largest;

(iv) The advantages of the compacted row-by-row storage scheme are substantial: even including all auxiliary vectors, fewer than two locations are required per nonzero in U. Without compaction, more than this is required to store U alone. (We note that other experiments (cf. Eisenstat [E2]) have shown that compaction increases the operational overhead by no more than ten percent.)

VI.4 Results for the Biharmonic Equation

In this section we give numerical results for the solution of linear systems arising from the biharmonic problem in the unit square.

Letting $D = (0,1)\times(0,1)$, we wish to solve the equation

$$\Delta^2 v = f \qquad \text{in } D \tag{4.1a}$$

$$v = 0, \quad \frac{\partial v}{\partial n} = 0 \qquad \text{on } \partial D, \tag{4.1b}$$

where ∂D is the boundary of D and $\frac{\partial v}{\partial n}$ denotes the normal derivative. To derive a discrete form of (4.1) we superimpose an $n\times n$ mesh D_n on D, replace the differential operator Δ^2 with the standard thirteen-point difference operator, and approximate the normal derivative along the boundary with central differences (cf. Buzbee and Dorr [B7]). We seek an approximation V_{ij} to $v(ih,jh)$ $(h = \frac{1}{n+1})$ for $(ih,jh) \in D_n$. Using the natural ordering of the mesh points, we obtain a linear system

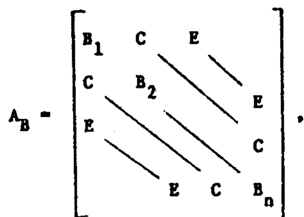
$$A_B \underset{\sim}{V} = \underset{\sim}{F}, \tag{4.2}$$

where

$$\underset{\sim}{V} = \{V_{ij}: (ih,jh) \in D_n\},$$

$$\underset{\sim}{F} = \{f(ih,jh): (ih,jh) \in D_n\},$$

and A_B is symmetric, positive definite, and irreducible. Moreover, A_B is an $n\times n$ block pentadiagonal matrix:



in which each of the blocks is $n\times n$. Each B_i is pentadiagonal, C is tridiagonal, and E is diagonal.

We compared the band, envelope, and sparse methods for the solution of (4.2), obtaining the same data as in Section 2. Our results, which are summarized in Table 4.1, appear to indicate that the envelope method is best for this problem. The reason for this is that we were unable to find an ordering of the mesh points for which the sparse algorithm for symmetric Gaussian elimination required many fewer multiplications than the envelope algorithm with the diagonal ordering. (The small difference in the number of multiplications was not large enough to overcome the difference in operational overhead.) The diagonal and minimum degree orderings did not give good results, and while we considered several asymptotically near-optimal divide-and-conquer orderings, they did not work well for the moderate size problems in our experiments.

Problem/Method (Ordering)		Θ_S	Θ_S^T	O_S	Θ_M	t_M	O_M	t_S
$h = \frac{1}{16}$	Band (Natural)	6510	6945	1.07	101680	1.020	10.0	—
	Envelope (Diagonal)	4872	5097	1.05	61094	.687	11.2	—
	Sparse (Minimum Degree)	4525	7295	1.61	64973	.840	12.9	.150
	Sparse (Diagonal)	4647	9985	2.15	61094	.813	13.3	.170
$h = \frac{1}{32}$	Band (Natural)	58590	60481	1.03	1854048	19.995	10.8	—
	Envelope (Diagonal)	41480	42441	1.02	1019718	10.42	10.2	—
	Sparse (Minimum Degree)	34346	47542	1.38	935085	11.393	12.2	.930
	Sparse (Diagonal)	40519	83953	2.07	1019718	—	—	1.10

Results for Biharmonic Model Problem

TABLE 4.1

VI.5 Guidelines for Sparse Symmetric Linear Systems

In this section we give some practical guidelines for solving sparse symmetric systems of linear equations. We base our recommendations on the theory discussed in this dissertation and on a variety of practical experience reported in the literature (cf. Price and Coats [P2], Woo, Roberts, and Gustavaon [W3], Seitelman [S1]).

To begin with, we note that for special problems such as our model problems, there are often special methods which work extremely well (cf. Dorr [D4, D5], Bank and Rose [B1]). However, since these methods may often not apply in general situations, we do not consider them further here.

Our guidelines are summarized in Table 5.1. We consider seven basic types of problems:

Type I: Systems arising from the use of finite difference or finite element methods in one dimension;

Type II: Systems arising from the use of five-, seven-, and nine-point difference operators (or similar Rayleigh-Ritz-Galerkin approximations) in regularly-shaped regions in two dimension;

Type III: Systems arising from the use of finite difference or finite element approximations in irregularly-shaped regions in two dimensions;

Type IV: Systems arising from the use of high-order finite difference or finite element approximations in two dimensions;

Type V: Systems arising from the use of finite difference

Problem Type	Method	
	Storage Most Important	Time Most Important
I	E	E
II	I, MS, OD	S, I
III	MS	S
IV	E	E
V	I	I
VI	N-I, NI	N-R, N-I, NI
VII	S	S

<u>Methods</u>

E Envelope Algorithm for Symmetric Gaussian Elimination

S Sparse Algorithm for Symmetric Gaussian Elimination

I Linear Iterative Methods

MS Minimal Storage Methods

OD One-Way Dissection

N-I Newton-Iterative Methods

NI Nonlinear Iterative Methods

N-R Newton-Richardson Methods

Recommended Methods for Sparse Symmetric Linear Systems

TABLE 5.1

or finite element approximations in three dimensions;

Type VI: Systems arising as part of a related sequence of systems (e.g. in the solution of nonlinear or time-dependent problems);

Type VII: Arbitrary sparse symmetric linear systems, such as might arise from the use of flow graphs or networks.

For each type of problem, we suggest a method or class of methods for use when either storage or time is the major consideration. Our recommendations are by no means hard-and-fast rules; they are merely pointers to methods which may offer reasonably efficient means of solution.

For Type I problems, the coefficient matrices can often be permuted to a dense envelope form (i.e. all of the entries of the envelope are actually nonzero), so that envelope methods are ideal. It is not possible to do better either storage-wise or time-wise with any other type of method.

For Type II problems (such as our model problems), a large number of special methods are available. Ignoring them, however, we recommend either iterative methods, especially Conjugate Gradient (CG) methods (cf. Chandra, Eisenstat, and Schultz [C1]), minimal storage methods (cf. Chapter VII and Eisenstat, Schultz, and Sherman [E5]), or perhaps one-way dissection (cf. George [G2], Sherman [S2]) when storage is the main consideration. Depending on the size and structure of a particular problem, either an iterative method (such as a variant of CG or SOR) or sparse symmetric Gaussian elimination should be used when time is the major con-

sideration.

For Type III problems, variants of Gaussian elimination seem particularly good because they are so robust, i.e., they will work quite well for a wide variety of problems over very irregular domains. To reduce storage, we recommend minimal storage methods, and to reduce time, we recommend sparse symmetric Gaussian elimination. As in Type II problems, however, iterative methods may be quite efficient in certain cases, and the best method to use is very problem-dependent.

For Type IV problems, envelope methods are workable, if not particularly efficient. Ordered properly, the coefficient matrices tend to have fairly dense envelopes, so that using a sparse storage scheme does not usually pay because of the increased overhead incurred. It is unclear how well iterative methods work for these problems, but the problems relating to starting guesses, iteration parameters, and stopping criteria would seem to be substantial. While not ideal, at least envelope methods will produce accurate results with a finite amount of effort.

For Type V problems, iterative methods appear to be the only feasible choice. As shown by Theorem IV.3.2, Gaussian elimination requires $\Omega(N^{4/3})$ storage locations and $\Omega(N^2)$ arithmetic operations for the Poisson problem in the unit cube, while a variant of the CG method (cf. Chandra, Eisenstat, and Schultz [C1]) requires only $O(N)$ storage locations and $O(N^{7/6}\log N)$ arithmetic operations. In general, iterative methods keep the required storage to a minimum, and they seem to be reasonably efficient time-wise (cf. Seitelman [S1]).

For Type VI systems, we recommend any method which exploits the relation among the systems in the sequence. For nonlinear problems, Newton-iterative or nonlinear iterative methods (cf. Ortega and Rheinboldt [O1]) require the least storage, while Newton-Richardson methods (cf. Chapter VIII and Eisenstat, Schultz, and Sherman [E4]), Newton-iterative methods, and nonlinear iterative methods should all be considered when time is the major consideration. For time-dependent problems, an iterative method such as CG or SOR should be used to reduce storage, while either an iterative method or a variant of the Newton-Richardson techniques might be best for reducing computation time.

Finally, for Type VII systems, we can recommend only sparse symmetric Gaussian elimination. That method can exploit the sparseness in the coefficient matrix and its upper triangular factor, and it will produce a solution in a moderate amount of time. The major difficulty is finding a good ordering for the equations and unknowns, but the minimum degree ordering (cf. Section IV.2) can be used for this purpose. For Type VII systems, iterative methods do not usually work well because the problems of starting guesses, iteration parameters and stopping criteria are particularly difficult.

CHAPTER VII: MINIMAL STORAGE METHODS

VII.1 Introduction

In preceding chapters we have examined the use of in-core sparse Gaussian elimination to solve large sparse symmetric systems of linear equations of the form

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

A practical problem with sparse Gaussian elimination for such systems is that the storage required for U may exceed the available core storage. When this occurs, some other method must be used to solve (1.1).

There are two common approaches when in-core Gaussian elimination is not feasible. One is to use iterative methods, since they only require enough storage for A, $\underset{\sim}{x}$, $\underset{\sim}{b}$, and some auxiliary vectors. However, as we pointed out earlier, there are several serious drawbacks to the general use of iterative methods, so we will not pursue them further.

An alternative is the use of special modifications of Gaussian elimination which take account of the fact that A cannot be

factored in core. The methods proposed in the past have used auxiliary storage on disk or tape to hold those portions of A and U which were not needed in core at any given time (cf. Jennings and Tuff [J2]). Auxiliary storage Gaussian elimination methods perform the same arithmetic operations as in-core methods, so our previous results on arithmetic cost still hold. In practice, though, the cost of these methods must also include the cost of the I/O (Input/Output) operations performed to move data between core and auxiliary storage. It is often difficult to implement this I/O efficiently, and, in any case, the resulting software is not very portable, in the sense that its performance is highly dependent on the particular combination of computer, operating system, and auxiliary storage device. If the amount of core storage is small and the available auxiliary storage devices are slow (e.g. as in certain mini-computer applications), the time required for the I/O operations may easily be nearly as large as that required for the arithmetic operations, even though fewer of them are performed. Because of these problems, we consider several machine-independent variants of Gaussian elimination (without any I/O) for which the core storage requirements are small enough that (1.1) may be solved even when U is too large to fit in core.

Throughout this chapter, we assume that there is enough core storage to store $\underset{\sim}{x}$, $\underset{\sim}{b}$, and the nonzero entries of A. The storage required to do this is a large part of the total storage required for the methods which we consider, but, for simplicity,

we do not include it in our estimates of storage costs, since it depends on the system (1.1) and not on the particular method used to solve it. Thus we measure storage requirements in terms of the storage needed to contain the parts of U necessary for any individual step of Gaussian elimination, and when we say that a method requires θ_S core storage locations, we mean θ_S storage locations in addition to any storage for A, $\underset{\sim}{x}$, and $\underset{\sim}{b}$.

In this chapter we introduce a class of algorithms which we call minimal storage Gaussian elimination methods (cf. Eisenstat, Schultz, and Sherman [E5]). Whereas the auxiliary storage methods trade a reduction in core storage for a certain amount of I/O, the minimal storage methods trade a reduction in core storage for an increase in the number of arithmetic operations. We will assume that there is enough core storage to perform any single step of Gaussian elimination, and we shall develop a variety of minimal storage methods which solve (1.1) without any I/O. The basic idea is to use the available core storage to compute some subset of the components of the solution $\underset{\sim}{x}$ and to use the computed components to reduce the size of the original system. The entire solution $\underset{\sim}{x}$ is obtained by repeating this partial solution step several times.

It is particularly natural and easy to implement minimal storage methods for banded linear systems (cf. Section VI.2) so we will use such systems to develop most of the ideas in this chapter. We discuss auxiliary storage band methods in Section 2, and in Section 3 we present two minimal storage band methods for general banded linear systems. In Section 4 we develop a minimal

storage band method specifically for the five- and nine-point model problems. Finally, in Section 5 we apply the minimal storage ideas to sparse symmetric Gaussian elimination to obtain minimal storage sparse methods for the model problems.

VII.2 Auxiliary Storage Band Methods

In this section we review auxiliary storage methods for solving systems of linear equations (1.1) in which the coefficient matrix A is banded.

If A is an $N \times N$ symmetric, positive definite matrix with bandwidth m, a common method for solving (1.1) is in-core band Gaussian elimination (cf. Martin and Wilkinson [M1]). One version of this method applies Gaussian elimination to the augmented matrix $(A,\underset{\sim}{b})$ (i.e. the matrix A with $\underset{\sim}{b}$ added as an extra column), storing and operating on just the entries of $B(A)$ and $B(U)$. This is equivalent to factoring A into the product U^TDU and solving $U^T\underset{\sim}{y} = \underset{\sim}{b}$ and $D\,\underset{\sim}{z} = \underset{\sim}{y}$. The resulting upper triangular system $U\,\underset{\sim}{x} = \underset{\sim}{z}$ must then be solved to obtain $\underset{\sim}{x}$. Since each row of the upper triangle of A or U can have only $m+1$ nonzero entries, it is straightforward to show (cf. Dahlquist and Björck [D1]) that the costs of band Gaussian elimination are

$$\theta_S = Nm + O(N+m^2)$$

and

$$\theta_A \approx \theta_M = \frac{1}{2}Nm^2 + \frac{3}{2}Nm - \frac{1}{3}m^3 + O(N+m^2). \quad (2.1)$$

Usually $m \ll N$, so that these costs are much less than those for

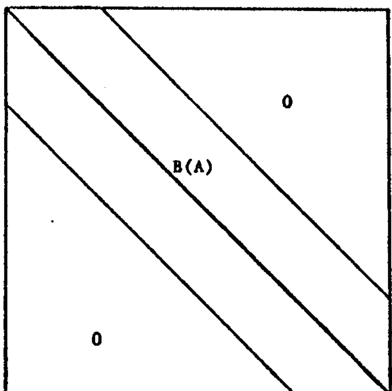
dense Gaussian elimination. Unfortunately, however, it is often the case that A is such that its nonzero entries fit into the core storage available, but that the $\sim Nm$ entries of $B(A)$ (or equivalently, $B(U)$) do not. Thus it is not possible to perform in-core band Gaussian Elimination, and it is necessary to modify the method to reduce the amount of storage required.

The problem of reducing the core storage for band methods has been studied before, and there are a number of algorithms which employ auxiliary storage to attack it (cf. Jennings and Tuff [J2],† Hindmarsh [H2]). These algorithms exploit the local nature of band methods, i.e., portions of at most $m+1$ consecutive columns or rows of $B(U)$ are required to perform any single step of the elimination or the backsolution. The auxiliary storage is used to hold the portions of $B(U)$ which are not required in core at any given time. In essence, core storage serves as a triangular window on $m+1$ columns of $B(U)$, which is fully stored in auxiliary storage (see Figure 2.1). After each step of the elimination, the next column from $B(A)$ is added to the righthand side of the window, and the entries of the oldest row at the top of the window are stored as a row of $B(U)$ in auxiliary storage.†† During the backsolution, the columns of $B(U)$ must be retrieved in reverse order.

Auxiliary storage band methods perform the same sequence of arithmetic operations as do ordinary band methods, so their arith-

† Jennings and Tuff [J2] actually develop an auxiliary storage envelope method, but it is straightforward to apply their ideas to band methods as well.

†† In practice, blocks of several columns and rows may be transferred at once in order to make the I/O more efficient.



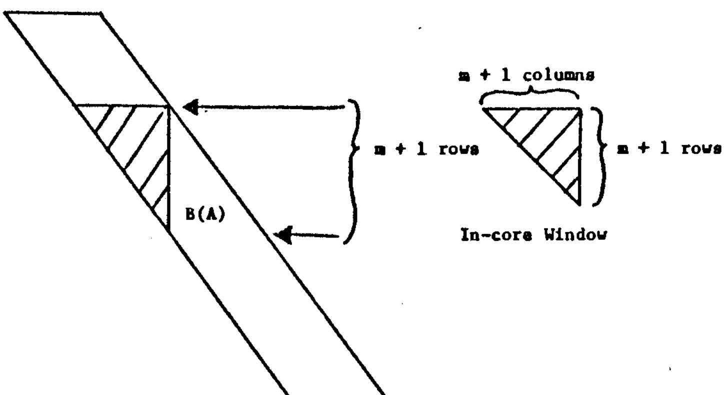


FIGURE 2.1

metic costs are given by (2.1). Since U is never stored entirely in core, the storage cost of the auxiliary storage band methods is exactly the amount of core storage required for the window on $B(U)$, i.e., $\theta_S \approx \frac{1}{2}(m+1)^2$. However, in addition to core storage and arithmetic operations, these methods also require $O(Nm)$ auxiliary storage locations and I/O operations to achieve the savings in core storage. As we pointed out in Section 1, the I/O may be difficult to implement efficiently, and, in any case, we desire machine independent algorithms which require no I/O operations. We develop two such algorithms in the next section.

VII.3 Minimal Storage Band Methods

In this section we introduce two minimal storage band methods which require approximately the same amount of core storage as the auxiliary storage band methods, but which use no auxiliary storage or I/O whatsoever (cf. Eisenstat, Schultz, and Sherman [E5]). Throughout, A is assumed to be an $N \times N$ symmetric, positive definite matrix having bandwidth m; for convenience in the presentation, we assume that $N = (m+1)^t$ for some positive integer t.

In the auxiliary storage band methods, portions of $m+1$ consecutive rows of $B(U)$ are kept in core at once, and at the conclusion of the elimination phase, the core storage window contains all the entries of the last $m+1$ rows of $B(U)$. Therefore, without retrieving any data from auxiliary storage, it is possible to

partially solve (1.1) by performing the first $m+1$ steps of backsolution to obtain the last $m+1$ components of $\underset{\sim}{x}$. We call this the ordinary partial solution algorithm.

The arithmetic operations performed in the ordinary partial solution algorithm are quite similar to those performed in an auxiliary storage band method, since band Gaussian elimination is applied to $(A,\underset{\sim}{b})$ in both cases. The storage and arithmetic costs for obtaining the last $m+1$ components are

$$\theta_S \approx \frac{1}{2}(m+1)^2$$

and

$$\theta_A \approx \theta_M = \frac{1}{2}Nm^2 + \frac{3}{2}Nm - \frac{1}{3}m^3 + O(N+m^2).$$

However, it is not necessary to re-use the first $N-(m+1)$ rows of $B(U)$ in computing the last $m+1$ components of $\underset{\sim}{x}$. By discarding the entries of these rows of $B(U)$ as soon as they are no longer part of the in-core window, we can avoid the use of auxiliary storage or I/O.

The MSB1 method obtains the full solution of (1.1) by applying the ordinary partial solution algorithm $N/(m+1)$ times. Each step computes $m+1$ components of $\underset{\sim}{x}$, thus reducing the original system by eliminating the corresponding $m+1$ rows and columns of A. At the $k+1$-st step of MSB1, the reduced system which remains to be partially solved is the $(N-k(m+1)) \times (N-k(m+1))$ system

$$A_{k+1}\tilde{\underset{\sim}{x}} = \tilde{\underset{\sim}{b}}$$

which is obtained from the first $N-k(m+1)$ equations of (1.1) by

substituting in the known components of $\underset{\sim}{x}$ as constants. In general, the bandwidth of A_{k+1} is the same as that of A, so that the total costs of MSB1 are given by the following theorem.

<u>Theorem 3.1</u>: Let A be an N×N symmetric matrix with bandwidth m. Then the costs of solving (1.1) with MSB1 are

$$\theta_S \approx \frac{1}{2}(m+1)^2$$

and

$$\theta_A \approx \theta_M = \frac{1}{4}N^2m + O(N^2+Nm^2).$$

Since it only reduces the size of the original problem by a small constant amount at each step, MSB1 increases the cost of a band solution of (1.1) by a factor of $\sim\frac{1}{2}N/m$. As in the solutions of many other problems, it is better to use a divide-and-conquer strategy (cf. Aho, Hopcroft, and Ullman [A1], pp. 60-66), i.e., to reduce the problem to two smaller problems of roughly half the size at each step. To do this, however, requires partially solving (1.1) not for the last m+1 components of $\underset{\sim}{x}$, but for some m+1 consecutive components of $\underset{\sim}{x}$ (say $x_k, x_{k+1}, \ldots, x_{k+m}$) which split the problem in half. To obtain such a divide-and-conquer partial solution, we make use of the observation that the Gaussian elimination process on $(A,\underset{\sim}{b})$ could just as well have been performed by eliminating the variables in reverse order, starting with x_N and working backwards to x_1.

We start with an initial in-core window containing the last m+1 rows of B(A) (see Figure 3.1a) and proceed as before, except that we eliminate the variables in reverse order and update the

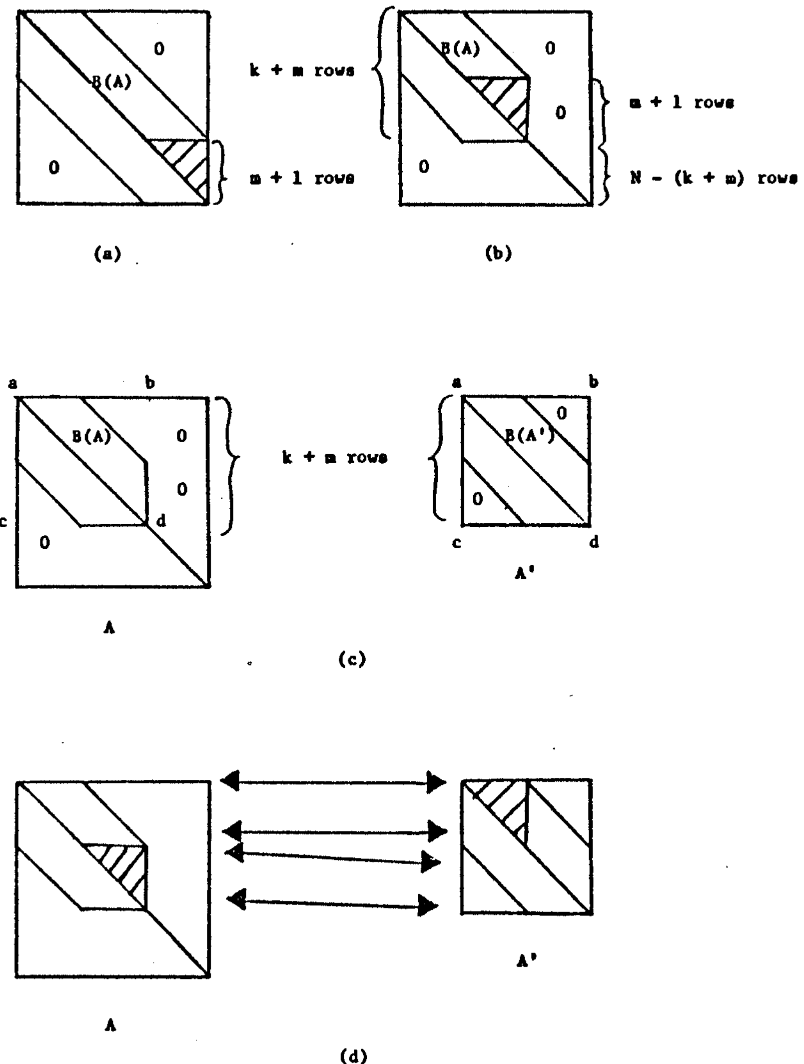


FIGURE 3.1

in-core window by adding rows to the top and discarding columns from the right. After $p = N-(k+m)$ steps of backwards elimination, the in-core window is as shown in Figure 3.1b, and the first $k+m$ equations of the system no longer depend on the variables $x_{k+m+1}, x_{k+m+2}, \ldots, x_N$. Thus the desired components of $\underset{\sim}{x}$ are the last $m+1$ components of the solution $\underset{\sim}{x}'$ to the system

$$A' \underset{\sim}{x}' = \underset{\sim}{b}', \tag{3.1}$$

where A' is the leading principal $(k+m)\times(k+m)$ minor of A and $\underset{\sim}{b}'$ is the updated righthand side after p steps of backwards elimination (see Figure 3.1c). We can obtain $x_k, x_{k+1}, \ldots, x_{k+m}$ by applying the ordinary partial solution algorithm to (3.1).

The arithmetic costs of the divide-and-conquer partial solution algorithm are the same as those of the ordinary one, since the two algorithms just use different variable orderings in applying band Gaussian elimination to $(A,\underset{\sim}{b})$. However, the storage costs are higher for the divide-and-conquer algorithm, since one in-core window is not sufficient to perform the computations. After the backwards elimination, the in-core window contains portions of rows k through $k+m$ of $B(U)$, but the window required for the initial steps of the ordinary partial solution of (3.1) contains portions of the first $m+1$ rows of $B(A)$ (see Figure 3.1d). Hence two separate in-core windows are needed, increasing the storage cost to $\theta_S \approx (m+1)^2$.

We obtain the MSB2 method by recursively applying the divide-and-conquer partial solution algorithm. The first step of MSB2 partially solves (1.1) for $x_k, x_{k+1}, \ldots, x_{k+m}$, where

$$k-1 \approx N-(k+m).$$

(See Figure 3.2a). These $m+1$ components of $\underset{\sim}{x}$ are used to reduce the original problem to two independent subproblems of order $\sim \frac{N}{2}$ and bandwidth m (see Figure 3.2b). At each succeeding step, the divide-and-conquer partial solution algorithm is applied to each of the remaining independent subproblems, thus further reducing the original problem. After k steps, the solution has been obtained for $(2^k-1)(m+1)$ components of $\underset{\sim}{x}$, and there are 2^k subproblems of order $\sim N/2^k$ and bandwidth m remaining to be solved. Hence the k+1-st step of MSB2 will require

$$\sim 2^k \left(\frac{1}{2} (N/2^k) m^2 \right) \approx \frac{1}{2} Nm^2$$

multiplications and additions, and after $\sim\log(N/(m+1))$ steps, (1.1) will be completely solved. This leads to the following result.

<u>Theorem 3.2</u>: Let A be an $N \times N$ symmetric matrix with bandwidth m. Then the costs of solving (1.1) with MSB2 are

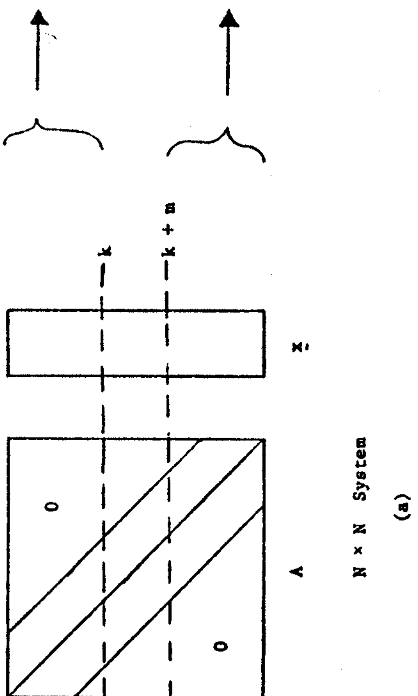
$$\theta_S \approx (m+1)^2$$

and

$$\theta_A \approx \theta_M = \frac{1}{2} Nm^2 \log(N/(m+1)) + O(Nm^2).$$

VII.4 <u>Minimal Storage Band Methods for the Model Problems</u>

It appears that both MSB1 and MSB2 must increase the cost of solving a banded system since the matrices in the subproblems gen-



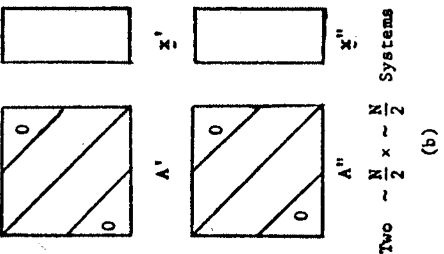


FIGURE 3.2

erated by the two methods all have the same bandwidth as the original matrix A. It is certainly possible to reorder the generated matrices to try to reduce their bandwidths, but this will not have any substantial effect for general banded systems. However, for the five- and nine-point model problems, it is possible to apply MSB2 with reordering in such a way that the bandwidths of the generated matrices are much less than the bandwidth of the original system.

In this section we present the MSBA method, a special version of MSB2 which exploits the mesh structure of the model problems to minimize the bandwidths of the generated matrices. MSBA has the same storage requirement as MSB2, but it applies the divide-and-conquer partial solution algorithm in a particularly careful way so that it solves the model problems at a cost less than twice that of in-core band Gaussian elimination. In order to simplify the discussion here, we will actually treat only the five-point model problem on an $n \times n$ mesh with $n = 2^t - 1$, but all the results also hold for the nine-point model problem and arbitrary values of n.

For $q \leq p$, the natural ordering of a $q \times p$ mesh is the ordering shown in Figure 4.1. (In a rectangular mesh we order the points in the shorter direction first.) In a five-point finite difference system derived from such a mesh, the coefficient matrix must always have bandwidth $m \geq q$ (cf. George [G3]), and the minimum possible bandwidth q can be achieved by ordering the system to correspond with the natural ordering of the mesh points. Thus

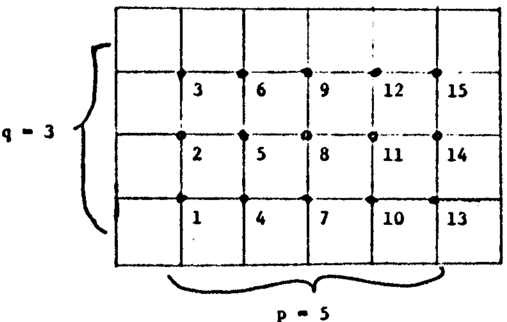


FIGURE 4.1

the system can be solved with in-core band Gaussian elimination at costs of

$$\theta_S = pq^2 + O(pq)$$

and

$$\theta_A \approx \theta_M = \frac{1}{2}pq^3 + \frac{3}{2}pq^2 - \frac{1}{3}q^3 + O(pq).$$

MSBA is similar to MSB2, except that care is taken to make sure that the generated subproblems are actually model problems on $q \times p$ meshes $(q \leq p \leq n = 2^t - 1)$, so that they can be reordered with the natural mesh ordering to achieve the minimum possible bandwidth. At the first step of MSBA, we partially solve the $n \times n$ model problem to obtain the n components of $\underset{\sim}{x}$ corresponding to the line of mesh points which divides the mesh into two $(\frac{n-1}{2}) \times n$ submeshes (see Figure 4.2a).† By incorporating the computed components of $\underset{\sim}{x}$ as boundary values, this produces two subproblems which correspond to five-point model problems on $(\frac{n-1}{2}) \times n$ submeshes. Each of these is reordered with the natural ordering (see Figure 4.2b) to produce a bandwidth of $\frac{n-1}{2}$ for the corresponding coefficient matrix. The second step of MSBA then partially solves each of the two generated subproblems to obtain the components of $\underset{\sim}{x}$ corresponding to lines of $\frac{n-1}{2}$ mesh points which divide the submeshes in half as shown in Figure 4.2c. After the first two steps of MSBA, the original problem has been reduced to four model subproblems which are structurally identical to the

---

† Here $m = n$, but we partially solve for only n components of $\underset{\sim}{x}$ in order to have them correspond to a mesh line.

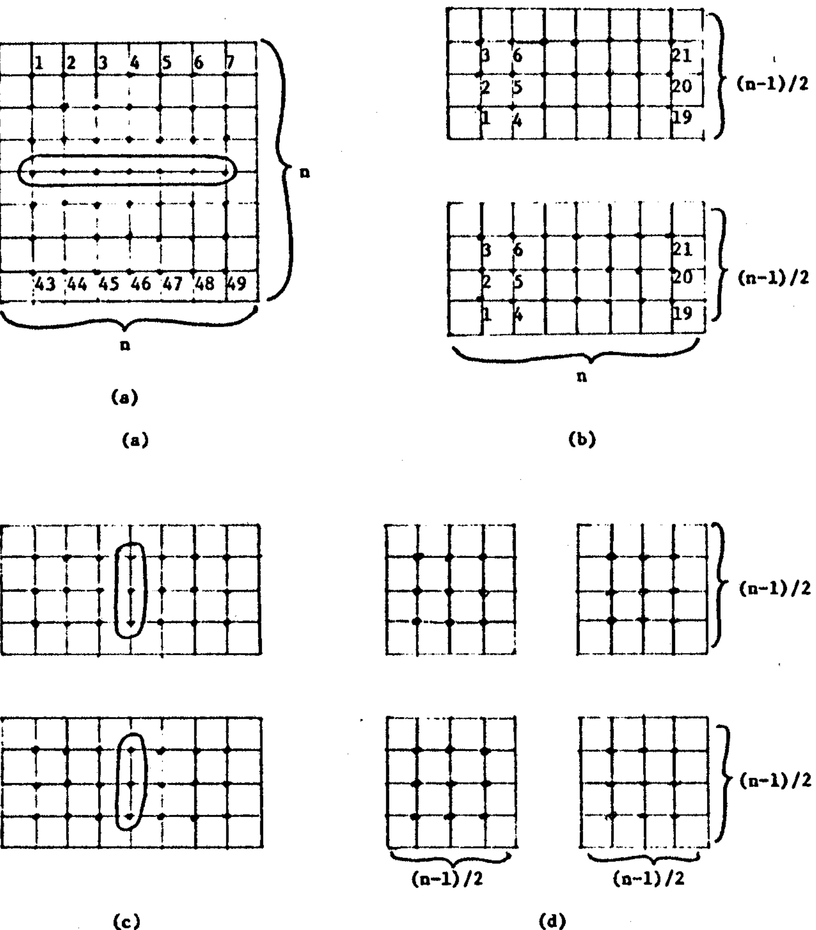


FIGURE 4.2

original problem, except that they are on $(\frac{n-1}{2}) \times (\frac{n-1}{2})$ meshes (see Figure 4.2d). Further pairs of steps apply this two-step reduction recursively to the generated model subproblems until the generated subproblems are "easily-solvable," i.e., the bands of their coefficient matrices can be entirely contained in core storage. These are then completely solved with in-core band Gaussian elimination. (In this case any subproblem on a $q \times p$ mesh, with $q \le p \le n^{2/3}$, is easily-solvable.) Since the $k+1$-st pair of steps partially solves for $4^k(2n/2^k - 1)$ components of $\underline{x}$, at most $\log n$ pairs of steps are required to solve the entire model problem.

The first step of MSBA requires $\sim n^2$ storage locations for the in-core windows, since the $n \times n$ model problem leads to a matrix of bandwidth n. Succeeding steps of the algorithm partially solve systems having bandwidths $m < n$, so that they require less storage than the first step. Hence the storage cost of MSBA is $\theta_S \approx n^2$. The arithmetic costs of MSBA are given by the following theorem.

<u>Theorem 4.1</u>: Let $n = 2^t - 1$. Then the arithmetic costs of using MSBA to solve a five-point model problem on an $n \times n$ mesh are

$$\theta_A \approx \theta_M = \frac{5}{6} n^4 + \frac{11}{3} n^3 + O(n^2 \log n).$$

<u>Proof</u>: The arithmetic costs of the first step of MSBA are

$${}_1\theta_A \approx {}_1\theta_M = \frac{1}{2} n^4 + \frac{7}{6} n^3 + O(n^2),$$

since the original problem is on an $n \times n$ mesh. After reordering, the second step of MSBA partially solves two subproblems on $(\frac{n-1}{2}) \times n$ meshes, so its arithmetic costs are

$$_2\theta_A \approx {_2\theta_M} = \frac{1}{8}n^4 + \frac{2}{3}n^3 + O(n^2).$$

Hence total arithmetic costs of

$$\theta_A^{(1)} \approx \theta_M^{(1)} = \frac{5}{8}n^4 + \frac{11}{6}n^3 + O(n^2)$$

are required to reduce the original problem on an $n \times n$ mesh to four five-point model subproblems on $(\frac{n-1}{2}) \times (\frac{n-1}{2})$ meshes.

Let $\theta(p)$ denote the number of multiplications required to solve a model problem on a $p \times p$ mesh with MSBA. Then $\theta_A \approx \theta_M = \theta(n)$, and we have

$$\theta(p) = \frac{5}{8}p^4 + \frac{11}{6}p^3 + 4\theta(\frac{p}{2}) + O(p^2) .$$

The difference equation can be solved to obtain

$$\theta(n) \le \sum_{k=0}^{\log n - 1} 4^k [\frac{5}{8}(n/2^k)^4 + \frac{11}{6}(n/2^k)^3 + O((n/2^k)^2)]$$

$$= \frac{5}{6}n^4 + \frac{11}{3}n^3 + O(n^2 \log n). \qquad \square$$

It is possible to save some arithmetic operations by completely solving the largest easily-solvable subproblems. (However, since almost all the work in MSBA occurs in the first several reductions, this will not have a large effect.) Nonetheless, it is surprising to see that even in the worst case, the arithmetic costs

of MSBA for the five-point model problem are less than twice as large as those of in-core band Gaussian elimination which requires far greater amounts of core storage. Similar results can be shown to hold for the nine-point model problem, since $q+1$ is the bandwidth of the coefficient matrix derived by using a nine-point finite difference operator on a $q \times p$ mesh $(q \le p)$ using the natural ordering.

VII.5 Minimal Storage Sparse Methods for the Model Problems

In this section we extend the ideas of Sections 3 and 4 to obtain minimal storage sparse methods. Although such methods can be applied to any sparse symmetric linear system, we will develop two specific methods (breadth-first MSSA and depth-first MSSA) for the nine-point model problem on an $n \times n$ mesh D_n. The MSSA methods illustrate the basic principles which would be involved in general minimal storage sparse methods, but we can present them much more simply and clearly. For convenience, we assume that $n = 2^t - 1$, but the results hold for arbitrary values of n.

The MSSA methods are quite similar in structure to the MSBA method, since their basic strategy is to reduce problems on $p \times p$ meshes to subproblems on $(\frac{p-1}{2}) \times (\frac{p-1}{2})$ meshes. However, where MSBA takes two partial solution steps to achieve this reduction, the MSSA methods require only one. At the first step of the MSSA methods, a sparse version of the ordinary band partial solution algorithm is used to compute the $2n-1$ components of $\underset{\sim}{x}$

corresponding to the nested dissection separating cross in the $n \times n$ mesh (see Figure 5.1a). This divides the original problem into four model subproblems on $(\frac{n-1}{2}) \times (\frac{n-1}{2})$ submeshes. The second step then applies the partial solution algorithm to each of the generated subproblems to obtain the solutions on the separating crosses in the submeshes (see Figure 5.1b). Further partial solutions are performed recursively in this manner until a group of easily-solvable subproblems is obtained, and then these are completely solved in core, one at a time, with in-core sparse Gaussian elimination.

The sparse partial solution algorithm for a $p \times p$ mesh $(p = 2^r-1)$ is easy to outline. We first order the mesh points with a nine-point nested dissection ordering and let

$$A\,\underset{\sim}{x} = \underset{\sim}{b} \qquad (5.1)$$

be the corresponding nine-point model linear system. We then perform sparse symmetric Gaussian elimination on $(A,\underset{\sim}{b})$, discarding those parts of A and U which are no longer required as we go along. (The storage reclaimed in this manner can subsequently be reused.) At the end of the elimination phase, the last $2p-1$ rows of U are retained in core, and the last $2p-1$ components of $\underset{\sim}{x}$ are obtained by partial backsolution. Since the last $2p-1$ variables in a nine-point nested dissection ordering are the variables corresponding to the separating cross in the $p \times p$ mesh, this yields exactly the components of $\underset{\sim}{x}$ which are required to reduce a problem on a $p \times p$ mesh to four subproblems on $(\frac{p-1}{2}) \times (\frac{p-1}{2})$

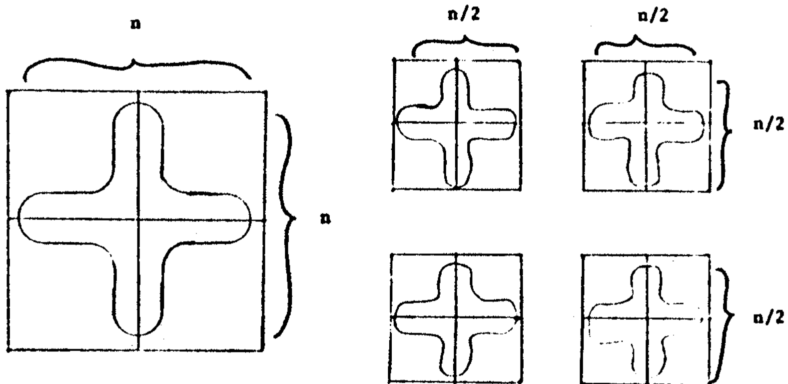
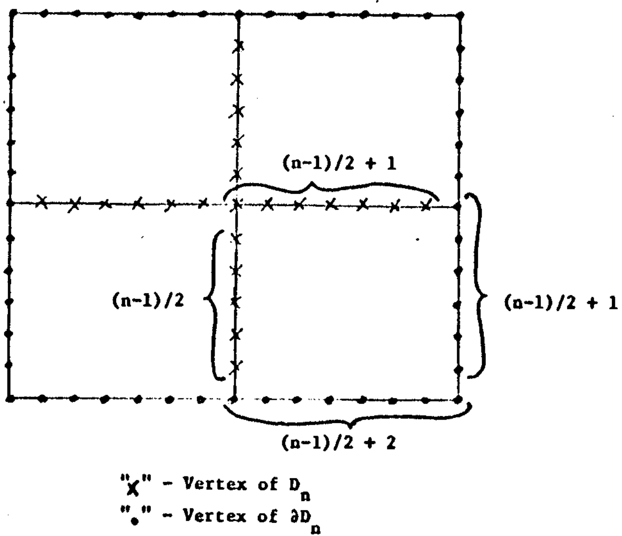


FIGURE 5.1



"X" - Vertex of D_n
"•" - Vertex of ∂D_n

Elements on Level 1

FIGURE 5.2

meshes. In terms of the amount of arithmetic performed, this sparse partial solution algorithm is essentially equivalent to performing sparse symmetric Gaussian elimination on A, so the arithmetic costs are given by Theorem IV.3.3.

In general, the k+1-st step of an MSSA method applies the sparse partial solution procedure to 4^k model subproblems on $(2^{t-k}-1) \times (2^{t-k}-1)$ meshes, obtaining $4^k(2^{t-k+1}-3)$ components of $\underset{\sim}{x}$. Thus no more than $\log n$ steps are required to completely solve the original problem on an $n \times n$ mesh, and we have the following theorem.

Theorem 5.1: Let $n = 2^t-1$. Then the arithmetic costs of using an MSSA method to solve a nine-point model problem on an $n \times n$ mesh are

$$\theta_A \approx \theta_M = \frac{267}{14} n^3 + O(n^2(\log n)^2).$$

Proof: The arithmetic costs of the first step of an MSSA method are

$$\theta_A^{(1)} \approx \theta_M^{(1)} = \frac{267}{28} n^3 - 17 n^2 \log n + O(n^2),$$

since it is equivalent to solving (5.1) with sparse symmetric Gaussian elimination. The original problem is then reduced to four identical model subproblems on $(\frac{n-1}{2}) \times (\frac{n-1}{2})$ meshes. Let $\theta(p)$ denote the number of multiplications required to solve a model problem on a $p \times p$ mesh with an MSSA method. Then $\theta_A \approx \theta_M = \theta(n)$, and we have

$$\theta(p) = \frac{267}{28} p^3 - 17 p^2 \log p + 4\theta(p/2) + O(p^2).$$

The difference equation may be solved to obtain

$$\theta(n) \le \sum_{k=0}^{\log n -1} 4^k \left[\frac{267}{28} (n/2^k)^3 - 17(n/2^k)^2 \log(n/2^k) + O((n/2^k)^2) \right]$$

$$= \sum_{k=0}^{\log n -1} 4^k \left[\frac{267}{28} (n/2^k)^3 \right] - \sum_{k=0}^{\log n -1} 4^k [17(n/2^k)^2 \log(n/2^k) + O((n/2^k)^2)]$$

$$= \frac{267}{14} n^3 + O(n^2(\log n)^2). \qquad \square$$

Up to now, we have ignored the main difficulty with the MSSA methods: determining just what parts of U must be in core for any given elimination step of the partial solution algorithm. We will use the element merging model of Section III.5 to solve this problem and to analyze the storage requirements of the MSSA methods.

Consider first the formation of a superelement by element merging in the compacted element merge tree corresponding to (5.1). All of the information required to perform the element merge is contained in the nonzeroes of A and U which correspond to the elements being merged; thus the element storage required for the merge is that needed for these nonzeroes and those nonzeroes which correspond to the superelement being formed.† Since each ele-

† The element storage does not include a fixed amount of extra

ment in the compacted element merge tree is merged with others at most once in the element merging process, the nonzeroes corresponding to an element may be discarded as soon as it has been merged.

We will call an element active if it has been or is being formed by merging, but has not yet been merged to form a larger superelement. Only the nonzeroea in U which correspond to active elements must be kept in core at any time, and the element storage requirement of the MSSA methods is simply the largest amount of storage ever needed to do this. (We assume that the nonzeroes corresponding to the original elements at the bottom of the tree may be obtained as needed, so that those elements are never considered active until they are actually required for an element merge.)

We can now bound the element storage required for the MSSA methods by examining the compacted element merge trees for the individual partial solutions. Because the generated subproblems are always nine-point model problems on $p \times p$ meshes $(p < n)$, the element storage required to partially solve any subproblem is less than that required for the partial solution at the first step. Hence we determine the total element storage required for MSSA methods by analyzing the storage cost of the first partial solu-

---

storage required to perform static condensation implicit in the compacted element merge. For the nine-point model problem, it can be shown that $\sim 2n^2$ storage locations is sufficient for this purpose, and we include that amount in Theorem 5.2.

tion. We consider MSSA methods using either breadth-first or depth-first nested dissection to order the meshes at each step.

In our analysis we ignore the single element on level 0 of the compacted element merge tree, since it does not correspond to any nonzeroes in U. Furthermore, we exploit the fact that each active element on level 1 of the tree contains $n+2$ vertices of ∂D_n which do not correspond to rows of A and U (see Figure 5.2), but we do not attempt to account for the inclusion of vertices of ∂D_n in elements on other levels of the tree. The storage required for the nonzeroes which correspond to an active element of size $|s|$ is $\frac{1}{2}|s|(|s|+1)$, since the nonzeroes form an $|s| \times |s|$ dense symmetric minor of a symmetric matrix. Thus letting $\theta_S^{(\ell)}$ be the storage required for the nonzeroes corresponding to an active element on level ℓ of the tree, we have

$$\theta_S^{(1)} = \frac{1}{2} n(n+1) \approx \frac{n^2}{2} \tag{5.2a}$$

and

$$\theta_S^{(\ell)} = \frac{1}{2} (4n/2^\ell)(4n/2^\ell+1) \approx 8n^2/4^\ell, \quad \text{for } 2 \le \ell \le \log n. \tag{5.2b}$$

Breadth-first MSSA is obtained by applying MSSA using breadth-first nested dissection to order the generated subproblems at each step. The compacted element merge tree for the first step is similar to that shown in Figure IV.3.4, and the merges in the tree are performed in a breadth-first or level-by-level order. When the elements on level ℓ $(\ell \ge 1)$ are being formed from the elements on level $\ell+1$, enough storage must be available to store the non-

zeroes which correspond to active elements on both levels. Elements below the two current levels in the tree have been merged and discarded; elements above the current levels have yet to be formed. Initially, all of the elements on level $\ell+1$ and the leftmost element on level ℓ are active. After forming the leftmost element on level ℓ, we discard storage associated with the four elements on level $\ell+1$ which were merged to form it, and we allocate storage for the nonzeroes corresponding to the next element on level ℓ. However, since $\theta_S^{(\ell)} < 4\theta_S^{(\ell+1)}$, the storage freed by discarding four elements on level $\ell+1$ is sufficient to store the nonzeroes associated with the new active element on level ℓ, so the total amount of element storage required does not increase. Thus the element storage for breadth-first MSSA method is

$$\theta_E^B = \max_{1\le\ell\le\log n} (\theta_S^{(\ell)} + 4^{\ell+1}\theta_S^{(\ell+1)})$$

$$= \theta_S^{(2)} + 4^3\theta_S^{(3)}$$

$$\approx \frac{17}{2} n^2 .$$

Depth-first MSSA is obtained by applying MSSA using depth-first nested dissection at each step. An examination of the compacted element merge tree at the first step of depth-first MSSA (see Figure IV.3.7) shows that if levels i through j $(1 \le i < j)$ of the tree contain active elements at any time, then there are at most three active elements on levels i through $j-2$

and at most four active elements on levels $j-1$ and j (see Figure 5.3). The reason for this is that merges are performed in a depth-first manner, as far to the left and top of the tree as possible. Thus making use of (5.2), we have that the element storage required for depth-first MSSA is given by

$$\theta_E^D = \max_{2\le\ell\le\log n} [\sum_{k=1}^{\ell-2} 3\theta_S^{(k)} + 4\theta_S^{(\ell-1)} + 4\theta_S^{(\ell)}]$$

$$= \max_{2\le\ell\le\log n} [3\theta_S^{(1)} + \sum_{k=2}^{\ell} 3\theta_S^{(k)} + \theta_S^{(\ell-1)} + \theta_S^{(\ell)}]$$

$$\approx \max_{2\le\ell\le\log n} [\frac{3}{2} n^2 + (2n^2 - 8n^2/4^\ell) + \theta_S^{(\ell-1)} + 8n^2/4^\ell]$$

$$\approx \frac{7}{2} n^2 + \theta_S^{(1)}$$

$$\approx 4n^2$$

Now taking into account the $\sim 2n^2$ extra storage locations required for static condensation, we have shown the following theorem.

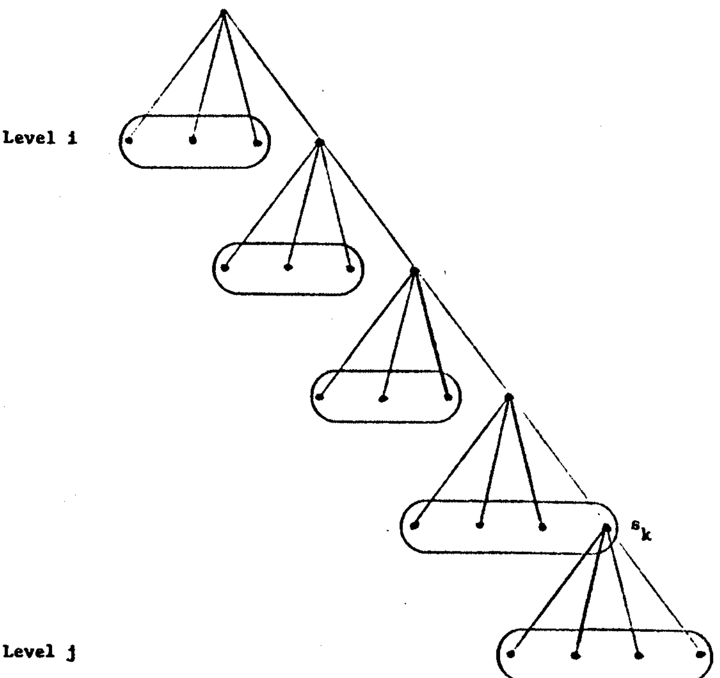
Theorem 5.2: Let $n = 2^t-1$. Then the storage costs of solving a nine-point model problem on an $n \times n$ mesh with the breadth-first and depth-first MSSA methods are

$$\theta_S^B \approx \frac{21}{2} n^2$$

and

$$\theta_S^D \approx 6n^2,$$

respectively.



Circled elements are active while forming s_k.

FIGURE 5.3

Method	Θ_S	$\Theta_M(\sim\Theta_A)$
MSB1	$\frac{1}{2}n^2$	$\frac{1}{4}n^5$
MSBA	n^2	$\frac{5}{6}n^4$
Depth-first MSSA	$6n^2$	$\frac{267}{14}n^3$
Breadth-first MSSA	$\frac{21}{2}n^2$	$\frac{267}{14}n^3$
In-core Sparse Gaussian Elimination	$\frac{31}{4}n^2 \log n$	$\frac{267}{28}n^3$

Comparison of In-core Gaussian Elimination Methods
for Nine-point Model Problem

TABLE 5.1

To summarize the results of this and the previous section, we refer to Table 5.1, which lists a hierarchy of in-core Gaussian elimination methods for the nine-point model problem. It demonstrates a consistent tradeoff between the storage cost and the arithmetic costs, and it indicates the fact that MSBA requires less storage than either version of MSSA. (This is somewhat surprising, since in-core sparse Gaussian elimination requires far less storage than in-core band Gaussian elimination for the model problem.) Hence when the storage requirement is the main consideration, as it might be in certain minicomputer applications, minimal storage band methods are more appropriate than minimal storage sparse methods.

CHAPTER VIII: SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS

VIII.1 Introduction

In this chapter we consider the application of our previous results to the solution of systems of nonlinear equations. In general, if R is a subset of $\mathbb{R}^N$ and $F: R \to \mathbb{R}^N$ is a nonlinear function, we are interested in computing the solution $\underset{\sim}{x}^*$ to the nonlinear system

$$F(\underset{\sim}{x}) = 0, \tag{1.1}$$

where the Jacobian matrix

$$F'(\underset{\sim}{x}) = \frac{\partial F}{\partial \underset{\sim}{x}}(\underset{\sim}{x})$$

is sparse, symmetric, and positive definite.

A very important method for the solution of (1.1) is Newton's method (cf. Ortega and Rheinboldt [O1]). For a given initial guess $\underset{\sim}{x}^{(0)}$, we generate a sequence of iterates $\underset{\sim}{x}^{(1)}, \underset{\sim}{x}^{(2)}, \underset{\sim}{x}^{(3)}, \ldots$ by

$$\underset{\sim}{x}^{(k+1)} = \underset{\sim}{x}^{(k)} - \underset{\sim}{\delta}^{(k)}, \tag{1.2}$$

where $\underset{\sim}{\delta}^{(k)}$ is the solution to the sparse linear system

$$F'(\underset{\sim}{x}^{(k)})\, \underset{\sim}{\delta}^{(k)} = F(\underset{\sim}{x}^{(k)}). \tag{1.3}$$

Under suitable assumptions on $\underset{\sim}{x}^{(0)}$ and F, the sequence is well-defined and converges quadratically to $\underset{\sim}{x}^*$, i.e.,

$$\exists c < 1 \text{ such that } \|\underset{\sim}{x}^{(k+1)} - \underset{\sim}{x}^*\| \le c^{2^k} \|\underset{\sim}{x}^{(k)} - \underset{\sim}{x}^*\|.^{\dagger\ \dagger\dagger}$$

A straightforward implementation of Newton's method could use sparse Gaussian elimination to solve the linear systems which arise. Unfortunately, this would be inefficient computationally because at each step it would require the computation of $F(\underset{\sim}{x}^{(k)})$ and $F'(\underset{\sim}{x}^{(k)})$ and the exact solution of (1.3). Also, a large amount of storage would be required for the factorization of $F'(\underset{\sim}{x}^{(k)})$.

A number of modifications have been proposed to reduce some of the costs of Newton's method, notably the simplified Newton's or chord method and a variety of Newton-iterative methods (cf. Ortega and Rheinboldt [O1]). For certain problems, these methods may require less storage or work than Newton's method. However, computational difficulties are typical in the solution of nonlinear systems, and each of these methods has its disadvantages. The

---

† Throughout this chapter we use $\|\cdot\|$ to denote the Euclidean norm.

†† In the notation of Ortega and Rheinboldt [O1], this condition defines R-quadratic convergence. This is slightly weaker than the more standard definition of Q-quadratic convergence, i.e.,

$$\exists c' < 1 \text{ such that } \|\underset{\sim}{x}^{(k+1)} - \underset{\sim}{x}^*\| \le c' \|\underset{\sim}{x}^{(k)} - \underset{\sim}{x}^*\|^2 .$$

choice of method for a specific nonlinear system (1.1) depends quite heavily on the properties of F and on the storage and time constraints imposed by the computational environment.

The simplified Newton's method replaces $F'(x^{(k)})$ with $F'(x^{(0)})$ in (1.3). This avoids the computation of $F'(x^{(k)})$ at each step and reduces the cost of solving (1.3) for $k \geq 1$, since, if $F'(x^{(0)})$ is factored at the first step, each succeeding step only requires a backsolution. However, the chord method converges only linearly† and requires the same amount of storage as Newton's method.

The Newton-iterative methods use a linear iterative method (such as SOR, cf. Ortega and Rheinboldt [O1], p. 215) to approximate the solution of (1.3) instead of solving it exactly. The sequence of iterates generated by such methods depends upon the particular iterative method chosen and the criteria used to stop the inner iteration. The advantages of Newton-iterative methods are that they require only a small amount of storage and that, when $x^{(k)}$ is near x^*, they can take advantage of the fact that zero is a good initial approximation to $\delta^{(k)}$. The problem with Newton-iterative methods is that their rate of convergence is essentially determined by the rate of convergence of the inner (linear) iterative method applied to the linearized problem (1.3). Thus they encounter all of the usual difficulties with linear iterative methods (see Chapter I).

† The iterates $x^{(k)}$ converge linearly to x^* if
$$\exists c < 1 \text{ such that } \|x^{(k+1)} - x^*\| \leq c\,\|x^{(k)} - x^*\| .$$

In this chapter we present the Newton-Richardson methods (cf. Eisenstat, Schultz, and Sherman [E4]), a class of Newton-iterative methods which use a Richardson-D'Jakonov iteration (cf. Richardson [R2], D'Jakonov [D3]) to approximate the solution of (1.3) at each step. The Newton-Richardson methods require the same amount of storage as Newton's method or the chord method, but if 2^k steps of the inner iteration are taken at the k-th step, they will converge quadratically with less work per step (on average) than with Newton's method. And as compared to most other Newton-iterative methods, the Newton-Richardson methods have the advantage that their rate of convergence is essentially independent of N.

In the next section we describe a model semilinear partial differential equation which will be used to motivate the Newton-Richardson methods. In Section 3 we discuss the implementation of Newton's method and show how to derive Newton-Richardson methods for the model problem. In Section 4, we examine the general theory of Newton-iterative methods and state several local convergence results. In Section 5 we consider the ramifications of the theory for the model problem, and in Sections 6 and 7 we present the results of numerical experiments involving two semilinear partial differential equations and the minimal surface equation, respectively. Finally, in Section 8 we give the proofs of the results stated in Section 4.

VIII.2 A Model Problem

In this section we describe a model problem arising in the numerical solution of semilinear partial differential equations and show how to apply Newton's method to it. In the next section, we will motivate the Newton-Richardson methods by analyzing the costs of Newton's method and certain of its modifications for this model problem.

Let D be the unit square $(0,1)\times(0,1)$ and let ∂D denote the boundary of D. The model problem we wish to solve is the nonlinear Poisson equation

$$\begin{aligned} -\Delta v &= g(v) \quad \text{in } D \\ \text{with } v &= 0 \quad \text{on } \partial D, \end{aligned} \tag{2.1}$$

where g satisfies

$$g_v(v) \le \lambda < \Lambda$$

(where Λ is the fundamental eigenvalue of $-\Delta$ on D), so that $v(x,y)$ exists and is unique (cf. Ortega and Rheinboldt [01], pp. 110,116).

To obtain an approximate solution to (2.1), the continuous problem must be reduced to a discrete form. Several techniques are available to do this, and we consider the use of the five-point finite difference approximation introduced in Section II.3. We superimpose on D a uniform square mesh D_n, with boundary ∂D_n, and replace the differential operator $-\Delta$ in (2.1) with the

standard five-point difference approximation on D_n. Letting V_{ij} be the approximation to $v(ih,jh)$ $(h = \frac{1}{n+1})$, we obtain the system of equations:

$$\begin{aligned} -V_{i-1,j} - V_{i,j-1} - V_{i,j+1} - V_{i+1,j} + 4V_{ij} &= h^2 g(V_{ij}) \\ &\quad \text{for } (ih,jh) \in D_n \\ V_{ij} &= 0 \quad \text{for } (ih,jh) \in \partial D_n. \end{aligned} \tag{2.2}$$

Ordering the mesh points with the natural ordering, we may write these equations more concisely as an N $(= n^2)$ by N system of nonlinear equations

$$A_5 \underset{\sim}{V} = \underset{\sim}{G}(\underset{\sim}{V}), \tag{2.3}$$

where

$$\underset{\sim}{V} = \{V_{ij}: (ih,jh) \in D_n\},$$

$$\underset{\sim}{G}(\underset{\sim}{V}) = \{h^2 g(V_{ij}): (ih,jh) \in D_n\},$$

and A_5 is the five-point matrix presented in Section II.3. Under the conditions imposed on (2.1), this system of equations has a unique solution $\underset{\sim}{V}$ which is a close approximation to $v(ih,jh)$ (cf. Ortega and Rheinboldt [01], p. 112).

Newton's method may be applied in a straightforward way to solve (2.3) for $\underset{\sim}{V}$. Given an initial guess $\underset{\sim}{V}^{(0)}$, we generate a sequence of successive approximations $\underset{\sim}{V}^{(1)}, \underset{\sim}{V}^{(2)}, \underset{\sim}{V}^{(3)}, \ldots$ to $\underset{\sim}{V}$ by

$$\underset{\sim}{V}^{(k+1)} = \underset{\sim}{V}^{(k)} - \underset{\sim}{\delta}^{(k)},$$

where $\delta^{(k)}$ is the solution of the system of linear equations

$$J(V^{(k)})\,\delta^{(k)} = A_5 V^{(k)} - G(V^{(k)}) \qquad (2.4)$$

and

$$J(V^{(k)}) = A_5 - \frac{\partial G}{\partial V}(V^{(k)}).$$

Under the assumptions made above, $J(V^{(k)})$ is an $N \times N$ sparse, symmetric, positive definite matrix having the same zero structure as A_5. If $V^{(0)}$ is sufficiently close to V, then Newton's method converges quadratically to the solution of (2.3) independent of h (cf. Greenspan and Parter [G6]), and only $O(\log \log N)$ Newton iterations are required to reduce the initial error $\|V^{(0)} - V\|$ by a factor of $h^2 = \frac{1}{N}$, the discretization error.

VIII.3 Newton-Richardson Methods for the Model Problem

As we saw in the last section, each step of Newton's method for the model problem requires the solution of a sparse symmetric linear system. Thus Newton's method really amounts to the solution of a sequence of related linear systems, and its costs are directly related to the manner in which these systems are solved. In this section we consider a number of approaches to solving the systems (2.4), and we introduce Newton-Richardson methods for the model problem (cf. Eisenstat, Schultz, and Sherman [E4]).

In the standard implementation of Newton's method for the model problem, the linear systems which arise can be solved with sparse symmetric Gaussian elimination using the techniques of

Chapter V. A numeric factorization is required for each step, but, since the zero structure of $J(V^{(k)})$ does not change from step to step, the symbolic factorization is only needed at the first step. Since (2.4) has the same zero structure as (II.3.5), its solution requires $O(N \log N)$ storage locations and $O(N^{3/2})$ arithmetic operations (cf. Section IV.3). For $V^{(0)}$ sufficiently close to V, Newton's method converges quadratically, and the initial error $\|V^{(0)} - V\|$ can be reduced by a factor of $h^2 = \frac{1}{N}$ in $O(\log \log N)$ steps at a total cost of $O(N \log N)$ storage locations and $O(N^{3/2} \log \log N)$ arithmetic operations.†

One of the reasons that Newton's method requires so many arithmetic operations is that it does not exploit the fact that the iterates $V^{(k)}$ are converging quadratically to V. The sequence of systems (2.4) is solved as if the systems were unrelated, when, in fact, their solutions rapidly approach zero as $V^{(k)}$ converges to V. To exploit this situation, we could use an iterative method to approximate the solution of (2.4) at each step. Most Newton-iterative methods require only $O(N)$ storage locations for the model problem, and some require fewer arithmetic operations than Newton's method. Unfortunately, however, for the model problem the rates of convergence of most Newton-iterative methods depend on the mesh spacing h (cf. Section 5) in such a way that they have higher asymptotic arithmetic costs than Newton's method.

---

† It is also possible to use the minimal storage techniques of Chapter VII. This would reduce the storage requirement to $O(N)$ locations, while not more than doubling the arithmetic cost.

For example, even though Newton-SOR converges linearly, the rate of convergence depends on h, and the method requires $O(N^{1/2}\log N)$ iterations to reduce the error by a factor of h^2 (see Section 5). If the rate of convergence were independent of h, then only $O(\log N)$ iterations would be required.

We now present the Newton-Richardson methods for the model problem. These methods require more storage than other Newton-iterative methods ($O(N \log N)$ locations), but they converge at rates which are independent of h. The methods are based on the principles embodied in the Strongly Implicit Procedures developed by Stone [S3], Dupont [D8], Diamond [D2], and others. The basic idea is to use sparse Gaussian elimination to obtain $\underset{\sim}{\delta}^{(0)}$ at the first step and to compute $\underset{\sim}{\delta}^{(k)}$ at succeeding steps with a Richardson-D'Jakonov iteration (cf. Richardson [R2], D'Jakonov [D3]) involving the factorization of $J(\underset{\sim}{V}^{(0)})$.

In particular, suppose that at the k+1-st step $(k \geq 1)$ we wish to solve the system

$$J(\underset{\sim}{V}^{(k)})\, \underset{\sim}{\delta}^{(k)} = \underset{\sim}{b}^{(k)}. \tag{3.1}$$

Since we have already factored $J(\underset{\sim}{V}^{(0)})$ at the first step, the system

$$J(\underset{\sim}{V}^{(0)})\, \underset{\sim}{w} = \underset{\sim}{d}$$

can be solved with $O(N \log N)$ arithmetic operations. We can then solve (3.1) by using a Richardson-D'Jakonov iteration to generate a sequence of iterates $\underset{\sim}{\delta}_0^{(k)}, \underset{\sim}{\delta}_1^{(k)}, \underset{\sim}{\delta}_2^{(k)}, \ldots$ which converges to $\underset{\sim}{\delta}^{(k)}$. (The idea here is similar to that of iterative improvement, cf. Forsythe and Moler [F1], pp. 49-54). For $\underset{\sim}{\delta}_0^{(k)} = 0$ and a fixed real constant γ, the Richardson-D'Jakonov iterates satisfy

$$\underset{\sim}{\delta}_{i+1}^{(k)} = \underset{\sim}{\delta}_i^{(k)} - \gamma \underset{\sim}{\varepsilon}_i^{(k)},$$

where $\underset{\sim}{\varepsilon}_i^{(k)}$ is the solution to the linear system

$$J(\underset{\sim}{V}^{(0)})\, \underset{\sim}{\varepsilon}_i^{(k)} = J(\underset{\sim}{V}^{(k)})\, \underset{\sim}{\delta}_i^{(k)} - \underset{\sim}{b}^{(k)}.$$

Each step of the iteration requires $O(N \log N)$ arithmetic operations, and the iterates will converge linearly to $\underset{\sim}{\delta}^{(k)}$ if

$$\rho = \left\| I - \gamma J(\underset{\sim}{V}^{(0)})^{-1} J(\underset{\sim}{V}^{(k)}) \right\| < 1$$

(cf. Young [Y1], p. 77).

To maximize the rate of convergence, we choose γ to minimize ρ. If all the eigenvalues of $J(\underset{\sim}{V}^{(0)})^{-1} J(\underset{\sim}{V}^{(k)})$ lie in the interval $[\alpha, \beta]$, this is accomplished by choosing

$$\gamma = \frac{2}{\alpha+\beta},$$

yielding

$$\rho = (\beta - \alpha)/(\alpha + \beta).$$

For the model problem, ρ can be made independent of h by choosing $\underset{\sim}{V}^{(0)}$ near enough to $\underset{\sim}{V}$ (cf. Section 5), so the Richardson-D'Jakonov iteration will converge linearly, independent of the mesh size.†

---

† It has been shown by several authors (cf. Young [Y1]) that the rate of convergence is increased by a constant factor by using Tchebychev acceleration to choose a sequence of parameters $\{\gamma_k\}$ instead of the single parameter γ.

Different Newton-Richardson methods are obtained by varying the number of Richardson-D'Jakonov iterates computed at each step of the outer iteration. Of particular interest are the case in which 2^k iterates are computed at the k-th step and the case in which just one iterate is computed at each step. In the next section, we will show that if $\underline{V}^{(0)}$ is sufficiently close to $\underline{V}$, then the first of these methods converges quadratically to $\underline{V}$, while the second, which reduces to a relaxed chord method, converges linearly. However, in either case the solution of the model problem requires a total of $O(N(\log N)^2)$ arithmetic operations in addition to the cost of the initial factorization of $J(\underline{V}^{(0)})$ (which we treat as preprocessing).

VIII.4 Convergence Results

In this section we present an analysis of the convergence properties of Newton-iterative methods. We will show that it is possible to adjust the rate of convergence of such methods by varying the number of inner iterations taken at each step and that by taking 2^k inner iterations at the k-th step, quadratic convergence is obtained. Since the Newton-Richardson methods described in the last section are a particular instance of Newton-iterative methods, the convergence results for them follow immediately. The proofs of the results described here are contained in Section 8.

If R is a bounded subset of $\mathbb{R}^N$ and $F: R \to \mathbb{R}^N$ is a

nonlinear function, we consider the solution of

$$F(\underline{x}) = 0, \tag{4.1}$$

where we assume that a solution $\underline{x}^*$ exists. We also assume that there is an $r_0 > 0$ such that if $S_0 = \{\underline{x}: \|\underline{x} - \underline{x}^*\| < r_0\}$, then

(i) F is differentiable in S_0;

(ii) $F'(\underline{x}) = \frac{\partial F}{\partial \underline{x}}(\underline{x})$ is nonsingular at $\underline{x}^*$;

(iii) There is an $L < +\infty$ such that for $\underline{x} \in S_0$,

$$\|F'(\underline{x}) - F'(\underline{x}^*)\| \le L \|\underline{x} - \underline{x}^*\| .$$

It can be shown that the discretized model problem of Section 2 satisfies (i) - (iii) (cf. Greenspan and Parter [G6]), so that we may directly apply the theory developed here. In the next section we will consider the model problem as a practical illustration of the theory.

From assumptions (i) - (iii), it follows that there is a $0 < r_1 \le r_0$ such that F' is continuous and nonsingular in $S_1 = \{\underline{x}: \|\underline{x} - \underline{x}^*\| < r_1\}$ (cf. Ortega and Rheinboldt [O1], p. 46). Thus Newton's method can be used to solve (4.1) for $\underline{x}^*$. For a given $\underline{x}^{(0)} \in S_1$, we generate the sequence of iterates $\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \ldots$ by

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - \underline{\delta}^{(k)}, \tag{4.2}$$

where

$$F'(\underline{x}^{(k)})\, \underline{\delta}^{(k)} = F(\underline{x}^{(k)}). \tag{4.3}$$

It can be shown that for some $0 < r_2 \le r_1$, Newton's method con-

verges quadratically to $\underset{\sim}{x}^*$ for $\underset{\sim}{x}^{(0)} \in S_2 = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r_2\}$ (cf. Ortega and Rheinboldt [O1], p. 312).

Newton-Richardson methods are a specific instance of general Newton-iterative methods in which a linear iterative method is used to approximate the solution of (4.3) at each step. A common way of generating Newton-iterative methods involves splitting $F'(\underset{\sim}{x})$ by writing it as

$$F'(\underset{\sim}{x}) = B(\underset{\sim}{x}) - C(\underset{\sim}{x}), \tag{4.4}$$

where

(iv) $B(\underset{\sim}{x})$ is continuous at $\underset{\sim}{x}^*$ and

(v) $B(\underset{\sim}{x}^*)$ is nonsingular.

Furthermore, if the splitting (4.4) is to lead to a computationally efficient method, it must be possible to quickly solve the system

$$B(\underset{\sim}{x})\, \underset{\sim}{w} = \underset{\sim}{d}. \tag{4.5}$$

It follows from (iv) - (v) that there is a $0 < r_3 \le r_2$ such that B is continuous and nonsingular in $S_3 = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r_3\}$ (cf. Ortega and Rheinboldt [O1], p. 46), and if we let

$$H(\underset{\sim}{x}) = B(\underset{\sim}{x})^{-1} C(\underset{\sim}{x})$$

$$= I - B(\underset{\sim}{x})^{-1} F'(\underset{\sim}{x}),$$

it then follows that H is continuous in S_3.

The Newton-Richardson methods are obtained with the splitting (4.4) in which B and C are defined by

$$B(\underset{\sim}{x}) = \frac{1}{\gamma} F'(\underset{\sim}{x}^{(0)}), \quad C(\underset{\sim}{x}) = B(\underset{\sim}{x}) - F'(\underset{\sim}{x}),$$

where γ is a real constant satisfying $0 < \gamma < 2$ and $\underset{\sim}{x}^{(0)} \in S_1$. Clearly this splitting satisfies (iv) and (v), and once $F'(\underset{\sim}{x}^{(0)})$ has been factored at the first step, the system (4.5) can be solved quickly.

At each step of a Newton-iterative method, we use the linear iteration to generate a sequence of inner iterates $\underset{\sim}{\delta}_0^{(k)}, \underset{\sim}{\delta}_1^{(k)}, \underset{\sim}{\delta}_2^{(k)}, \ldots$ in which $\underset{\sim}{\delta}_0^{(k)} = 0$ and

$$\underset{\sim}{\delta}_{i+1}^{(k)} = \underset{\sim}{\delta}_i^{(k)} - \underset{\sim}{\varepsilon}_i^{(k)},$$

where

$$B(\underset{\sim}{x}^{(k)})\, \underset{\sim}{\varepsilon}_i^{(k)} = F'(\underset{\sim}{x}^{(k)})\, \underset{\sim}{\delta}_i^{(k)} - F(\underset{\sim}{x}^{(k)}).$$

For any positive integer m, let

$$A_m(\underset{\sim}{x}) = (I + H(\underset{\sim}{x}) + H(\underset{\sim}{x})^2 + \ldots + H(\underset{\sim}{x})^{m-1})\, B(\underset{\sim}{x})^{-1}$$

$$= (I - H(\underset{\sim}{x})^m)(I - H(\underset{\sim}{x}))^{-1}\, B(\underset{\sim}{x})^{-1}$$

$$= (I - H(\underset{\sim}{x})^m)\, F'(\underset{\sim}{x})^{-1}.$$

Then letting

$$B_k = B(\underset{\sim}{x}^{(k)})$$

and

$$H_k = H(\underset{\sim}{x}^{(k)}),$$

we have

$$\begin{aligned}
\delta_{i+1}^{(k)} &= \delta_i^{(k)} - B_k^{-1}(F'(x^{(k)})\,\delta_i^{(k)} - F(x^{(k)})) \\
&= H_k\,\delta_i^{(k)} + B_k^{-1}\,F(x^{(k)}) \\
&= H_k[H_k\,\delta_{i-1}^{(k)} + B_k^{-1}\,F(x^{(k)})] + B_k^{-1}\,F(x^{(k)}) \\
&= H_k^2\,\delta_{i-1}^{(k)} + (I + H_k)B_k^{-1}\,F(x^{(k)}) \\
&= H_k^{i+1}\,\delta_0^{(k)} + (I + H_k + \dots + H_k^i)B_k^{-1}\,F(x^{(k)}) \\
&= (I + H_k + \dots + H_k^i)B_k^{-1}\,F(x^{(k)}) \\
&= A_{i+1}(x^{(k)})F(x^{(k)}).
\end{aligned}$$

At the k-th Newton-iterative step, we define

$$\delta^{(k)} = \delta_{m_k}^{(k)},$$

for some positive integer m_k. Letting

$$G_m(x) = x - A_m(x)F(x),$$

we have

$$\begin{aligned}
x^{(k+1)} &= x^{(k)} - \delta^{(k)} \\
&= G_{m_k}(x^{(k)}).
\end{aligned} \tag{4.6}$$

If $\|H(x^*)\| < 1$, then under assumptions (i) - (iii) it can be shown that for some $0 < r_4 \le r_3$, the outer iterates $\{x^{(k)}\}$ defined by (4.6) exist and converge to x^* for $x^{(0)} \in S_4 = \{x: \|x - x^*\| < r_4\}$ (cf. Ortega and Rheinboldt [01], pp. 350-351).†

---

† It actually suffices that the spectral radius of $H(x^*)$ satisfy $\rho(H(x^*)) < 1$, but in order to avoid problems in relating norms to

The rate at which the Newton-iterative iterates converge to x^* depends mainly on $\|H(x^*)\|$ and the sequence $\{m_k\}$. To obtain our main convergence result, we make two further assumptions on the splitting (4.4):

(vi) For some constant λ, $0 < \lambda < 1$, $\|H(x^*)\| < \lambda$;

(vii) There are an $L_1 < +\infty$ and a $0 < r_5 \le r_4$ such that for $x \in S_5 = \{x: \|x - x^*\| < r_5\}$,

$$\|B(x) - B(x^*)\| \le L_1 \|x - x^*\| .$$

We now state our main convergence result, deferring its proof until Section 8.

<u>Theorem 4.1</u>: Let F satisfy (i) - (iii) and let $F'(x) = B(x) - C(x)$ be a splitting which satisfies (iv) - (vii). Let $m_0, m_1, m_2, \dots$ be a sequence of positive integers, and define

$$m = \max\,\{\{1, m_0\} \cup \{(m_k - \sum_{\ell=0}^{k-1} m_\ell): k = 1,2,3,\dots\}\}.$$

Then, if $m < +\infty$, there are a $0 < r \le r_5$ (depending on λ, H, and F) and a positive constant c (depending on λ and m) with $\lambda < c < 1$ such that for $x^{(0)} \in S = \{x: \|x - x^*\| < r\}$, the sequence of iterates defined by (4.5) satisfies

$$\|x^{(k+1)} - x^*\| \le c^{m_k} \|x^{(k)} - x^*\| .$$

---

the spectral radius, we give all our results in terms of the Euclidean norm. (If $H(x^*)$ is symmetric, then of course we have $\|H(x^*)\| = \rho(H(x^*))$.)

If we choose either $m_k = 1$ or $m_k = 2^k$, then $m = 1$ in Theorem 4.1. Thus we have the following corollaries.

Corollary 4.2: Let F satisfy (i) - (iii) and let $F'(\underset{\sim}{x}) = B(\underset{\sim}{x}) - C(\underset{\sim}{x})$ be a splitting which satisfies (iv) - (vii). Define the sequence $\{m_k: k = 0,1,2,\ldots\}$ by $m_k = 1$. Then there are a $0 < r \le r_5$ (depending on λ, H, and F) and a positive constant c (depending on λ) with $\lambda < c < 1$ such that for $\underset{\sim}{x}^{(0)} \in S = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r\}$, the sequence of iterates defined by (4.5) converges linearly, i.e.,

$$\|\underset{\sim}{x}^{(k+1)} - \underset{\sim}{x}^*\| \le c\|\underset{\sim}{x}^{(k)} - \underset{\sim}{x}^*\| .$$

Corollary 4.3: Let F satisfy (i) - (iii) and let $F'(\underset{\sim}{x}) = B(\underset{\sim}{x}) - C(\underset{\sim}{x})$ be a splitting which satisifes (iv) - (vii). Define the sequence $\{m_k: k = 0,1,2,\ldots\}$ by $m_k = 2^k$. Then there are a $0 < r \le r_5$ (depending on λ, H, and F) and a positive constant c (depending on λ) with $\lambda < c < 1$ such that for $\underset{\sim}{x}^{(0)} \in S = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r\}$, the sequence of iterates defined by (4.5) converges quadratically, i.e.,

$$\|\underset{\sim}{x}^{(k+1)} - \underset{\sim}{x}^*\| \le c^{2^k}\|\underset{\sim}{x}^{(k)} - \underset{\sim}{x}^*\| .$$

VIII.5 Ramifications of the Theory for the Model Problem

Corollary 4.2 expresses the previously known result that a Newton-iterative method converges linearly for $m_k = 1$ (cf. Ortega and Rheinboldt [O1], p. 350). The real import of Theorem 4.1

is the rather surprising result of Corollary 4.3, i.e, that we can obtain R-quadratic convergence with a Newton-iterative method by choosing $m_k = 2^k$. Since Newton's method itself is only quadratically convergent (cf. Ortega and Rheinboldt [O1], p. 312) this is the fastest convergence that can be expected for any Newton-iterative method.

A problem with the result of Theorem 4.1 is that c depends on λ: specifically, $\lambda < c < 1$. To reduce the initial error $\|\underset{\sim}{v}^{(0)} - \underset{\sim}{v}\|$ by a factor of K requires ℓ iterations, where

$$K \approx \sum_{k=0}^{\ell-1} c^{m_k} . \tag{5.1}$$

Taking logarithms, we obtain

$$\log K \approx \left(\sum_{k=0}^{\ell-1} m_k\right)(\log c),$$

so that ℓ satisfies

$$\sum_{k=0}^{\ell-1} m_k \approx \log K/\log c \ge \log K/\log \lambda .$$

As an illustration, we consider the model problem of Section 2. For Newton-SOR methods it can be shown that even with the optimal SOR parameter ω, $\log \lambda \approx -2\pi h$ (cf. Young [Y1], p. 190). Reducing the initial error by a factor of the discretization error h^2 requires ℓ iterations, where ℓ satisfies

$$\sum_{k=0}^{\ell-1} m_k \approx \log h^2/\log \lambda$$

$$\approx (-\log N)/(-2\pi h)$$

$$= O(N^{1/2}\log N).$$

Thus if $m_k = 2^k$, we have

$$\sum_{k=0}^{\ell-1} m_k \approx 2^\ell = O(N^{1/2}\log N),$$

so that $\ell = O(\log N)$. And if $m_k = 1$, then we have

$$\sum_{k=0}^{\ell-1} m_k = \ell = O(N^{1/2}\log N).$$

In either case, since each inner iteration requires $O(N)$ operations, a total of $O(N^{3/2}\log N)$ iteration time is required, as compared to the $O(N^{3/2}\log\log N)$ time required by Newton's method.

For Newton-Richardson methods, however, the situation is quite different. We have

$$\|H(x^*)\| = \|B(x^*)^{-1} C(x^*)\|$$

$$= \|\gamma F'(x^{(0)})^{-1} [(\tfrac{1}{\gamma})F'(x^{(0)}) - F'(x^*)]\|$$

$$= \|I - \gamma F'(x^{(0)})^{-1} F'(x^*)\| .$$

Since $F'(x)^{-1}$ is continuous in S_3, we may make $\|H(x^*)\|$ arbitrarily close to $|1-\gamma|$ by choosing $\|x^{(0)} - x^*\|$ small enough.

In particular, we can always make $\|H(x^*)\| < (1 + |1-\gamma|)/2 < 1$. Thus for the model problem, the constants λ and c of Theorem 4.1 are independent of h for Newton-Richardson methods. To reduce the initial error by a factor of h^2 requires ℓ iterations, where

$$\sum_{k=0}^{\ell-1} m_k \approx \log h^2/\log c \approx \log N/\log c$$

and $\log c$ is a constant independent of N. Hence if $m_k = 2^k$, we have

$$\sum_{k=0}^{\ell-1} m_k \approx 2^\ell \approx \log N/\log c,$$

so that $\ell \approx \log\log N$. And if $m_k = 1$, then we have

$$\sum_{k=0}^{\ell-1} m_k = \ell \approx \log N/\log c.$$

This leads to the following result on the cost of solving the model problem with Newton-Richardson methods (cf. Eisenstat, Schultz, and Sherman [E4]).

<u>Theorem 5.1</u>: Consider the use of Newton-Richardson methods to solve the model semilinear problem presented in Section 2, and assume that $v^{(0)}$ and γ are chosen so that Theorem 4.1 holds for c independent of h. Then ignoring the costs of preprocessing and the factorization at the first step, we may reduce the initial error $\|v^{(0)} - v\|$ by a factor of h^2 in $O(N(\log N)^2)$ iteration time by choosing either $m_k = 2^k$ or $m_k = 1$.

Proof: Each inner Richardson-D'Jakonov iteration requires $O(N \log N)$ time. If $m_k = 2^k$, then we require $O(\log \log N)$ Newton-Richardson steps to reduce the initial error by a factor of h^2. Thus the total iteration time is

$$O\left(\sum_{k=1}^{\log \log N} 2^k N \log N\right) = O(N(\log N)^2).$$

On the other hand, if $m_k = 1$, then we need $O(\log N)$ Newton-Richardson steps to reduce the initial error by a factor of h^2. Again the total iteration time is $O(N(\log N)^2)$. □

From Theorems 4.1 and 5.1 we conclude that the choice of m_k for Newton-Richardson methods should be based on the relative costs of function and derivative evaluations. For the model problem, choosing $m_k = 2^k$ requires $O(\log \log N)$ function evaluations and $O(\log \log N)$ derivative evaluations, while choosing $m_k = 1$ requires $O(\log N)$ function evaluations and only one derivative evaluation (i.e. $F'(\underset{\sim}{v}^{(0)})$). Since function evaluations and derivative evaluations are equally costly for the model problem ($\frac{\partial G}{\partial \underset{\sim}{v}}(\underset{\sim}{v}^{(k)})$ is a diagonal matrix), it seems best to choose $m_k = 2^k$. However, we have observed experimentally that for other problems it is often more efficient computationally to choose $m_k = 1$ (cf. Sections 6 and 7). The reason for this is that derivative evaluations are often far more costly than function evaluations for problems more general than the model problem.

VIII.6 Numerical Results for the Model Problem

In this section, we illustrate the Newton-Richardson methods by applying them to general semilinear problems of the form

$$(pv_x)_x + (qv_y)_y = f(x,y,v)$$

over the domain

$$D = (0,1)\times(0,1) - [\tfrac{3}{8}, \tfrac{5}{8}]\times[\tfrac{3}{8}, \tfrac{5}{8}]$$

(i.e. the unit square with a hole cut out). In particular we consider two problems given by Bartels and Daniel [B2] (also cf. Eisenstat, Schultz, and Sherman [E4]):

$$\{(1 + x^2 + y^2)v_x\}_x + \{(1 + e^x + e^y)v_y\}_y = g(x,y)e^v \text{ in } D \quad (6.1a)$$

$$v(x,y) = x^2 + y^2 \text{ on } \partial D \quad (6.1b)$$

and

$$\{(1 + (x-y)^2)v_x\}_x + \{(10 + e^{xy})v_y\}_y = g(x,y)(1 + v)^3 \text{ in } D \quad (6.2a)$$

$$v(x,y) = x^2 + y^2 \text{ on } \partial D, \quad (6.2b)$$

where in each case $g(x,y)$ is chosen to make the unique exact solution

$$v(x,y) = x^2 + y^2.$$

To solve either of these problems, we cover $(0,1)\times(0,1)$ with a square mesh D_n and discretize D with a mesh $\tilde{D}_n = D_n \cap D$ having boundary $\partial\tilde{D}_n = \partial D \cap \{D_n \cup \partial D_n\}$. We then replace the differential operator with the standard five-point finite difference approximation in D_n (cf. Bartels and Daniel [B2]). This leads

to a system of semilinear equations similar to the model problem of Section 2, and we can solve it with either Newton's method or a Newton-Richardson method. At each step, the Jacobian matrix is sparse, symmetric, positive definite and irreducible and has a structure similar to the coefficient matrix of the five-point model problem of Chapter II.

For our tests we solved the discretized forms of (6.1) and (6.2) with Newton's method and with Newton-Richardson methods using both $m_k = 2^k$ and $m_k = 1$. For the Newton-Richardson methods, we chose $\gamma = 1$, and in all cases we chose the initial approximation to be zero in D and to satisfy the boundary conditions. We used two different stopping criteria in our tests: either stopping when the 2-norm of the nonlinear residual had been reduced by a factor of 10^{-11} or when the 2-norm of the error in the computed solution had been reduced by a factor of h^2 $(h = \frac{1}{n+1})$. The first of these is a possible stopping criterion in practice (where the actual solution is not known); the second is the stopping criterion used by Bartels and Daniel [B2].

Our results are summarized in Tables 6.1 and 6.2. By way of comparison, we also give in Table 6.3 the corresponding results for the solution of the problems with the Nonlinear Conjugate Gradient method (NCG) proposed by Bartels and Daniel [B2]. All times are in seconds and reflect execution times on a CDC-6600. Our times were obtained with a FORTRAN program using the RUN compiler under the KRONOS operating system.

The results are clear: even including preprocessing time (i.e.

Problem	(4.1) h = 1/16			(4.1) h = 1/32			(4.2) h = 1/16			(4.2) h = 1/32		
Method	N	N-R	N-R	N	N-R	N-R	N	N-R	N-R	N	N-R	N-R
m_k	—	2^k	1	—	2^k	1	—	2^k	1	—	2^k	1
Number of Iterations to reduce residual by factor of 10-11	4	4	9	3	4	8	4	4	12	4	4	12
Preprocessing Time (Ordering and Symbolic Factorization)	.654	.651	.658	3.940	3.928	3.940	.636	.634	.635	3.886	3.958	3.867
Time for First Iteration	.265	.271	.269	2.228	2.199	2.188	.253	.253	.254	2.190	2.198	2.124
Total Iteration Time	1.072	1.369	1.225	6.672	8.461	6.550	1.016	1.309	1.398	8.745	8.370	8.217
Total Time	1.754	2.050	1.948	10.659	12.455	10.611	1.680	1.975	2.116	12.690	12.391	12.270

N = Newton's Method
N-R = Newton-Richardson Method

Results for Newton-Richardson Methods reducing the residual by a factor of 10^{-11}

TABLE 6.1

Problem	Method	m_k	Number of Iterations to reduce error by factor of h^2	Preprocessing Time (Ordering and Symbolic Factorization)	Time for First Iteration	Total Iteration Time	Total Time
(4.1) h = 1/16	N	—	2	.662	.265	.540	1.218
	N-R	2^k	2	.658	.275	.464	1.138
	N-R	1	2	.653	.269	.388	1.058
(4.1) h = 1/32	N	—	2	3.983	2.205	4.392	8.406
	N-R	2^k	2	3.935	2.214	3.244	7.209
	N-R	1	3	3.869	2.195	3.437	7.382
(4.2) h = 1/16	N	—	2	.635	.256	.510	1.160
	N-R	2^k	2	.640	.255	.425	1.079
	N-R	1	3	.650	.269	.486	1.160
(4.2) h = 1/32	N	—	2	3.961	2.217	4.421	8.416
	N-R	2^k	3	3.950	2.201	4.963	8.964
	N-R	1	4	3.889	2.157	3.825	7.777

N = Newton's Method

N-R = Newton-Richardson Method

Results for Newton-Richardson Methods reducing error by a factor of h^2

TABLE 6.2

the cost of ordering the mesh points and computing the symbolic factorization at the first step), both Newton's method and the Newton-Richardson methods are substantially faster than NCG. Using identical stopping criteria (see Tables 6.2 and 6.3), the methods are approximately twice as fast as NCG, and in the time used by NCG to reduce the error by a factor of h^2, they obtain far greater accuracy (see Table 6.1). With either stopping criterion, the Newton-Richardson methods are competitive with the straightforward implementation of Newton's method, becoming somewhat faster when $h = \frac{1}{32}$. For these problems, there is little to choose between the two variants of the Newton-Richardson method, since the cost associated with computing the Jacobian at each step is quite small. In the next section, however, we consider a problem where the advantage of choosing $m_k = 1$ is quite apparent.

Problem	(4.1) h = 1/16	(4.1) h = 1/32	(4.2) h = 1/16	(4.2) h = 1/32
Number of Iterations to reduce error by factor of h^2	6	8	9	12
Preprocessing Time	.394	2.90	.398	2.82
Total Iteration Time	2.08	9.63	2.58	11.3
Total Time	2.57	12.9	3.05	14.4

Results for Nonlinear Conjugate Gradient

TABLE 6.3

VIII.7 Numerical Results for the Minimal Surface Equation

In this section we apply Newton-Richardson methods to the solution of the minimal surface or Plateau's equation (cf. Ortega and Rheinboldt [O1], p. 31). Our results illustrate the behavior of the methods for problems more nonlinear than the model problem and show that they are competitive with Newton's method and other efficient methods for this problem. For purposes of comparison in this regard, we include results due to Concus [C4] on the solution of the minimal surface equation with block nonlinear SOR.

Let $D = (0,1)\times(0,1)$. We wish to solve the equation

$$\frac{\partial}{\partial x}\left[\psi(|\nabla v|^2)\frac{\partial v}{\partial x}\right] + \frac{\partial}{\partial y}\left[\psi(|\nabla v|^2)\frac{\partial v}{\partial y}\right] = 0 \qquad (7.1a)$$

in D, where

$$\psi(t) = (1+t)^{1/2}. \qquad (7.1b)$$

On the boundaries of D we specify

$$v = 0 \quad \text{for} \quad x = 0 \quad \text{or} \quad y = 1,$$

$$v = K \sin(1/2\ \pi x) \quad \text{for} \quad y = 0, \qquad (7.1c)$$

$$\frac{\partial v}{\partial x} = 0 \quad \text{for} \quad x = 1.$$

The amplitude K in (7.1c) affects the convergence properties of methods to solve this equation (cf. Concus [C3] for a detailed discussion), and Newton's method and the Newton-Richardson methods seem somewhat more sensitive to the value of K than the block nonlinear SOR method used by Concus. For convenience, however, we

will avoid this problem by considering only values of K for which all these methods converge quite readily.

Following Concus [C3], we derive a discrete form of (7.1) by placing a square grid D_n of meshwidth $h = \frac{1}{n+1}$ on D and approximating the continuous equation with a nonlinear difference equation at each mesh point. The resulting set of equations approximates (7.1) locally to $O(h^2)$ accuracy (cf. Concus [C3], Forsythe and Wasow [F2]).

Letting V_{ij} denote the approximation to $v(ih, jh)$ and ordering the mesh points in the natural ordering, we may write the difference equations as a system of $N = n^2 + n$ nonlinear equations:

$$\underset{\sim}{F}(\underset{\sim}{V}) = 0, \qquad (7.2)$$

where

$$\underset{\sim}{V} = \{V_{ij}: 1 \le i \le n+1,\ 1 \le j \le n\}.$$

In order to solve (7.2) with either Newton's method or a Newton-Richardson method, we will need to use the $N\times N$ Jacobian matrix

$$F'(\underset{\sim}{V}) = \frac{\partial \underset{\sim}{F}}{\partial \underset{\sim}{V}}(\underset{\sim}{V}).$$

For this problem, $F'(\underset{\sim}{V})$ is symmetric, positive definite, and irreducible (cf. Concus [C3]) and has a zero structure similar to A_9.

We obtained four different test problems by using two values of K (0.5 and 1.0) and two values of h (0.1 and 0.05). We solved the problems with Newton's method and with Newton-Richardson methods using $m_k = 2^k$ and $m_k = 1$. We also investigated a

Newton-Richardson method in which m_k was chosen adaptively based on the rate at which the iterates were converging and in which the Jacobian was refactored when it was less exepnsive to do so than to take the Newton-Richardson step with the computed m_k. However, we found that the adaptive method was no faster than the other methods, so we have not included those results. Since we chose values of K for which these methods converge fairly readily, we made no effort to optimize the choice of the parameter γ for the Newton-Richardson methods: we simply took $\gamma = 1$. Following Concus [C4] we chose the initial approximation $\underset{\sim}{v}^{(0)}$ to satisfy

$$v_{ij}^{(0)} = [K \sin(\pi ih/2) \sinh(\pi(n+1-j)h/2)] / \sinh(\pi/2),$$

which is the limiting solution as $K \to 0$ (cf. Concus [C3]). In order to facilitate comparisons with Concus' results (see below), we stopped iterating when the relative error
$\|\underset{\sim}{v}^{(k)} - \underset{\sim}{v}^{(k-1)}\| \,/\, \|\underset{\sim}{v}^{(k)}\|$ became less than 5×10^{-7}.

Our results are summarized in Table 7.1. The times are in seconds and were obtained with a FORTRAN program using the RUN compiler on a CDC-6600 under the KRONOS operating system.

There are two important features to note. First, the Newton-Richardson method with $m_k = 2^k$ converged in approximately the same number of iterations as Newton's method, just as suggested by the theory developed in Section 4. And second, the Newton-Richardson method with $m_k = 1$ is clearly the best of the methods with respect to total computation time. The reason for this is that choosing $m_k = 1$ avoids the computation of $F'(\underset{\sim}{v}^{(k)})$ for

Problem	h = 0.1, K = 0.5			h = 0.1, K = 1.0			h = 0.05, K = 0.5			h = 0.05, K = 1.0		
Method	N	N-R	N-R	N	N-R	N-R	N	N-R	N-R	N	N-R	N-R
m_k	—	2^k	1	—	2^k	1	—	2^k	1	—	2^k	1
Number of Iterations	4	4	6	5	5	15	4	4	6	5	6	22
Preprocessing Time (Ordering and Symbolic Factorization)	.152	.158	.155	.156	.156	.153	.763	.756	.753	.757	.758	.752
Time for First Iteration	.205	.205	.203	.203	.203	.202	1.490	1.504	1.496	1.500	1.495	1.492
Total Iteration Time	.817	.794	.449	1.008	1.403	..898	5.997	4.939	2.844	7.500	15.684	7.156
Total Time	.991	.977	.639	1.194	1.590	1.139	6.797	5.732	3.654	8.303	16.502	8.111

N = Newton's Method
N-R = Newton-Richardson Method

Results for Newton-Richardson Methods

TABLE 7.1

Problem	h = 0.1 K = 0.5		h = 0.1 K = 1.0		h = 0.05 K = 0.5		h = 0.05 K = 1.0	
SOR Parameter ω	1.51 (ω_{opt})†	1.50 (ω_b)‡	1.52 (ω_{opt})	1.51 (ω_b)	1.72 (ω_{opt})	1.71 (ω_b)	1.72 (ω_{opt})	1.72 (ω_b)
Number of Iterations	17	20	21	22	31	37	40	40
Total Time	.459	.540	.572	.602	3.577	4.250	4.711	4.711

† ω_{opt} = Optimal value of ω, determined empirically

‡ ω_b = Theoretically optimal asymptotic value of ω

Results for Block Nonlinear SOR

TABLE 7.2

$k > 0$.

In order to determine the real value of Newton-Richardson methods for the solution of the minimal surface equation, we compared our results with those of Concus [C4], some of which are summarized in Table 7.2. Although the comparisons for our test problems favor block nonlinear SOR, it is evident that the Newton-Richardson methods are competitive, and we believe that they will become superior for large values of n.†

---

† This is due to the fact that the rate of convergence of Newton-Richardson methods is independent of h, while that of block nonlinear SOR is not.

VIII.8 Proofs of Convergence Results

In this section we shall prove Theorem 4.1. We will make frequent use of the assumptions and results of Section 4, and the reader is advised to refer back to that section, particularly for the statements of assumptions (i) - (vii). Much of our analysis follows that of Ortega and Rheinboldt [O1], pp. 350-352.

Lemma 8.1: Let F satisy (i) - (iii) and let $F'(\underset{\sim}{x}) = B(\underset{\sim}{x}) - C(\underset{\sim}{x})$ be a splitting which satisfies (iv) - (vi). Then there is a $0 < r_6 \le r_4$ such that for $\underset{\sim}{x} \in S_6 = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r_6\}$,

$$\|H(\underset{\sim}{x})\| \le \lambda$$

and for any positive integer m,

$$\|H(\underset{\sim}{x})^m - H(\underset{\sim}{x}^*)^m\| \le m\lambda^{m-1} \|H(\underset{\sim}{x}) - H(\underset{\sim}{x}^*)\| .$$

Proof: Since H is continuous in S_4, we can choose $r_6 \le r_4$ so that $\|H(\underset{\sim}{x})\| \le \lambda$ for $\underset{\sim}{x} \in S_6$. The second part of the lemma follows by induction. It holds trivially for $m = 1$; and if it holds for $m = k$, then

$$\begin{aligned}\|H(\underset{\sim}{x})^{k+1} - H(\underset{\sim}{x}^*)^{k+1}\| &= \|H(\underset{\sim}{x})^k(H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)) + (H(\underset{\sim}{x})^k-H(\underset{\sim}{x}^*)^k)H(\underset{\sim}{x}^*)\| \\ &\le \lambda^k\|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\| + k\lambda^{k-1} \|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\| \|H(\underset{\sim}{x}^*)\| \\ &\le (k+1)\lambda^k \|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\| .\end{aligned}$$

□

Lemma 8.2: Let F satisfy (i) - (iii) and let $F'(\underset{\sim}{x}) = B(\underset{\sim}{x}) - C(\underset{\sim}{x})$ be a splitting which satisfies (iv) - (vi). Then there are posi-

tive constants $c_1 < +\infty$ and $c_2 < +\infty$ and a $0 < r_7 \le r_6$ such that for any positive integer m and $\underset{\sim}{x} \in S_7 = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r_7\}$,

$$\|G_m(\underset{\sim}{x}) - \underset{\sim}{x}^*\| \le c_1\|\underset{\sim}{x}-\underset{\sim}{x}^*\|^2 + c_2\|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| + \lambda^m\|\underset{\sim}{x}-\underset{\sim}{x}^*\| .$$

<u>Proof</u>: From (iv) - (vi) and Lemma 8.1, it follows that we may choose r_7 so that there is a $\beta < +\infty$ such that for $\underset{\sim}{x} \in S_7$, $\|H(\underset{\sim}{x})\| \le \lambda$ and $\|B(\underset{\sim}{x})^{-1}\| \le \beta$ (cf. Ortega and Rheinboldt [01], p. 351). Then for $\underset{\sim}{x} \in S_7$ we have

$$\begin{aligned}\|G_m(\underset{\sim}{x}) - \underset{\sim}{x}^*\| &\le \|\underset{\sim}{x}-A_m(\underset{\sim}{x})F(\underset{\sim}{x})-\underset{\sim}{x}^*-H(\underset{\sim}{x}^*)^m(\underset{\sim}{x}-\underset{\sim}{x}^*)\| + \|H(\underset{\sim}{x}^*)^m(\underset{\sim}{x}-\underset{\sim}{x}^*)\| \\ &= \|-A_m(\underset{\sim}{x})F(\underset{\sim}{x}) + (I-H(\underset{\sim}{x}^*)^m)(\underset{\sim}{x}-\underset{\sim}{x}^*)\| + \|H(\underset{\sim}{x}^*)^m(\underset{\sim}{x}-\underset{\sim}{x}^*)\| \\ &= \|-A_m(\underset{\sim}{x})F(\underset{\sim}{x}) + A_m(\underset{\sim}{x}^*)F'(\underset{\sim}{x}^*)(\underset{\sim}{x}-\underset{\sim}{x}^*)\| + \|H(\underset{\sim}{x}^*)^m(\underset{\sim}{x}-\underset{\sim}{x}^*)\| \\ &\le \|-A_m(\underset{\sim}{x})[F(\underset{\sim}{x})-F(\underset{\sim}{x}^*)-F'(\underset{\sim}{x}^*)(\underset{\sim}{x}-\underset{\sim}{x}^*)]\| + \\ &\qquad \|A_m(\underset{\sim}{x})[F'(\underset{\sim}{x})-F'(\underset{\sim}{x}^*)](\underset{\sim}{x}-\underset{\sim}{x}^*)\| + \qquad (8.1) \\ &\qquad \|[A_m(\underset{\sim}{x})F'(\underset{\sim}{x})-A_m(\underset{\sim}{x}^*)F'(\underset{\sim}{x}^*)](\underset{\sim}{x}-\underset{\sim}{x}^*)\| + \lambda^m\|\underset{\sim}{x}-\underset{\sim}{x}^*\|.\end{aligned}$$

Using

$$\begin{aligned}\|A_m(\underset{\sim}{x})\| &= \|(I+H(\underset{\sim}{x})+H(\underset{\sim}{x})^2+\ldots+H(\underset{\sim}{x})^{m-1})B(\underset{\sim}{x})^{-1}\| \\ &\le (1 + \lambda + \lambda^2 + \ldots + \lambda^{m-1})\beta \\ &\le \frac{\beta}{1-\lambda} ,\end{aligned}$$

we can bound the terms of (8.1) individually. From the mean value theorem we obtain

$$\begin{aligned}&\|-A_m(\underset{\sim}{x})[F(\underset{\sim}{x})-F(\underset{\sim}{x}^*)-F'(\underset{\sim}{x}^*)(\underset{\sim}{x}-\underset{\sim}{x}^*)]\| \\ &\qquad\le \|A_m(\underset{\sim}{x})\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| \sup_{0\le t\le 1}\|F'(\underset{\sim}{x}^*+t(\underset{\sim}{x}-\underset{\sim}{x}^*))-F'(\underset{\sim}{x}^*)\| \\ &\qquad\le \|A_m(\underset{\sim}{x})\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| \sup_{0\le t\le 1} L\,\|t(\underset{\sim}{x}-\underset{\sim}{x}^*)\| \\ &\qquad\le \beta L/(1-\lambda)\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\|^2 .\end{aligned}$$

The second term of (8.1) is bounded by

$$\begin{aligned}\|A_m(\underset{\sim}{x})[F'(\underset{\sim}{x})-F'(\underset{\sim}{x}^*)](\underset{\sim}{x}-\underset{\sim}{x}^*)\| &\le \|A_m(\underset{\sim}{x})\|\,\|F'(\underset{\sim}{x})-F'(\underset{\sim}{x}^*)\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| \\ &\le \beta L/(1-\lambda)\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\|^2 .\end{aligned}$$

Lemma 8.1 now yields

$$\begin{aligned}\|[A_m(\underset{\sim}{x})F'(\underset{\sim}{x})-A_m(\underset{\sim}{x}^*)F'(\underset{\sim}{x}^*)](\underset{\sim}{x}-\underset{\sim}{x}^*)\| &= \|(H(\underset{\sim}{x}^*)^m-H(\underset{\sim}{x})^m)(\underset{\sim}{x}-\underset{\sim}{x}^*)\| \\ &\le m\lambda^{m-1}\|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| .\end{aligned}$$

Since $\lambda < 1$, we can choose $c_2 = c_2(\lambda) < +\infty$ so that the set $\{m\lambda^{m-1}\}$ is bounded by c_2. Letting $c_1 = c_1(\lambda, r_7, \beta, L) = 2\beta L/(1-\lambda)$ and combining terms, we obtain

$$\|G_m(\underset{\sim}{x}) - \underset{\sim}{x}^*\| \le c_1\|\underset{\sim}{x}-\underset{\sim}{x}^*\|^2 + c_2\|H(\underset{\sim}{x})-H(\underset{\sim}{x}^*)\|\,\|\underset{\sim}{x}-\underset{\sim}{x}^*\| + \lambda^m\|\underset{\sim}{x}-\underset{\sim}{x}^*\|. \quad \square$$

Assumption (vii) is not the only restriction which could be used to prove Theorem 4.1, but it is one which the Newton-Richardson methods satisfy trivially since $B(\underset{\sim}{x})$ is a constant function. From assumptions (iii) and (vii), it follows that $C(\underset{\sim}{x})$ also satisfies a Lipschitz condition. Thus, since we have already noted that B

is continuous and nonsingular and B^{-1} is uniformly bounded in S_7, it follows that there are an $L_1 < +\infty$ and a $0 < r_8 \le \min(r_5, r_7)$ such that for $\underset{\sim}{x} \in S_8 = \{\underset{\sim}{x}: \|\underset{\sim}{x} - \underset{\sim}{x}^*\| < r_8\}$,

$$\|H(\underset{\sim}{x}) - H(\underset{\sim}{x}^*)\| \le L_1\|\underset{\sim}{x} - \underset{\sim}{x}^*\|.$$

Using this result, we now prove Theorem 4.1.

Proof of Theorem 4.1: Let β, c_1, c_2, and r_7 be defined as in Lemma 8.2 and define

$$c(t) = ((c_1 + c_2L_1)t + \lambda_1)^{1/m}.$$

We select $r < r_8$ so that $c(r) \le c = (\lambda+1)/2 < 1$ and choose $\underset{\sim}{x}^{(0)} \in S$. We can now prove the theorem by induction using Lemma 8.2. For $k = 0$ we have

$$\|\underset{\sim}{x}^{(1)} - \underset{\sim}{x}^*\| = \|G_{m_0}(\underset{\sim}{x}^{(0)}) - \underset{\sim}{x}^*\|$$

$$\le c_1\|\underset{\sim}{x}^{(0)}-\underset{\sim}{x}^*\|^2 + c_2\|H(\underset{\sim}{x}^{(0)})-H(\underset{\sim}{x}^*)\|\|\underset{\sim}{x}^{(0)}-\underset{\sim}{x}^*\| + \lambda^{m_0}\|\underset{\sim}{x}^{(0)}-\underset{\sim}{x}^*\|$$

$$\le ((c_1 + c_2L_1)r + \lambda^{m_0})\|\underset{\sim}{x}^{(0)} - \underset{\sim}{x}^*\|$$

$$\le (c^m - \lambda + \lambda^{m_0})\|\underset{\sim}{x}^{(0)} - \underset{\sim}{x}^*\|.$$

But $0 < \lambda^{m_0} < \lambda < c^m$, so

$$0 < c^m - \lambda + \lambda^{m_0} \le c^m \le c^{m_0},$$

and we have

$$\|\underset{\sim}{x}^{(1)} - \underset{\sim}{x}^*\| \le c^{m_0}\|\underset{\sim}{x}^{(0)} - \underset{\sim}{x}^*\|.$$

If we assume that the theorem holds for $0 \le k < j$, then for

$k = j$ we have

$$\|\underset{\sim}{x}^{(j+1)}-\underset{\sim}{x}^*\| = \|G_{m_j}(\underset{\sim}{x}^{(j)})-\underset{\sim}{x}^*\|$$

$$\le c_1\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|^2 + c_2\|H(\underset{\sim}{x}^{(j)})-H(\underset{\sim}{x}^*)\|\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\| + \lambda^{m_j}\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$\le (c_1 + c_2L_1)\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|^2 + \lambda^{m_j}\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$\le [(c_1 + c_2L_1)(\prod_{\ell=0}^{j-1} c^{m_\ell})\|\underset{\sim}{x}^{(0)}-\underset{\sim}{x}^*\| + \lambda^{m_j}]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$\le [(c_1 + c_2L_1)r\,c^{m_j-m} + \lambda^{m_j}]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$\le [(c^m - \lambda)c^{m_j-m} + \lambda^{m_j}]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$= [c^{m_j} - \lambda c^{m_j-m} + \lambda^{m_j}]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$= c^{m_j}[1 - \lambda c^{-m} + \lambda^{m_j} c^{-m_j}]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|$$

$$= c^{m_j}[1 + \lambda c^{-m}(\lambda^{m_j-1} c^{m-m_j} - 1)]\|\underset{\sim}{x}^{(j)}-\underset{\sim}{x}^*\|.$$

But $0 < \lambda < c^m < 1$, so $\lambda c^{-m} < 1$ and

$$0 < \lambda^{m_j-1} c^{m-m_j} < c^{m(m_j-1)} c^{m-m_j} = c^{(m-1)m_j} < 1.$$

Hence

$$-1 < \lambda c^{-m}(\lambda^{m_j-1} c^{m-m_j} - 1) < 0,$$

and we have

$$\|\underset{\sim}{x}^{(j+1)} - \underset{\sim}{x}^*\| \le c^{m_j}\|\underset{\sim}{x}^{(j)} - \underset{\sim}{x}^*\|.$$ □

CONCLUDING REMARKS

Much work remains to be done on the solution of sparse systems of linear and nonlinear equations. It is, of course, impossible to give a list of subjects for research which is anywhere near exhaustive, but there are three general areas, in particular, which seem to be of prime importance:

(i) the efficient solution of sparse nonsymmetric and symmetric indefinite systems requiring pivoting for reasons of numerical stability;

(ii) the application of general methods of solution to specific problems arising in science and engineering; and

(iii) the development of efficient, well-designed, and portable software for the solution of sparse systems of equations, especially for use on the minicomputers and parallel computers which are rapidly becoming important tools in scientific computation.

These three general areas encompass a diverse range of specific research topics, but all have the common goal of providing the effective computational tools which will be required to solve the numerous critical problems with which scientists and engineers will have to deal in the years ahead.

REFERENCES

[A1] A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[A2] W. F. Ames. *Numerical Methods for Partial Differential Equations*. Barnes and Noble, 1969.

[B1] R. E. Bank and D. J. Rose. An $O(n^2)$ method for solving constant coefficient boundary value problems in two dimensions. *SIAM Journal on Numerical Analysis*, to appear.

[B2] R. Bartels and J. Daniel. A conjugate gradient approach to nonlinear elliptic boundary value problems in irregular regions. Report CNA63, Center for Numerical Analysis, The University of Texas at Austin, 1973.

[B3] G. Birkhoff and J. A. George. Elimination by nested dissection. In Traub [T1], 1973, pp. 221-269.

[B4] E. K. Blum. *Numerical Analysis and Computation: Theory and Practice*. Addison-Wesley, 1972.

[B5] R. K. Brayton, F. G. Gustavson, and R. A. Willoughby. Some results on sparse matrices. *Mathematics of Computation* 24: 937-954, 1970.

[B6] J. R. Bunch. Complexity of sparse elimination. In Traub [T1], 1973, pp. 197-220.

[B7] B. L. Buzbee and F. W. Dorr. The direct solution of the biharmonic equation on rectangular regions and the Poisson equation on irregular regions. *SIAM Journal on Numerical Analysis* 11:753-763, 1974.

[C1] R. Chandra, S. C. Eisenstat, and M. H. Schultz. Conjugate gradient methods for partial differential equations. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975, pp. 60-64.

[C2] A. Chang. Application of sparse matrix methods in electric power system analysis. In Willoughby [W1], 1968, pp. 113-122.

[C3] P. Concus. Numerical solution of the minimal surface equation. *Mathematics of Computation* 21:340-350, 1967.

[C4] P. Concus. Numerical solution of the minimal surface equation by block nonlinear successive overrelaxation. Proceedings of the IFIP Conference, 1968, pp. 153-158.

[C5] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the AMS* 49:1-23, 1943.

[C6] E. Cuthill. Several strategies for reducing the bandwidth of matrices. In Rose and Willoughby [R10], 1972, pp. 157-166.

[D1] G. Dahlquist and A. Björck. *Numerical Methods*. Prentice-Hall, 1974.

[D2] M. Diamond. *An Economical Algorithm for the Solution of Finite Difference Equations*. PhD dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, 1971.

[D3] E. G. D'Jakonov. On certain iterative methods for solving nonlinear differential equations. *Proceedings of the Conference on the Numerical Solution of Differential Equations (Scotland, 1969)*. Springer-Verlag, 1969, pp. 7-22.

[D4] F. W. Dorr. The direct solution of the discrete Poisson equation in $O(n^2)$ operations. *SIAM Review* 17:412-415, 1975.

[D5] F. W. Dorr. The direct solution of the discrete Poisson equation on a rectangle. *SIAM Review* 12:248-263, 1970.

[D6] I. S. Duff, A. M. Erisman, and J. K. Reid. On George's nested dissection method. AERE Report, Harwell, England, 1975.

[D7] I. S. Duff and J. K. Reid. A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination. *Journal of the Institute of Mathematics and its Applications* 14:281-291, 1974.

[D8] T. Dupont. A factorization procedure for the solution of elliptic difference equations. *SIAM Journal on Numerical Analysis* 5:753-782, 1968.

[E1] S. C. Eisenstat. Complexity bounds for Gaussian elimination. To appear.

[E2] S. C. Eisenstat. Private communication.

[E3] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Application of sparse matrix methods to partial differential equations. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975, pp. 40-45.

[E4] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. The application of sparse matrix methods to the numerical solution of nonlinear elliptic partial differential equations. In A. Dold and B. Eckmann, editors, *Proceedings of the Symposium on Constructive and Computational Methods for Differential Equations*. Springer-Verlag, 1974, pp. 131-153.

[E5] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Direct methods for the solution of large sparse systems of linear equations with limited core storage. Presented at the SIAM 1974 Fall Meeting, Alexandria, Virginia, 1974.

[E6] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Efficient implementation of sparse symmetric Gaussian elimination. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975, pp. 33-39.

[E7] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. A user's guide to the Yale Sparse Matrix Package. To appear.

[E8] S. C. Eisenstat and A. H. Sherman. Subroutines for envelope solution of sparse linear systems. Report 35, Department of Computer Science, Yale University, 1974.

[F1] G. E. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, 1967.

[F2] G. E. Forsythe and W. R. Wasow. *Finite Difference Methods for Partial Differential Equations*. John Wiley and Sons, 1960.

[G1] J. A. George. *Computer Implementation of the Finite Element Method*. PhD dissertation, Department of Computer Science, Stanford University, 1971.

[G2] J. A. George. An efficient band-oriented scheme for solving n by n grid problems. Proceedings of the Fall Joint Computer Conference, 1972, pp. 1317-1320.

[G3] J. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis* 10:345-363, 1973.

[G4] J. A. George. Private communication.

[G5] N. E. Gibbs, W. G. Poole, Jr., and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, to appear.

[G6] D. Greenspan and S. V. Parter. Mildly nonlinear elliptic partial differential equations and their numerical solution, II. *Numerische Mathematik* 7:129-146, 1965.

[G7] F. G. Gustavson. Some basic techniques for solving sparse systems of linear equations. In Rose and Willoughby [R10], 1972, pp. 41-52.

[H1] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

[H2] A. C. Hindmarsh. Solution of banded linear systems. Report UCID-30045, Lawrence Livermore Laboratory, Livermore, California, 1972.

[H3] A. J. Hoffman, M. S. Martin, and D. J. Rose. Complexity bounds for regular finite difference and finite element grids. *SIAM Journal on Numerical Analysis* 10:364-369, 1973.

[I1] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. John Wiley and Sons, 1966.

[J1] A. Jennings. A compact storage scheme for the solution of symmetric linear simultaneous equations. *Computer Journal* 9:281-285, 1966.

[J2] A. Jennings and A. D. Tuff. A direct method for the solution of large sparse symmetric simultaneous equations. In Reid [R1], 1971, pp. 97-104.

[K1] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, 1972, pp. 85-103.

[K2] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[L1] W. H. Liu and A. H. Sherman. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, to appear.

[M1] R. S. Martin and J. H. Wilkinson. Symmetric decomposition of positive definite band matrices. *Numerische Mathematik* 7:355-361, 1965.

[O1] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[P1] S. V. Parter. The use of linear graphs in Gauss elimination. *SIAM Review* 3:119-130, 1961.

[P2] H. S. Price and K. H. Coats. Direct methods in reservoir simulation. Paper SPE 4278, Society of Petroleum Engineers of the AIME, 1973.

[R1] J. K. Reid, editor. *Large Sparse Sets of Linear Equations*. Academic Press, 1971.

[R2] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society*, Series A(210), London, pp. 307-357, 1910.

[R3] R. L. Rivest and J. Vuillemin. A generalization and proof of the Aanderaa-Rosenberg conjecture. Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing, 1975, pp. 6-11.

[R4] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. Read, editor, *Graph Theory and Computing*. Academic Press, 1972, pp. 183-217.

[R5] D. J. Rose. *Symmetric Elimination on Sparse Positive Definite Systems and the Potential Flow Network Problem*. PhD dissertation, Division of Engineering and Applied Physics, Harvard University, 1970.

[R6] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on graphs. Submitted to *SIAM Journal on Computing*.

[R7] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination. Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing, 1975, pp. 245-254.

[R8] D. J. Rose and G. F. Whitten. Automatic nested dissection. Proceedings of the ACM National Conference, 1974, pp. 82-88.