# ANALYSIS OF SPARSE SYSTEMS

Iain Spencer Duff

New College

A thesis submitted for the degree of

Doctor of Philosophy

in the University of Oxford

October 1972

# PREFACE.

This thesis is the result of two years work at the Oxford University Computing Laboratory carried out between October 1970 and September 1971.

The thesis contains several theoretical and experimental results on the analysis of sparse matrices and discusses the solution of linear equations, the least squares problem, and the eigenvalue problem where the matrices concerned are large and sparse. The computed results for the first six chapters were obtained using a KDF9 computer and subsequently a 1906A at Oxford, while the results in chapters seven and eight were obtained using an IBM 360/165 at AERE, Harwell. I am indebted to the Atomic Energy Authority for this use of their computing facilities.

I would like to thank the three supervisors whom I have had during my research years at Oxford. I am grateful to Professor Tewarson for fanning the enthusiasm kindled in me by Professor Fox who supervised my diploma research and encouraged me to go to a sparse matrix conference at Oxford in Easter 1970 which introduced me to the topic of this thesis. I am most appreciative of the subsequent guidance by my current supervisor, Dr. Reid, and his help and encouragement during the long months of writing this thesis.

I would also like to thank the Carnegie Trust for the Universities of Scotland for providing financial assistance, my mother, for the many long hours spent typing this thesis, and to Diana, my fiancée, for keeping me on a regular diet during the process of thesis production.

Iain Duff.

14th October, 1972.

ABSTRACT.

The aim of this thesis is to conduct a general investigation in the field of sparse matrices, to investigate and compare various techniques for handling sparse systems suggested in the literature, to develop some new techniques, and to discuss the feasibility of using sparsity techniques in the solution of overdetermined equations and the eigenvalue problem.

Chapter one is mainly concerned with an introduction to the subject of this thesis. It defines some of the terms which occur frequently in subsequent chapters and introduces the notion of 'sparsity' explaining some of the characteristics which distinguish a sparse matrix from a full one. Since combinatorial techniques and principally the theory of graphs are used throughout this thesis some graph theory terms are also defined. Some discussion of the aspects in the sparse matrix field not discussed by this thesis is also given in this chapter.

The next five chapters deal with direct methods of solving systems of large sparse linear equations. Chapters two to four deal with techniques for partitioning systems so that the introduction of non-zeros to the matrix is confined to particular regions of it, while chapter five looks at controlling growth of non-zeros within the whole system and chapter six investigates properties of random matrices.

Chapter two discusses the importance of ordering a matrix so that there are non-zeros on its diagonal and considers techniques of obtaining such an ordering for a general matrix. A method is suggested, discussed theoretically, and shown to be considerably faster than current techniques on a wide range of matrices. In this chapter, the equivalence of bireducibility and reducibility is shown for a very wide class of matrices.

Chapter three surveys several of the techniques available for ordering a matrix so that it is in block triangular form. A graph theoretic framework is established in which these current techniques are examined and from which a new method of block triangularisation is obtained. This method is found to be very competitive for a certain range of sparse matrices and computed results indicate the relative empirical merits of some of the other techniques from which conclusions of the most appropriate technique can be drawn.

In chapter four, the discussion is concerned with matrices which can be ordered so that they are nearly block triangular with only a few non-zeros outside the block triangular pattern. The equivalence of tearing and the method of modification is discussed and an algorithm for solving such sets of equations, which uses the structure of the decomposition to great advantage in maintaining sparsity, is also described. The advantages of using tearing techniques on systems of equations arising from Laplace's equation are examined in some detail and it is seen that while savings of the order of the number of mesh points in one direction can be made in the two dimensional case, extensions to the third dimension do not indicate that direct methods form a viable alternative over iterative schemes.

Chapter five carries a fairly extensive survey of techniques for storing sparse matrices and discusses techniques for selecting a pivotal sequence in Gaussian elimination such that the number of non-zeros introduced in the elimination process is kept low. Many numerical tests are done on both a priori and local orderings and it is seen that even the best a priori orderings do not achieve the advantages of a local method. Some examples are given to show that methods which produce local minimum fill-in do not necessarily lead to global minimum fill-in, and techniques are examined for reducing the time required for performing such an ordering. Comparisons of times and fill-ins for some local criteria are given,

and it is seen that the more it is required to reduce the fill-in, the more time the algorithm will take.

Chapter six indicates the drastic fill-in which will occur in random matrices if no sparsity pivoting is employed. A theoretical study of the fill-in to random systems when Gaussian elimination is performed on them is conducted using graph theory and formulae relating to this fill-in are found. The predictions of this theory are seen to compare extremely favourably with computed results from a series of random matrices.

Chapter seven discusses the least squares solution of sparse overdetermined systems of equations. Several currently used techniques are discussed as regards the way in which they retain the sparsity of the initial system during the solution process. It is seen that the use of orthogonalisation techniques even with good pivoting strategies is unlikely to be very competitive. Although it is seen by the production of many test examples, that the structure of the system can be vital, it is observed from a series of experiments that the method of Wilkinson and Peters (1970) would be the best one for a general sparsity-oriented routine.

In the final chapter, the eigenvalue problem for sparse matrices is discussed. Although it is proved that, for the same pivotal sequence, Householder's method of reduction can never give less fill-in than Givens, neither method is seen to be of any good for preserving the original sparsity of the system, even if sparsity pivoting techniques are utilised. While for unsymmetric systems, the method of Gaussian reduction is seen to be just feasible in a sparsity context, the situation for symmetric systems is much less pleasant. Some discussion of techniques for overcoming this problem is given and an algorithm for modifying the triangular factorisation is also stated. In this chapter there is also a detailed description of a symmetric sparse matrix storage scheme.

ERRATA

| | |
|---|---|
| page 42 | Figure 4.3. The value of the element of the matrix in position (4,9) should be 8 and not 7. |
| page 90 | Line 2 of second paragraph. Change "unknown" to "known". |
| page 98 | Line 5 after equation (4.22). Change "5" to "20". |
| page 155 | Equation at foot of page. Change ${}^{n}C_{k-1}$ to ${}^{n-1}C_{k-1}$. |
| page 156 | Equation after line 1. Change ${}^{n}C_{k-1}$ to ${}^{n-1}C_{k-1}$. |
| page 225 | Line 3 after 4.1 until end of paragraph. Delete "while the eigenvectors ... diagonal block". |

CONTENTS

# CHAPTER ONE.

## INTRODUCTORY REMARKS.

## Section 1. Introduction and some comments on thesis organisation.

This chapter serves as a general introduction to this thesis. In it, some terms used commonly in this thesis are defined and it is advisable for the reader to be familiar with its contents before embarking on any reading of the later chapters. In section 2 of this chapter, some comments are made concerning sparse matrices and ways in which they differ from full matrices. At this juncture, it should be remarked that a matrix is considered 'sparse' whenever the number of non-zeros in the matrix is substantially less than the total number of its entries. A matrix which is not 'sparse' is called 'full'. Because of the prominence of graph theoretic techniques in the development of many of the arguments in this thesis, section 3 of this chapter has been devoted to establishing certain definitions used frequently in the subsequent text. The field of sparse matrix analysis is a very broad one and so, in spite of the length of this thesis, for which apologies are made, it has been impossible to examine every facet of it in three years research. Therefore, part of section 4 has been devoted to explaining some of the major aspects not covered by this thesis, the rest of the section being concerned with possible future developments in the field.

The basic problems dealt with in this thesis are the solution of the linear equations

$$Ax = b \qquad \ldots\ (1.1)$$

and the eigenvalue problem

$$Ax = \lambda x \qquad \ldots\ (1.2)$$

where A is a sparse matrix. For square non-singular A in equation (1.1) the examination is restricted to direct methods of solution based on Gaussian elimination. Clearly, once a suitable decomposition of the form

$$PAQ = LU,$$

has been found, where P and Q are permutation matrices and L and U are lower and upper triangular ones respectively, the solution to equation (1.1) with A replaced by $A^T$ can be obtained without any further decomposition, since

$$Q^T A^T P^T = L_1 U_1$$

where

$$L_1 = U^T \quad , \quad U_1 = L^T$$

Thus, throughout this thesis, although the discussion is centred on the matrix A, it should be remembered that this dual problem has also been effectively solved. When A is $m \times n$ $(m > n)$ in equation (1.1), the investigation is restricted to the least squares problem where the solution to equation (1.1) is found in the $l_2$ norm. The eigenvalue problem (1.2) is discussed for both symmetric and unsymmetric square matrices A.

The author's motivation for doing this research has already been mentioned in the preface to this thesis but it is not difficult to see the importance of this topic in many fields of pure and applied science. Tewarson (1970c) and Duff (1970) give impressive lists of subjects where sparse matrix analysis has played a large and vital part, and Sato and Tinney (1963) and subsequently many others have pointed out the great gains to be made by using sparsity techniques. There have been three major conferences in this field in recent years. A conference was held at Oxford in Easter 1970 (Reid (1971a)) and two autumn conferences were held at the IBM Research Center at Yorktown Heights in 1968 and 1971 (Willoughby (1969) and Rose and Willoughby (1972)). The fact that these three conferences were held and the variety of papers and contributions from various fields given at them testifies to the importance of sparse matrix work. Many sparse matrix references can be found in the proceedings of these three conferences and a substantial number of other references are given by Brayton et al (1970), Duff (1970), Tewarson (1970c), and

Willoughby (1970). In addition, the January 1971 edition of the IEEE publication on circuit theory was devoted to computer aided design and carries many papers and references on the use of sparse matrix techniques in that particular field.

A summary of the main results in this thesis will be found in the abstract. The introductory section to each chapter expands on this initial summary. The machines used in producing experimental results were an English Electric KDF9 and an ICL 1906A at Oxford and an IBM 370/165 at Harwell. When the research for this thesis was being done, the KDF9 was passing out of existence, and it eventually expired in December 1971. The only results used here which were generated by this machine are some of those in chapter two, the bulk of the computing being done on the 1906A. Since the 1906A works in a multi-programming environment it is difficult to get an accurate and meaningful estimate of the time taken for the procedures tested and the times given as 1906A milltime are, in fact, times based on the number of operations which the procedure performs. Since the method of charging for the use of the 1906A is based on a similar estimate, it seems a reasonable basis on which to obtain times to compare procedures. The programming language used on the 1906A was ALGOL largely because the author is more accustomed to using it and because of the superior debugging facilities present in the compiler for this language. It should be remarked, however, that an identical program in FORTRAN on the 1906A is likely to take up to half as much time as its ALGOL counterpart and so would be obviously preferred if the procedures were to be used in a production context. Because of the IBM bias towards FORTRAN, this language was used on the 370/165 for the results given in chapters seven and eight. This machine is considerably faster than the 1906A but no comparisons have been made between runs on the different machines nor, indeed, on the IBM machine itself. Several programs used to produce the results are given in the appendices. Of all the programs used in the production of the

results in this thesis only a few have been selected for inclusion in these appendices. The basis on which this selection was made has been to only choose programs which either illustrate some important facet of sparse matrix analysis or which the author believes might be useful to future entrepreneurs in the field.

This thesis is divided into eight chapters, an introductory chapter and seven other chapters which are basically self-supporting units each containing an introduction including a survey of the chapter and a summary of the main results contained in it. In general, the experimental results for each section are to be found in tables or graphs at the end of the section and equations and figures are numbered by sections. Thus, figure 2.3 refers to the third figure of section 2. Cross-references of figures and equations always refer to the appropriate section of the chapter in which the cross-reference is made unless an explicit chapter reference is given with it. The numbering of tables and graphs is conducted similarly.

## Section 2. Sparse matrices.

It is certainly true that sparse matrices are not a mystical new beast but are rather a mutant form of an old beast. They are subject to the normal laws and conventions of ordinary matrix analysis, and, throughout this thesis, the accepted conventions in matrix terminology and notation have been adhered to. Thus, capital letters are used to refer to matrices, the element (or entry) in the i th row and j th column of the matrix A is denoted by $a_{ij}$, and the i th row and j th column of A are written as $A_{i\bullet}$ and $A_{\bullet j}$ respectively. The only deviation from normal convention is that the term elements is often used to refer only to the non-zero elements of the matrix although the meaning in individual cases should be abundantly clear from the context.

Nothing is the most important feature of the sparse mutant. That is to say, great advantage can be taken of the fact that most of the elements in a sparse matrix are zero. Since operations with zero elements can be avoided altogether, it is possible to make great savings in operation counts as well as storage by the suitable organisation of sparse matrix procedures.

It is often a question of debate as to what is meant by 'operation count'. In most sparse matrix procedures a useful measure of 'operation count' is the total number of divisions and multiplications in the procedure. In the 1906A, a floating-point division takes 5.97 microseconds while floating-point multiplications and additions (subtractions) take 2.04 and 0.95 microseconds respectively. Hence, since the number of divisions in most matrix processes is low and a multiplication is very often accompanied by an addition, the total number of multiplications and divisions is likely to give a good estimate of the computational cost of a matrix process. It is this measure which is used throughout this thesis. However, it should be borne in mind that the fixed-point overheads

(for example, accessing an array element or going through a loop), which are present in most sparse matrix codes, represent a very significant proportion of the total work done. The above-mentioned operation count can be used for comparisons of procedures since the fixed-point overheads are likely to be roughly proportional to the number of operations, but it is not suitable as an absolute measure of the work involved in the process. It is by avoiding such fixed-point operations that Gustavson et al (1970) can make gains of a factor of about 3 over sparse matrix codes which include such operations.

In full systems, the relevant factor for storage and operation counts is $n$, the order of the matrix. Thus, it requires $O(n^3)$ operations for the LU decomposition of a matrix, or the multiplication of two matrices, $O(n^2)$ for the solution of subsequent right hand sides and for the multiplication of a vector by matrix, and $n^2$ units of core to store the matrix in a machine. For sparse systems, the relative factors are the number of non-zeros in the matrix, and the number of non-zeros in a row (say, row i) or a column (say, column j) of the matrix. Throughout this thesis, these will be denoted by $\tau$, $r_i$, and $c_j$ respectively. Thus, for example, the first stage of LU decomposition (L unit triangular) will require $r_1(c_1-1) + 1$ operations in the sparse case as opposed to $n^2-n+1$ in the full case. Since $r_i$ and $c_j$ are typically less than one tenth of $n$, this initial stage for the sparse matrix involves only about 1% of the work for the full case. It is very important in sparse matrix analysis to derive the maximum benefit from this type of gain and to ensure that the advantage of having many elements zero is kept as much as possible during the elimination process. Thus, it would be very desirable if a pivotal sequence was obtained which ensured that the sparsity of the initial matrix was kept throughout the process and was reflected in the form of the LU decomposition. The standard example of this is shown in figure 2.1, where zero elements are represented by blanks and non-zeros by x, as is the general convention in this thesis.

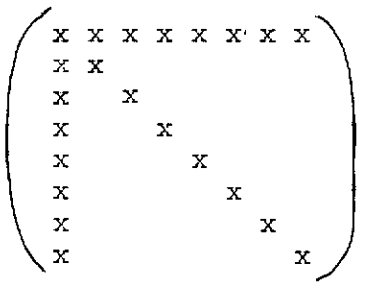


figure 2.1

If the (1,1) element is chosen as pivot at the first stage of the elimination, then clearly the fill-in will be total and so all advantageous properties of the great sparsity of the initial matrix are totally lost. If, however, the decomposition is performed with pivots selected from the diagonal with the (1,1) element chosen last, then the LU decomposition will contain no more non-zeros than the initial matrix and will take $O(n)$ instead of $O(n^3)$ operations to perform. This type of pivoting is called sparsity pivoting and is considered in more detail in chapter five. While it is certainly true that the pivotal sequence chosen by sparsity pivoting is very unlikely to be the same as that for pivoting based on controlling numerical stability (Wilkinson (1965)), it is most certainly not always the case that such a strategy is unstable. In fact, it is sometimes the case that, since rounding errors can only occur when arithmetic operations are performed, sparsity pivoting gives better results than total pivoting. This has been observed experimentally when elimination has been performed on a matrix of the form of figure 2.1 where the element (1,1) is the largest element. In practice, however, it is desirable to keep some check or control on the stability of the elimination process since there are no satisfactory a priori upper bounds to the errors induced if sparsity pivoting is used. This can take the form of a control over the growth in size of the non-zero elements in successive reduced matrices (Reid (1971c)). Even in the full case, it is desirable, when solving a set of equations, not to calculate and store the explicit inverse of the coefficient matrix since significantly more work has to be done to obtain it. However,

in the sparse case, there are even more reasons for storing a decomposition of the matrix as opposed to its inverse. This is because the inverse of a sparse matrix is usually very full while its LU decomposition is often not much denser than the original matrix. In fact, it is customary to store L and U in factored form where

$$L^{-1} = L_n \ldots L_1 \qquad \ldots\ (2.1)$$

and

$$U^{-1} = U_2^{-1} \ldots U_n^{-1} \qquad \ldots\ (2.2)$$

with

$$PAQ = LU \qquad \ldots\ (2.3)$$

(P and Q permutation matrices), in order to reap the full benefit of the original sparsity of A. (Tewarson (1966), (1967a)).

Although much stress has been laid on the importance of zeros in a sparse matrix, it is their quality as well as their quantity which is of interest to the sparse matrix analyst. That is to say, it is not only the number of non-zeros but their position which is important. This fact is particularly significant in chapters 2, 3 and 4 of this thesis where a main concern is over the bireducibility of a matrix. A matrix A is bireducible if there exist permutation matrices P and Q such PAQ has the form of figure 2.2.

$$\begin{pmatrix} A_1 & A_2 \\ 0 & A_3 \end{pmatrix}$$

figure 2.2

If $Q = P^T$, then A is said to be reducible. It would seem to be obvious that a matrix is more likely to be reducible the higher percentage of zero elements it has. Certainly such a correlation exists but the property of reducibility is to a large extent dependent on the position of the non-zeros as well as their number. This is brought out most forceably in the examples of the following figures.

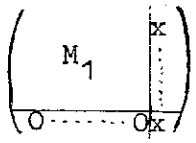



figure 2.3(a)

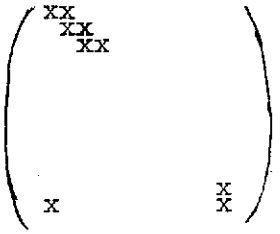



figure 2.3(b)

In figure 2.3(a) $M_1$ is a matrix of order n-1 all of whose elements are non-zero, the last column of the matrix in the figure has all non-zero entries while the last row has zeros in the first n-1 positions. Clearly, the matrix of figure 2.3(a) with only n-1 zeros is reducible while the matrix in figure 2.3(b) is not even bireducible even although it has $n^2-2n$ zero elements! Undoubtedly, however, the matrix of figure 2.3(b) is of more interest in a sparsity context.

Because of this importance of structure it is quite often of interest to examine in some detail the sparsity structure of the matrix as opposed to the actual values of the non-zeros concerned. This examination is performed on the Boolean representation of the original matrix. The Boolean representation of a matrix A is a matrix whose entries are either 0 or 1, the element $b_{ij}$ being 1 if and only if the corresponding element, $a_{ij}$, of A is non-zero. The term binary is also often used for such a matrix. It is, of course, true that, when matrix algorithms are performed using Boolean arithmetic (1+1 = 1) on Boolean representations, the effects of numerical cancellation and instability in the process are ignored (see, for example, chapter seven). The first defect is, however, not very worrying since most numerical cancellation is dependent on the initial value of the non-zeros which may well vary even although the sparsity structure remains fixed. This means that most sparse matrix codes for use on general matrices cannot take advantage of such cancellations anyway and so they can be happily ignored in any theoretical investigations. The second defect is dependent on the purpose for which the Boolean form is being used. The use of it in chapters 2 and 3

causes no worries on this score, while its use in the theory of chapter five should not bias the results greatly as can be observed from the work of Curtis and Reid (1971d). Willoughby (1971c) avoids this difficulty of cancellations by considering M matrices (Fiedler and Pták (1962)). In this short discussion of Boolean representation, the following two definitions can be made. The Boolean inverse of a matrix A, defined for matrices whose diagonal elements are non-zero, is a (0,1) matrix which has its j th column equal to the result of solving

$$Ux = L^{-1}e_j \qquad \ldots\ (2.4)$$

where L and U are the matrices of equations (2.1) to (2.4), the decomposition has been done on the Boolean representation of A, and equation (2.4) is solved using Boolean arithmetic with $e_j$ being the j th column of the identity matrix of appropriate order. A theorem on Boolean inverses is proved in the next section of this chapter. A second definition is that of structural non-singularity. A general matrix is structurally non-singular if there exists a set of real numbers which may be assigned to the possible non-zeros of the matrix to make it non-singular. Thus, non-singular matrices are structurally non-singular but the converse does not hold, an example being given in figure 2.1 of chapter two. In fact, a matrix is structurally non-singular if and only if the permanent of its Boolean representation is non-zero (Marcus and Minc (1965)).

This discussion of Boolean representations of structured matrices highlights one of the major deficiences in this field. This is the lack of non-artificial examples of structured sparse matrices. Throughout this thesis, use has been made of three structured matrices whose sparsity pattern has been described in the literature. One is a 54 x 54 matrix obtained by A.R. Curtis (UKAEA, Harwell) in the solution of stiff differential equations arising from calculations on biochemical reactions and is illustrated in Curtis and Reid (1971c). A 57 x 57 matrix is also used and was again obtained from stiff

equations this time arising in circuit theory calculations. An illustration of this matrix can be found in Willoughby (1971b). The third structured matrix is of order 199 and was obtained from stress analysis calculations. It is illustrated in Willoughby (1971b). Matrices identical to these three are used in chapters 2 to 5. Chapter eight uses identical matrices to the first two, but the 199 case is altered by having non-zeros added so that all its diagonal elements are non-zero. In chapter seven, this is again done to the 199 case and, in addition, an extra row with a non-zero in the first column is added to all three matrices. Throughout this thesis these three matrices and their variants are called CURTIS 1, WILLOUGHBY 1 and WILLOUGHBY 2 respectively.

Random matrices have also been used throughout this thesis. These matrices have been used because of the ease with which they can be generated and the lack of readily available structured examples. They have non-zeros on the diagonal and each of their off-diagonal elements has probability p of being non-zero. In the actual implementation, a congruential pseudo-random generator was used to generate the indices of the non-zeros. This generator had a cycle of 2796203 on the 1906A and $2^{30}$ on the 370/165. The procedure which was used to generate random matrices on the 1906A is given in the appendices. Although it is certainly true that particular algorithms operating on structures for which the algorithm was specifically designed will not have this performance reflected when tested on random matrices, it is probably true to say that if an algorithm performs comparatively well on a selection of random matrices, it will also do well on a wide variety of structured ones.

## Section 3. Some notes on graph theory.

In most of the chapters of this thesis, graph theory plays an important role. It is used in order to aid the visualisation of procedures and to simplify some of the proofs. It is true that the world divides into two groups, the matrix men and the graph men and and the author is one of the latter, but in many instances the interconnection between networks, graphs, and matrices is so evident that it seems foolish not to use this obvious alternative description of some of the processes being explained. Certainly, it is difficult to envisage how the theorems in chapter six could be proved without the use of graph theory.

There have been numerous publications on graph theory but among the more general reference works have been books by König (1950), Berge (1962), Ore (1963), Harary (1969), Busacker and Saaty (1965), Dulmage and Mendelsohn (1959), and Harary, Norman and Cartwright (1965). The first five references deal with the theory of general graphs while the last two authors specialise in bigraphs (see chapter two) and digraphs respectively. Many people have indicated the strong links between graph theory and sparse matrix analysis among whom can be numbered Carré (1966), Dulmage and Mendelsohn (1962,1967), Harary (1959b, 1962a, 1962b, 1971), Mason (1953), Parter (1960, 1961), Ponstein (1966), Read (1971), Rose (1970, 1971) and Tewarson (1967a). Definitions of most of the terms used in this thesis can be found in the above-mentioned literature although, for the benefit of the reader, some of the more commonly used terms are defined here.

A (finite) directed graph (digraph) is composed of a (finite) set, X, of points and a set, E, of directed lines whose members are ordered pairs of points. E is called a line set. Thus a typical member of set E might be $(x,y)$, where $x$ and $y \in X$. The directed line $(x,y)$ then joins or links the point $x$ to the point $y$ in the digraph and the point $x$ is said to be joined or linked to the point $y$. The digraph described above would be denoted by $G(X,E)$. A directed path

(dipath) is said to exist from the point a to the point b, if there exists a sequence of directed lines from a to b. That is, if there exists a set of ordered pairs of the form

$$(a,x_1),\ (x_1,x_2),\ (x_2,x_3) \quad \ldots \quad (x_{n-1},x_n),\ (x_n,b)$$

in the set E. If such a dipath exists from point a to point b, then a is connected to b in the directed graph and b can be reached from a in the digraph. If the points in the digraph are in one-to-one correspondence with a subset of the positive integers (generally the subset is (1, .. n) where n is the number of points in the digraph), then the digraph is said to be labelled. A subgraph of a directed graph G(X,E) is itself a directed graph which has as its point set a subset of X and has as its line set those lines in E for which both elements of the ordered pair lie in X. A complete subgraph or digraph (also called a clique) is a digraph where the line set consists of all possible pairs of points in the point set. A subgraph of a digraph is a strong component if every point of the subgraph can be reached from every other point of the subgraph. A directed cycle in a digraph is a dipath beginning and ending at the same point; it is simple if no points (apart from the first and the last) are repeated in the dipath. A graph is connected if for any two points a and b, a can be reached from b or b from a. A connected acyclic graph is called a tree. A point has indegree (outdegree) equal to the number of ordered pairs in E for which it is the second (first) point of the ordered pair. A cutpoint is a point whose removal will result in a connected graph becoming disconnected. A cutpoint set is a set of points whose removal will leave the remaining subgraph disconnected.

Most of the above definitions also apply to a undirected graph (also called a graph), the major difference being that the pairs in the line set are unordered, (a,b) being the same line as (b,a). If points a and b are linked in a graph, they are called adjacent.

Such a graph is triangulated if for every simple cycle containing more than 3 points there exists a line joining any two non-adjacent points of the cycle. A triangulation of a graph is performed by adding lines to the original graph so that the new graph formed is trangulated. Definitions similar to the above but applicable to bigraphs are given in section two of chapter two. In much of the literature the term node or vertex is used to denote a point while the term edge is used instead of line.

Matrices are generally associated with graphs in the following manner. The adjacency matrix of a graph is a matrix whose elements are either 0 or 1 and whose (i,j) th entry is 1 if and only if there is an ordered pair (i,j) in the line set of the graph. Similarly, a graph can be associated with a matrix by letting line (i,j) exist if and only if the (i,j) th entry of the matrix is non-zero. In an obvious manner, undirected graphs can be associated with symmetric matrices. The reachability matrix, R, of a digraph is Boolean and has $r_{ij} = 1$ if and only if the point i is connected to the point j by a dipath in the digraph.

Many graph theory algorithms have been developed (see, particularly, the algorithms section of Communications of the ACM) which find, for example, the longest paths, the strong components, or the cycles in graphs and some of these can be put to use in sparse matrix algorithms particularly the ones in chapters 2 to 4 on assignment sets and partitioning and tearing. Some principal exponents in this graph analysis field have been Warshall (1962), Thorelli (1966), Tiernan (1970), Purdom (1970), and Paton (1971). Indeed, it is sometimes the case that such a graph analysis procedure is an integral part of a sparse matrix scheme. For example, the work of Bree (1964) on linear equations or Barkley and Motard (1972) on tearing requires the use of an algorithm to find cutpoint sets (Paton (1971)).

Most of the graph theorems used in this thesis are either obvious or have appeared in the literature mentioned earlier although several

proofs are given explicitly in chapter three. The following theorem is now proved here since the author has not been able to find a correct proof of it in the literature. It uses the Boolean inverse of a matrix which is defined in the previous section of this chapter.

Theorem 3.1

The (i,j) th entry (denoted $a^I_{ij}$) of the Boolean inverse of a matrix, A, is 1 if and only if the point i of the directed graph associated with the matrix is connected to the point j.

Proof.

As mentioned in section 2, the j th column of the Boolean inverse of A is obtained by finding the solution to the set of equations

$$L(Ux) = e_j \qquad \ldots\ldots (3.1)$$

where $e_j$ is the j th column of the identity matrix and L and U are the factors in the Boolean LU decomposition of A.

Now, the i th entry of x (which is $a^I_{ij}$) in equation (3.1) is 1 if and only if there exists an l ($1 \leq l \leq n$) such that there is a dipath in the digraph of L from l to j and a dipath in the digraph of U from i to l.

Since any path which exists in the digraph of A also exists in the digraph of the matrix C where

$$c_{ij} = \begin{cases} l_{ij} & i \geq j \\ u_{ij} & i < j \end{cases}$$

because no cancellation occurs in Boolean decomposition. Hence the sufficiency part of the theorem has been proved.

Furthermore, if any two points in the digraph of L or U are jointed by a directed line, then they were connected by a dipath in the digraph of A. Therefore, l is connected to j and i is connected to l by dipaths in the digraph of A and the necessity part of the theorem has been proved.

## Section 4. Possibilities for future work.

This thesis is similar to its topic inasmuch as it is open ended. Although several aspects of sparse matrix analysis are examined here, it by no means covers all facets of the subject nor is it claimed to be a closed book on the field. This section indicates some of the major parts of the field which are not discussed at any length in the thesis and indicates which paths are likely to be worth exploring in the future.

Throughout this thesis the only method of solution of linear equations which is discussed in depth is that of Gaussian elimination with pivoting. In this solution process, the matrix is reduced to upper triangular form by means of successive premultiplications by n elementary lower triangular matrices which are stored as a factored form of the $L^{-1}$ in the LU decomposition of A. Part of the reason for this is that it has been shown by the work of Brayton et al (1970) and others that this elimination form of the inverse is a method of solving equations which does, in general, take best advantage of the sparsity of the original matrix. For this reason neither the product form of the inverse (Tewarson (1966)) nor orthogonalisation methods (for example, Tewarson (1968), have been discussed in detail, and the results of chapters seven and eight would seem to indicate that the latter methods are especially bad. Certainly row Gauss elimination (Gustavson (1972) and suggestions in chapter seven) and Crout reduction (Gustavson et al (1970)) are possible variants of Gaussian elimination which could be used successfully, but the sparsity analysis for these methods is similar to that described in this thesis for the straightforward method. In addition, no mention has been made in this thesis of the variability typing of elements where different attributes are allowed to a non-zero depending on its dependence on different independent variables. This is at present a very fruitful source of research in integration and the solution of non-linear equations, Hatchel (Hatchel et al (1971), Hatchel (1972)) being one of the principal pioneers in this field (see also, Lo Data (1970)).

Iterative techniques (Varga (1962)) have scarcely been mentioned here, nor has the finite iteration method of conjugate gradients (Reid (1971b)). Evans (1972) has done several experiments in the use of iterative techniques for sparse systems and the discussion on the example of the three dimensional Laplace's equation in chapter four would seem to indicate that further research in this field might be fruitful. Iterative methods of finding the inverse (Hershkowitz and Noble (1965)) or of solving sets of equations (Khabaza (1963) who uses a truncated form of conjugate gradients) using matrix multiplications have not been discussed, and it is not believed that they offer a viable alternative method of solving sets of sparse equations.

Perhaps one of the most open of open questions in numerical linear algebra at the moment is that of scaling. It is assumed that, before the pivot selections for any of the algorithms in this thesis are made, the matrices under consideration have already been well-scaled. The meaning of well-scaled is also a question for debate, but it is generally assumed that no element of the matrix is dominant over the rest in size and that all matrix elements have a weight appropriate to their influence in the problem under consideration. Curtis and Reid (1971a) have conducted a number of experiments on scaling by taking well-scaled matrices, deliberately making them badly scaled, and then attempting to recover the original well-scaled matrix by using scaling techniques. They have found that a least squares minimisation of the sum of the squares of the logarithms (Hamming (1971)) gives better results than minimising the maximum of the moduli of the logorithms in a minimax sense (Fulkerson and Wolfe (1962)) or than using successive applications of row and column equilibration. One of the reasons for the superiority of the least squares method over the minimax method is that the minimax method will lay too much stress on very small elements which will, perhaps, be small in the original well-scaled problem. There is evidently, however, more work

both in theory and practice that could be done on this problem.

It is certainly true that bandwidth minimisation (see references in chapter eight and Cuthill (1972)) is no longer a very popular tool for the solution of linear algebraic equations. In addition, there appear to be fairly well-defined limits to the amount of imagination that can be used in such processes. However, there may be some future in examining such techniques with a view to incorporating them in eigenvalue procedures as was suggested in section 4 of chapter eight. In fact, the results of chapter eight seem to indicate that there is plenty of scope in the field of eigensystems of sparse matrices for future work. Clearly, the classical methods of reduction to condensed forms, particularly in the case of symmetric systems, offer little encouragement to the sparse matrix analyst and avenues like those mentioned in section 4 of that chapter are worth exploring.

Virtually no mention has been made in this thesis on the solution of non-linear systems. Certainly much of the work of chapters 2 to 4 is directly applicable to non-linear systems as is explicitly stated in the papers of Steward (1962, 1965) and Sargent and Westerberg (1964), and since techniques for solving non-linear systems often involve iteration on linear systems, the sparsity techniques for linear systems described in chapter five and seven are directly applicable in the non-linear case (see, for example, Reid (1972b)).

Not terribly much has been said about the implementation of the algorithms. This thesis is concerned with considering codes applicable to the solution of general systems, but another approach is to have a pilot program which can generate a code to solve a particular sparsity structure. An example of this symbolic generation is found in Gustavson et al (1970). Evidently, there is a future to pursuing this approach to solving equations as can be seen in the work of Erisman (1972) who has developed this method so that the defect of having too much generated code is overcome and the algorithm is

proportionately speeded up. Some other techniques (which can also be used in the case of full matrices) can be incorporated into solution procedures. An example of this might be the use of Winograd's identity (Winograd (1968)) for calculating an inner product. This halves the number of multiplications at the cost of doubling the number of additions, clearly a saving in the computing costs for most machines.

Some work has been done recently in developing specialist languages. For example, Nuding and Kahlert-Warmbold (1970) have developed a language for dealing with matrices, while Rheinboldt and Meztenyi (1972) have developed a graph algorithmic language. APL (IBM) could also be considered a matrix oriented language. Certainly, just as it is easier to program in high level languages than machine codes, the use of such higher level languages does make the manipulation of the structures they are designed to deal with easier, but they often obscure the actual fundamental processes being executed and make a detailed analysis difficult. Certainly, they have their uses in computing science and algorithm development but to encourage their development would be to encourage their use .. not a particularly good thing for the reason mentioned above and, in addition, perhaps not all that useful because the high specialisation of such languages forbids the interdisciplinary nature of the field of sparse matrices.

Little mention has been made of the dynamic programming techniques of Bellman (1957), but they have been put to use in several sparse matrix procedures. For example, Sargent and Westerberg (1964) and Spillers and Hickerson (1968) suggest their use. Certainly they facilitate searches for optimal results but as yet the speed (or lack of it) of using such techniques has put them out of favour.

Finally, there has to date been too little detailed examination of the structures of sparse matrices arising in practical problems. Tewarson (1971) and Harary (1972) have made some attempt at a

classification of structures but theirs is a theoretically-based one and gives no indication of the relative occurence of the various structures in practice. In this thesis really only 3 sparse matrix structures are used and it would be very useful if more were readily available. This availability could well give rise to the development of specific techniques to deal with specific commonly occurring structures or hybrid methods applicable to some broader band in the calssification. It is certainly interesting to create examples to show what can arise (see the various counter-examples in this thesis or Wilkinson's example (1965) showing the failure of partial pivoting), but it is arguably more useful to consider examples which are likely to arise. For some sparse matrix problems, it is often the case that there is an 'obvious' way of tackling them and it would be interesting to find out how common these 'obvious' ways are in practice. Naturally, this examination would have to include matrices from many different fields and it might be found that specialist procedures made for matrices arising in one field were better than any attempt at general sparse matrix packages.

# CHAPTER 2.

## SELECTING A MAXIMAL TRANSVERSAL.

## Section 1. Introduction.

The subject discussed in this chapter has perhaps received more attention in the literature than any other particular part of this thesis. The topic has appeared in a number of papers under various guises in several different branches of applied mathematics from graph theory to operations research. All the algorithms which the author has been able to find in the literature are developed from the algorithm of Marshall Hall (1956) to whom should probably go the credit for first tackling this problem in an algorithmic form suitable for computation. The various papers differ in their method of presentation of the algorithm and one or two of the more interesting ones will be looked at in the next section.

The purpose of this chapter is threefold. Firstly, to explain what is being done in a sparse matrix context when the maximal transversal is selected and to correlate and rationalise previous attempts in this field. Secondly, to explain why this topic should have any importance in the field of sparse matrix research. And finally, to give a simplified form of the selection algorithm and an extension to it which greatly reduces computing costs by accelerating the transversal selection process considerably. Comparison runs for various methods on the KDF9 and 1906A machines are also given.

## Section 2. A rationalisation and explanation.

A longer but perhaps more explicit title for this chapter might be "finding a permutation of a matrix which puts non-zeros on the diagonal". Such a permutation will exist if and only if the matrix is structurally non-singular and is really an alternative definition for structural non-singularity (see chapter one). If a matrix is non-singular then certainly such a permutation exists but singularity does not imply that it does not exist. The example in figure 2.1 illustrates this.

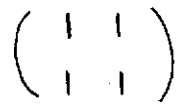


figure 2.1

The origins of the study of this problem are to be found in the various assignment problems (bottleneck, optimal, and Hitchcock) occurring in the field of operations research, (see Munkres (1957), Ford and Fulkerson (1957), Kuhn (1955), and Dulmage and Mendelsohn (1963b). Since the initial algorithm was published (Hall (1956)) several authors have described a variety of essentially similar algorithms under various guises. Every algorithm uses basically the theorem of Philip Hall (1935) which can be stated as follows:

Theorem (Philip Hall)

Consider a set $\mathfrak{R}$ of subsets $A_1,..A_n$ of a universal set S. Then it is possible to select one element from each subset such that the n elements chosen are distinct if and only if the set theoretic union of every selection of k of these subsets ($1 \leq k \leq n$) contains at least k distinct elements. Such a set of distinct elements, one from each subset is called a system of distinct representatives (SDR).

It is this SDR, not necessarily unique, which Marshall Hall's algorithm finds (or if a full system of n elements does not exist, his algorithm finds a maximal such set of distinct elements from different subsets). As an example, consider the set of subsets

$\{1,3\}, \{2\}, \{1,2\}$ from the universal set N, say.

Then an SDR for this system would be 3,2,1 which is, in this case (but not in general) a unique SDR. If the subset $A_i$ is identified with the row (or column) i of a matrix of order n and its elements are the column (row) indices of the non-zeros in row (column) i, then to obtain an SDR is to be able to assign a set of n non-zeros such that no two lie in the same row or column of the matrix. An obvious permutation then puts this non-zero assignment set onto the diagonal of the matrix and the problem is solved. If the above example is considered then the matrix of figure 2.2 is obtained

$$\begin{pmatrix} x & 0 & x \\ 0 & x & 0 \\ x & x & 0 \end{pmatrix}$$

figure 2.2.

where the x's represent non-zero elements. The above algorithm yields the SDR represented by the reverse diagonal of the matrix and the row permutation (1,3) brings these onto the main diagonal.

The two main publications in the operations research field tackling this question have been by Ford and Fulkerson (1962. p.55) and Yaspan (1966) where they use the term admissable elements for non-zeros and describe Hall's method for finding a maximal assignment identical to the SDR above. Simonnard (1962) has also published a revised version of the Ford-Fulkerson algorithm in his book on linear programming.

The paper of Dulmage and Mendelsohn (1963a) addresses itself to a more general theory of assignments in bipartite graphs but the algorithm which they develop for decomposing the graph (p.187) can be seen to be a generalisation of Hall's 1956 algorithm. The terminology of their paper is explained here since the terms are similar to those used in this thesis. The algorithm, however, is too general for the present purpose and so will not be discussed here.

A bigraph is defined to have two sets of points S and T and a set of lines E. A line (s,t) belongs to E implies that s $\epsilon$ S and t $\epsilon$ T. A subgraph of this graph is a set of points and lines belonging to S, T, and E respectively such that any lines in this subset join points of the subgraph only. A transversal is a subgraph, every point of which has exactly one line (i.e. no two lines of a transversal have a point in common). The order of a transversal is the number of its lines and a transversal of maximum order is said to be a maximal transversal. With every matrix there can be associated a bigraph by making the points of set S correspond to the rows of the matrix, the points of set T to the columns, and having line (i,j) belonging to E if and only if element $a_{ij}$ (the element in row i and column j) of the matrix is non-zero. For any matrix $\mathbf{A}$ of order n, each term in the fully expanded form of the determinant of a matrix consists of the product of the elements of the matrix, no two of which come from the same row or column,

$$\text{i.e.,} \quad \det \mathbf{A} = \sum (-1)^{\epsilon} a_{i_1 j_1} a_{i_2 j_2} \cdots a_{i_n j_n}, \qquad \ldots.(2.1)$$

where the sum is taken over all permutations of $\begin{pmatrix} i_1 \cdot\cdot i_n \\ j_1 \cdot\cdot j_n \end{pmatrix}$

among integers 1..n where $\epsilon$ = 1 or 0.

Then there is a one to one correspondence between the transversals of order $n$ and the non-zero terms in the expanded form of the determinant. In particular, any non-singular matrix of order $n$ will always have at least one maximal transversal of length $n$ (Duff (1970 p.95)). This transversal language is used by Weil and Kettler (1969) who have perhaps given the clearest exposition of Hall's algorithm in matrix terms. Unfortunately, they do not give any algorithm or detailed description of the method but merely a sketch of what it is about. This defect is remedied in section 4 of this chapter.

Another person to seriously tackle this problem from the standpoint of sparse matrices has been Steward (1962). He calls this set of $n$ elements taken from distinct rows and columns an output set since element $a_{ij} \neq 0$ is taken to mean that variable $j$ occurs in equation $i$ of the system of linear equations $Ax=b$, where $x$ and $b$ are both column vectors of order $n$. If element $a_{ij}$ is a member of the output set, the unique variable $j$ is the output variable of equation $i$ while all other variables appearing in equation $i$ (all $k$ for which $a_{ik} \neq 0$, $k \neq j$) are called the input variables of the equation.

## Section 3. The importance of selecting a maximal transversal in sparse matrix analysis.

The first part of this section explains why it is desirable to permute the matrix so that the diagonal elements are non-zero. The latter part then indicates (see theorem on page 28) that the block triangular structure remains unchanged irrespective of which n non-zeros are permuted to the diagonal. Thus it is necessary to select only a single output set and it is unnecessary to utilise Steward's algorithm (1962) to find further output sets.

Clearly, for the solution processes on block triangular systems (see chapter 3) it is necessary that all of the diagonal blocks are non-singular and therefore there exists at least one non-zero term in the expanded form of the determinant. Row and column permutations within each block can then be used to order these non-zeros onto the diagonal. Since this type of permutation does not alter the block structure of the matrix, there is no loss in generality in assuming that the permutation which puts the matrix into block triangular form is a combination of an ordinary permutation to put non-zeros on the diagonal followed by a symmetric permutation to put it into block triangular form. That is to say, the permutation of a matrix to block triangular form can be envisaged as a two stage process. The first stage is an unsymmetric permutation to put n non-zeros on its diagonal. This can be considered as, for example, a column permutation only. The following argument justifies this.

Clearly, any symmetric permutation $PAP^T$ corresponds merely to the renumbering of the points of the directed graph of A and so the structure of A and $PAP^T$ will be the same for the purposes of this discussion. This is denoted by $S(A) = S(PAP^T)$. Hence, for any matrix A and any arbitrary permutations P and Q, it

is true that

$$\begin{aligned} S(PAQ) &= S(P^{T}PAQP) \\ &= S(AQP) \\ &= S(AQ_1) \quad , \quad \text{say.} \end{aligned}$$

and thus it is possible, without loss of generality, to consider only column permutations of $A$.

The second stage is considered as a symmetric permutation to order the rows and columns in the appropriate diagonal blocks so that they are together in the final matrix. This process is shown in equations (3.1) and (3.2).

$$AQ = A^{1} \qquad \ldots \qquad (3.1)$$

$$\hat{A} = PA^{1}P^{T} \qquad \ldots \qquad (3.2)$$

where $A$ is a general non-singular matrix of order $n$

$A^{1}$ has all its diagonal entries non-zero

and $\hat{A}$ is block triangular.

If the step indicated by equation (3.1) is not performed, then the example of figure 3.1 shows that a $P$ as in equation (3.2) may not exist to render the matrix block triangular although, after choosing a $Q$ satisfying equation (3.1), $P$ does exist. For the example in figure 3.1 a suitable $Q$ is shown in figure 3.2 giving a block triangular matrix if the matrix $P$ of equation (3.2) is the identity matrix of order 2.

$$\begin{pmatrix} 0 & x \\ x & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

figure 3.1 figure 3.2

Obviously there is, in general, more than one matrix $Q$ in equation (3.1) which can permute the matrix to ensure that $A^{1}$

has non-zeros on its diagonal. It is the object of the following theorem to show that the structure of the block triangular form is independent of this Q. This theorem has already been stated in an intuitive manner by Steward (1962) and in a philosophical one by Duff (1970) but the following is a more concise statement followed by a proof of the statement.

Theorem 3.1

The block triangular form of any matrix A is independent of the set of n non-zeros that have been ordered onto the diagonal prior to the block triangular form being found.

Proof.

Before commencing this proof note that, whenever it is stated that a column j belongs to a block $A_j$, it is meant not that all of the column is in the block but rather that part of it is and no other part is in any other diagonal block. A similar comment applies to rows.

Let $A = [a_{ij}]$ be an arbitrary matrix of order n where, without loss of generality, it can be assumed that the diagonal of A has no zero elements on it, and let $B = [\ell_{ij}]$ be a column permutation of A which also has non-zeros on the diagonal, such that

$$B = AQ \qquad \ldots\ (3.3)$$

where the column permutation Q takes

$$\{1,\ldots n\} \longrightarrow \{i_1,\ldots i_n\}$$

Further, $\sigma$ is taken to be the member of the symmetric group $S_n$ which takes the permutation $\{1,\ldots n\}$ onto the permutation $\{i_1,\ldots i_n\}$.

Thus, $\sigma(j) = i_j \quad (j = 1,\ldots n)$. Let the unique cyclic decomposition of $\sigma$ be given by

$$\sigma = (s_1^1 s_2^1 \ldots s_{r_1}^1)(s_1^2 s_2^2 \ldots s_{r_2}^2) \ldots (s_1^t s_2^t \ldots s_{r_t}^t) \qquad \ldots (3.4)$$

where the S's are such that

$$\sigma(s_j^i) = s_{j+1}^i \qquad 1 \leq i \leq t, \quad 1 \leq j < r_i$$

$$\sigma(s_{r_i}^i) = s_1^i \qquad 1 \leq i \leq t$$

The assertion will be proved if it can be shown that:

(i) Columns i and j of A are in the same block if and only if columns $\sigma(i)$ and $\sigma(j)$ of B are in the same block.

(ii) Rows i and j of A, are in the same block if and only if rows i and j of B are in the same block.

(iii) The block containing row i of A is connected to the block containing row j of A if and only if the block containing row i of B is connected to the block containing row j of B.

The following trivial lemma is central to the proof.

Lemma: $b_{i\sigma(i)} \neq 0 \qquad 1 \leq i \leq n$

Proof of Lemma:

By the definition of A, $a_{ii} \neq 0$ for all i, and $a_{ii} = b_{i\sigma(i)}$ by the definition of $\sigma$.

Proof of theorem:

(i) If column i and column j are in the same block then there exists $(\alpha_1, \ldots \alpha_s)$, $(\beta_1, \ldots \beta_t)$ $1 \leq \alpha_i, \beta_i \leq n$, for all i such that

$$a_{\alpha_1 i}, a_{\alpha_2 \alpha_1}, \ldots a_{\alpha_s \alpha_{s-1}}, a_{j\alpha_s}, a_{\beta_1 j}, a_{\beta_2 \beta_1}, \ldots a_{i\beta_t} \neq 0.$$

but this implies, by virtue of the definition of $\sigma$ and $b$ that:

$$b_{\alpha_1 \sigma(i)}, b_{\alpha_2 \sigma(\alpha_1)}, \ldots b_{\alpha_s \sigma(\alpha_{s-1})}, b_{j\sigma(\alpha_s)}, b_{\beta_1 \sigma(j)}, b_{\beta_2 \sigma(\beta_1)}, \ldots b_{i\sigma(\beta_t)} \neq 0.$$

but,

by the definition of $\sigma$ given in equation (3.4), it is

clear that $\alpha_1$ lies in one and only one of the disjoint cycles in the decomposition of $\sigma$. This, combined with the lemma, implies that

$$b_{\alpha_2\sigma(\alpha_1)}, \; b_{\sigma(\alpha_1)\sigma^2(\alpha_1)}, \; b_{\sigma^2(\alpha_1)\sigma^3(\alpha_1)} \neq 0 \qquad \text{etc.}$$

and, in fact, since $\alpha_1$ is in this cycle, there exists an integer $\ell$ such that

$$b_{\sigma^{\ell}(\alpha_1)\alpha_1} \neq 0 .$$

which means that there exists a chain $k_1, \ldots k_\ell$ say such that

$$b_{\alpha_1\sigma(i)}, \; b_{k_\ell\alpha_1}, \; b_{k_{\ell-1}k_\ell}, \; \ldots \; b_{k_1k_2}, \; b_{\alpha_2k_1} \neq 0 .$$

where

$$k_r = \sigma^r(\alpha_1) \qquad 1 \leq r \leq \ell$$

and

$$k_{\ell+1} = \alpha_1 .$$

In this fashion, it is possible to form a chain from $b_{\alpha_1\sigma(i)}$ through $b_{j\sigma(\alpha_s)}$ and $b_{\beta_t\sigma(j)}$ back to $b_{i\sigma(\beta_t)}$. This implies that columns $\sigma(i)$ and $\sigma(j)$ are in the same block of the matrix B.

By using the inverse permutation (uniquely defined) to $\sigma$, the sufficiency part of the first part of the theorem is proved.

(ii) The proof for this part is almost identical and so will not be repeated here.

(iii) This part is fairly trivial. Let $A_1$ and $A_2$ be the two blocks of A and $B_1$ and $B_2$ the corresponding blocks of B. Then $A_1 \leq A_2$ if there exists $i \in A_1$ and $j \in A_2$ such that $a_{ij} \neq 0$. This implies that $b_{i\sigma(j)} \neq 0$, but $i \in B_1$, and $\sigma(j) \in B_2$ from parts (ii) and (i). Since the above proof is immediately reversible, part (iii) has been proved.

The theorem has now been proved.

The following theorem now follows from the discussion at the beginning of this section.

Theorem 3.2

If A is a matrix which has non-zeros on its diagonal, then A is bireducible if and only if it is reducible.

Proof

From the definitions of the terms used in the statement of the theorem it is evident that reducibility implies bireducibility.

Assume A is bireducible and P and Q are permutation matrices such that PAQ is block triangular. Then, there exists a permutation matrix $Q_2$ which orders the columns within single blocks only such that $PAQQ_2$ is also block triangular and the diagonal elements are diagonal elements of the original A. Since this is so, the matrix $PAQQ_2$ is a symmetric permutation of A making $QQ_2 = P^T$ and indicating that the symmetric permutation $PAP^T$ renders A block triangular.

The theorem has now been proved.

The following Corollary is immediately evident.

Corollary 3.2.1.

Any non-singular matrix A of order n is bireducible if and only if any permutation of A with non-zeros on the diagonal is reducible.

Proof. The proof is trivial.

When theorems 3.1 and 3.2 are examined in the context of this section, it is seen that if a general matrix A is ordered by any column permutation to have non-zeros on the diagonal (see equation

3.1) and if the subsequent matrix $A^1$ is symmetrically ordered to be block triangular as in equation 3.2 , then this block form is unique (to within permutations within the blocks) and, furthermore, if no such P exists then the original matrix is biirreducible by Corollary 3.2.1. Thus, any algorithm based on finding the matrices P and Q of equations (3.1) and (3.2) will always succeed in ordering an arbitrary matrix to its unique block triangular form.

## Section 4. Methods for finding a maximal transversal.

As mentioned in section 1, the original algorithm for finding a maximal transversal was described by Hall (1956). This method has been described in various ways and, from these, the description due to Weil and Kettler (1969) has been the one chosen for this section.

This paper certainly presents a very simple and clear exposition of Hall's algorithm and it is this explanation which is developed in the following pages. This method shall henceforth be referred to as Hall's algorithm.

In Hall's algorithm, the matrix is scanned until a path is found connecting a non-zero in column k+1 in the NE corner of the matrix to a non-zero in the SW corner. An example is given in figure 4.1. It is, of course, assumed that the SE corner is zero otherwise a non-zero there would be used immediately to extend the transversal.

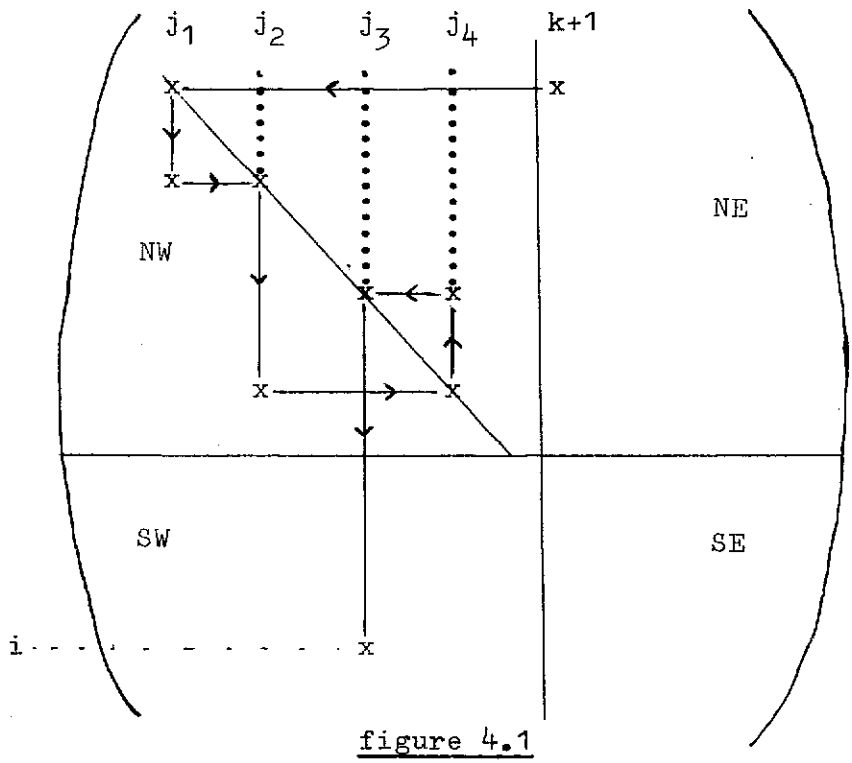



figure 4.1

Columns are then interchanged in a cyclic manner (called a

circular column shift) to bring this element (a non-zero in the SW corner) to the SE corner where it can be easily used to extend the transversal. In figure 4.1., the required column shift would be $k+1 \longrightarrow j_1 \longrightarrow j_2 \longrightarrow j_4 \longrightarrow j_3 \longrightarrow k+1$. Row i is then interchanged with row $k+1$ to bring this SW corner non-zero to the $k+1$ th position on the diagonal, $k$ is incremented by 1 and the above procedure is repeated until no such column shifts can be found or a maximal transversal of order $n$ is obtained.

To be more precise, the process can be described in the following terms. Scan column $k+1$ for a non-zero and mark this column and the row of the non-zero. Scan the column whose index is the same as that of this row to find a non-zero off-diagonal entry in an unmarked row. Three things can happen at this stage

(i) The non-zero just found lies in a row of index greater than $k$ implying that the non-zero in the SW corner has been found together with the required column shift to bring it to the SE corner.

(ii) The non-zero just found lies in a row of index less than $k+1$. The row and column of that non-zero are marked and the process continues by scanning the column, whose index is the same as this row, for non-zeros.

(iii) No such non-zeros exist in the column. The method then proceeds by marking this column and examining all the marked columns for non-zeros in unmarked rows. If one is found, then its row is marked and the scan continues by looking at the column (which will be unmarked at this stage) whose index is that of this row.

This searching procedure when systematically implemented is called a <u>tree search</u>.

It is evident that, since the number of rows and columns is

finite, the process must terminate either by finding a non-zero in the SW corner or by discovering in (iii) above that no such non-zero can be found. This means, because of the marking processes pursued, that there exists a set of, say, r+1 marked columns ($1 \leq r \leq k$) which have all their non-zero elements in only r rows (which are the marked rows). Hence, by an application of Philip Hall's theorem given on page 22, no assignment can be made and the matrix is declared singular.

This algorithm is described in the form of a flow chart on page 37. This flow chart is detailed enough to be the basis for a program flow chart and is included as a supplementary aide to understanding Hall's algorithm. The various variables and arrays used in this flow chart are now defined.

For a straight application of Hall's method (method 1 on page 38), k is set to zero before beginning the algorithm. The value of k at any subsequent stage indicates the current order of the transversal. TC and TR are integers holding the column index and the row index of the particular non-zero under consideration, either a diagonal non-zero element (a member of the output set in Steward's terminology) or a non-zero being considered as a possible new member of a future output set. TAGR is a 1×n array whose ith element has value 1, if row i has been chosen in a previous part of the search, and zero otherwise. PC is a 1×n array whose ith element is the column index of the column previous to i in a loop (a loop here being equivalent to the circular column shift of figure 4.1). PC(k+1) equals -1.

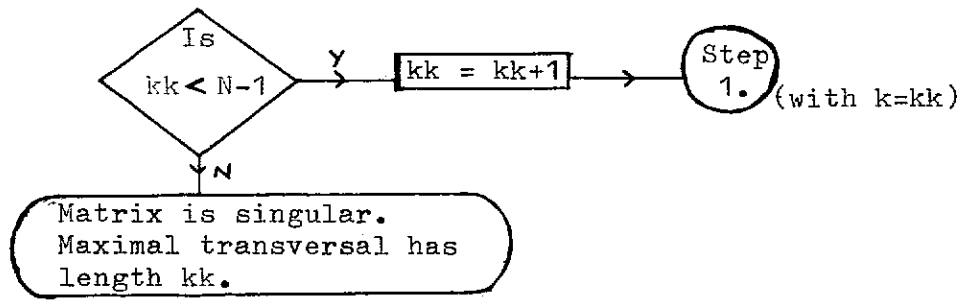
The following notes can be made about this presentation of the algorithm.

(a) The algorithm always terminates at either *C or *D. This happens because a TAGR(i) previously zero is put equal to 1 at

stages *A and *B. Thus, if it does not terminate at *D, there must eventually come a stage at which all the rows have TAGR(i) = 1 and so the algorithm terminates at *C.

(b) The argument here is identical to that used beneath note (iii) on page 34. PC is set equal to -1 at step 1 and at steps *A and *B both an element of PC and one of TAGR are set from zero to non-zero. Thus the number of marked columns (PC(i) $\neq$ 0) is always one greater than the number of marked rows (TAGR(i) $\neq$ 0).

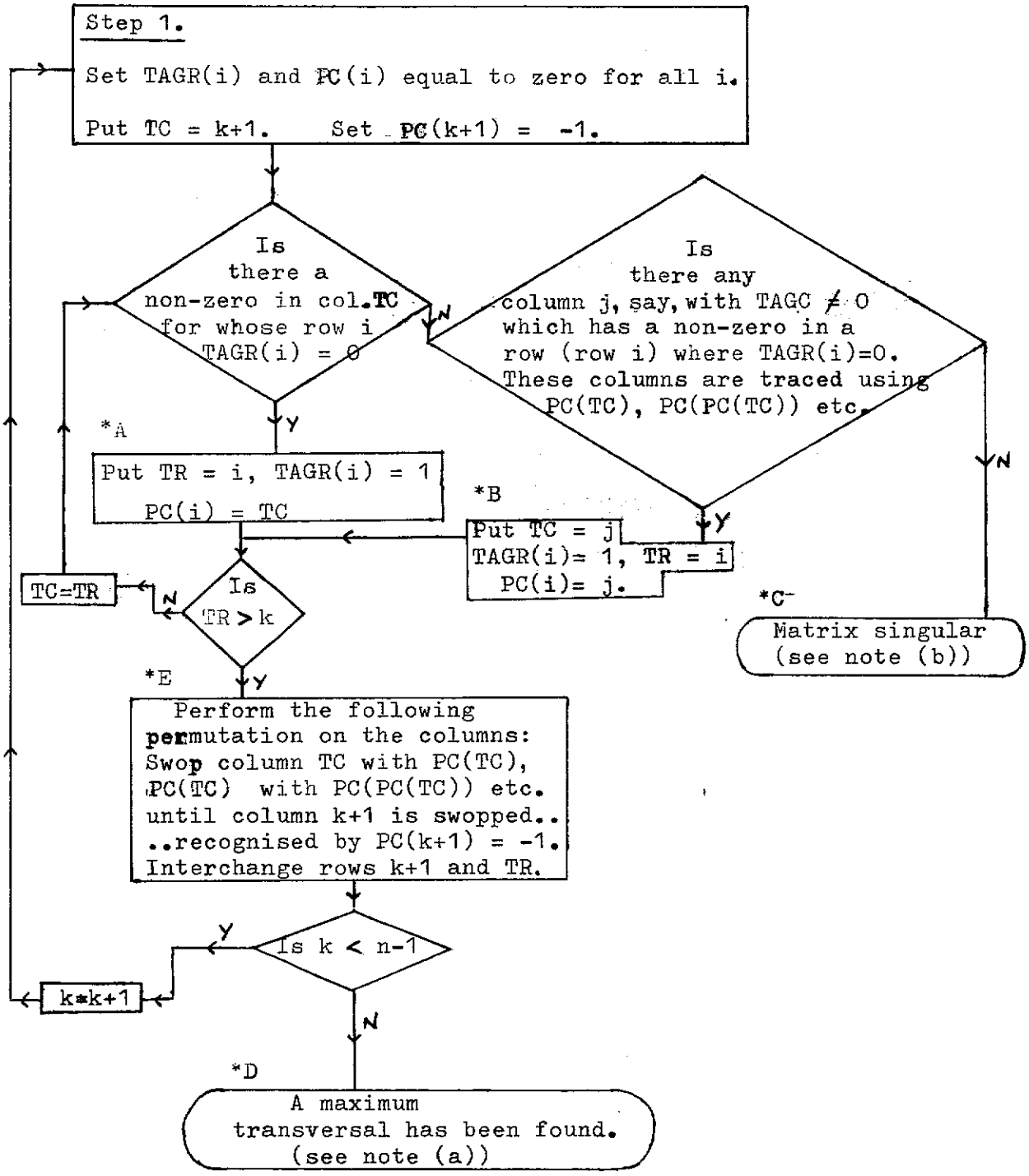
(c) If the matrix is structurally singular causing an exit at *C, then it should be noted that this algorithm will not necessarily yield a maximum transversal. If it is desired to do this *C should be replaced by




where kk is set equal to k at the beginning and whenever it is incremented.

Hall's method is the kernel of all four methods which are described here and compared experimentally for time and efficiency in section 5.

An algorithm for stretching the transversal



Method 1.

This method uses Hall's method starting with $k = 0$. Steward in his paper (1962) suggests starting this way although, as is seen from the results, this makes the procedure rather slow. The program used for the experiments reported here is based on that paper and the flow chart for this program can be found on page 58 of Duff (1970). It is a bit untidier than the flow chart on page 37 illustrating Hall's method, but is similar in logic and time of execution.

Method 2.

As a preliminary to this and the other two methods a couple of definitions are first made. $r_i$ is defined to be the number of non-zero elements in row $i$ of the matrix and $c_j$ the number of non-zeros in column $j$.

If the element count $r_i + c_j$ is calculated for every non-zero element, then the element with minimum element count can be chosen as the first element of the transversal. The element with minimum element count in the submatrix obtained by deleting the row and column of this first selected transversal element is then chosen as the second transversal element and the process continues choosing elements in order of ascending element count thus obtaining a transversal of the matrix. This transversal need not be maximal (see section 5), but has, however, the advantage of being quickly obtained and can then be extended to a maximal one by Hall's method, where $k$ is set equal to the length of the transversal found in the first part and the matrix is first permuted to put these $k$ non-zeros at the beginning of the main diagonal.

Method 3.

The $r_i + c_j$ of method 2 are again calculated for each non-zero

element but this time instead of being used as an 'a priori' criterion for transversal selection these element counts are updated at each stage (i.e. after the selection of each transversal element). This can be expressed in the following manner:-

At stage $k+1$ we are left with a square matrix of order $n-k$. $r_i^{(k+1)}$ and $c_j^{(k+1)}$, the row and column counts for this reduced matrix, are then calculated either from scratch or by updating the previous element counts. This is very easily done by looking along the row (in the reduced matrix) of the transversal element selected at stage $k$ and reducing by one the column count of any column with a non-zero in that row. A similar search of the column of the stage $k$ transversal element updates the row counts.

Stage $k+1$ is then completed by selecting the non-zero element with $\min_{i,j} (r_i^{(k+1)} + c_j^{(k+1)})$, marking it, and proceeding to stage $k+2$ with the matrix of order $n-k-1$ obtained from the matrix of stage $k+1$ by eliminating the row and column corresponding to this selected transversal element. (One way of effectively eliminating the above mentioned row and column is to put

$$r_s^{(k+1)} = 2n$$

$$c_t^{(k+1)} = 2n$$

if element $(s,t)$ was the transversal element selected at stage $k$. The restriction that $\min (r_i^{(k)} + c_j^{(k)})$ must be less than $2n$ or else the algorithm terminates is then imposed). The algorithm terminates either after stage $n$ or when the reduced $n-k+1$ submatrix has all zero entries ($\min (r_i^{(k)} + c_j^{(k)}) > 2n$ using method outlined in the bracket above). It has been found experimentally (in section 5) that without taking substantially

longer time than the first part of method 2, it has been generally possible, using this method, to select a transversal which is maximal. In a small percentage of cases, however, Hall's method has to be resorted to, again setting k equal to the length of the transversal found and permuting the output set on to the main diagonal (see section 5).

Method 4.

Clearly, in the case of a nonsingular matrix or reduced matrix, singletons will lie in every term in an expansion of the determinant as in equation (2.1) and so must be included in any maximal transversal.

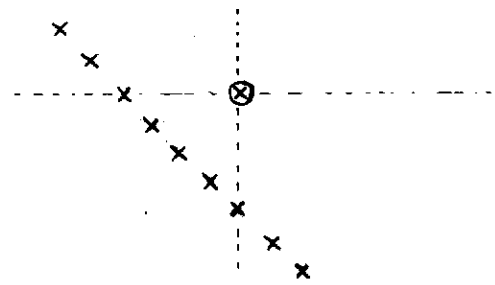


figure 4.2

If the diagram in figure 4.2 is examined where the diagonal line of non-zeros represents a maximal transversal, then it is seen that the removal of a row and column from a matrix reduces the length of its maximal transversal by at most two. This can only happen in cases like those shown where the circled non-zero is the element being examined as a possible transversal element and its row and column are under consideration for removal from the matrix. Clearly, when this happens the circled element can never be a singleton (a singleton is a non-zero element lying in a row or column whose other elements are all zero). Therefore, the choice of a singleton can never eliminate more than one transversal element from future consideration and is therefore always a good choice for such an elimination. Thus, it is

advisable, even if the matrix or submatrix is singular, to choose singletons as transversal elements.

Method 4 can thus be described in the following manner.

(i) Select all non-zeros lying in singleton rows and columns of the matrix or reduced matrix, i.e. select all $a_{ij}^{(k)} \neq 0$ for which $r_i^{(k)} = 1$ or $c_j^{(k)} = 1$. Increase k appropriately. Go to (ii)

(ii) Choose the non-zero element of the matrix (reduced matrix) which has minimum element count, i.e. select $a_{ij}^{(k)} \neq 0$ for which $r_i^{(k)} + c_j^{(k)}$ is minimum. Ties are broken by choosing the first such element encountered in a row scan of the matrix from left to right and top to bottom. Increase k by 1. If the reduced matrix is the zero matrix or n transversal elements have been selected go to (iii), else go to (i).

(iii) If $k = n$, then a maximal transversal has been found. If not, Hall's method with appropriate k and permuted form is used to extend the transversal to a maximum.

That this method is more powerful than that of Method 3 is indicated by means of the example in figure 4.3. Method 3 would select the underlined elements as transversal elements, $\underline{X}_i$ being the ith such element selected. This method fails by observing that the ticked rows violate Hall's condition for an SDR. Method 4 works on this example (without the use of the latter part of (iii) above) and selects the circled elements as transversal elements, the numbers alongside the rows and columns in figure 4.3 indicating the order in which selection takes place. The numbers of the elements indicate the initial element count of the non-zero elements.

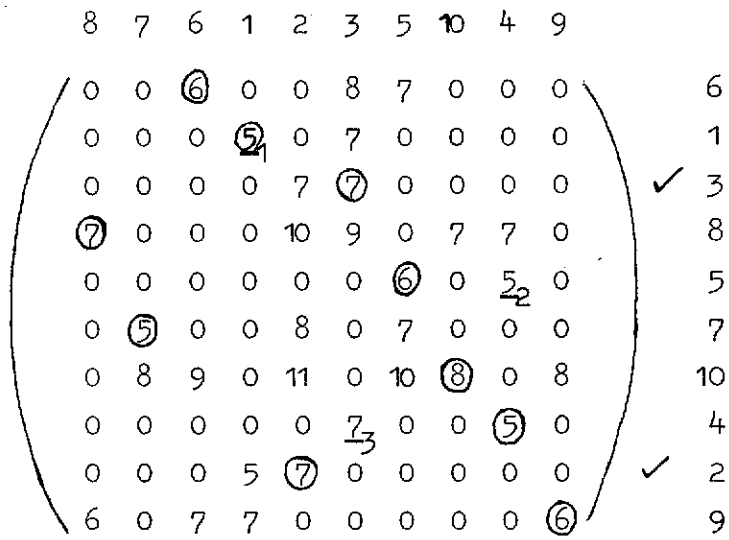



figure 4.3.

As well as working successfully on this example it is observed that method 4 (without resorting to the use of step (iii)) produces a maximal transversal in the 3 x 3 example of Weil and Kettler (1969), the 10 x 10 example of Yaspan (1966), the 9 x 9 example of Ford and Fulkerson (1962), the 7 x 7 example of Dulmage and Mendelsohn (1967), the 7 x 7 example of Steward (1962), and the 4 x 4 example of Simonnard (1962). Preliminary tests by the author on random matrices in February 1971 (tables 5.1 and 5.2) indicated 100% success for obtaining a maximal transversal using method 4 without using step (iii) and it was at one time thought that step (iii) might be unnecessary. This, however, is not the case as is shown by the results of subsequent runs on random matrices in table 5.3 and the following discussion which creates a minimal counter-example for which method 4 will, irrespective of tie-breaking procedures, require the use of step (iii).

The counter-example is constructed by making the element with $\min_{i,j} (r_i + c_j)$ not belong to any non-zero term in the expanded form of the determinant. Since there can be no singletons in the counter-example the minimum element count must

be greater than or equal to four and, if ties are to be avoided, then four is also a lower bound on the order of the counter-example. When the non-zero with the lowest element count has been selected as a transversal element and has its row and column deleted, the remaining matrix must be structurally singular. Hence, if a counter-example of order n exists, the statements of the last two sentences, imply that the remaining n-1 x n-1 matrix left after the removal of the row and column of the selected transversal element must be singular and contain no zero rows or columns. Further, it must have no singleton rows or columns since otherwise the non-zero element making the singleton row or column a doubleton (this non-zero exists since there are no singleton rows or columns in the full n x n matrix) which lies in the column or row of the selected transversal element must have an element count less than or equal to it and so there is a contradiction. Since a singular 3 x 3 matrix without any zero or singleton rows or columns does not exist, the smallest possible value for (n - 1) is four. Any 4 x 4 singular matrix with no zero rows or singletons must have a 3 x 3 singular submatrix with no zero rows and the only matrix (unique to within permutations) satisfying this is shown in figure 4.4 where A can be non-zero or zero. This gives the only 4 x 4 matrix with no zero-rows or singletons and a singular 3 x 3 block as that in figure 4.5

$$\begin{pmatrix} A & X & X \\ X & 0 & 0 \\ X & 0 & 0 \end{pmatrix}$$

figure 4.4.

$$\begin{pmatrix} X & A & X & X \\ A & A & X & X \\ X & X & 0 & 0 \\ X & X & 0 & 0 \end{pmatrix}$$

figure 4.5.

This matrix is clearly seen, by virtue of its reverse diagonal to be structurally non-singular. Hence, if any counter-example

(without ties) to method 4 exists, it must be of order six or more. Such a counter-example is provided in figure 4.6 where again the number on each non-zero represents its initial element count. Clearly, the 5 x 5 matrix left after selecting the 1,2 element as a transversal element on the basis of untied $\min_{i,j} (r_i + c_j)$ is singular and the greatest transversal obtainable by method 4 is of order five. However, a transversal of order six for the matrix does exist with possible transversal elements being those on the main diagonal.

$$\begin{pmatrix} 5 & 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 6 & 6 & 0 & 0 \\ 0 & 0 & 5 & 5 & 0 & 0 \\ 0 & 0 & 5 & 5 & 0 & 0 \\ 6 & 0 & 0 & 0 & 5 & 5 \\ 6 & 0 & 0 & 0 & 5 & 5 \end{pmatrix}$$

figure 4.6.

## Section 5. Experimental results and conclusions.

The four methods described in section 4 were tested with several different random matrices. In table 5.1 the results using 16 matrices of order 100 and probability density about 0.03 are given. These runs were made on the KDF9 and the timings given are in seconds of KDF9 CPU time. The figures opposite method 2 above the timings indicate the order of the transversal selected before Hall's method was applied. In table 5.2 similar runs were made with matrices of varying order and density in order to see if this would have any effect on the relative timings. These runs were made on the 1906A and the timings are in seconds of 1906A milltime. Finally, methods 2, 3, and 4 were run against a sizeable number of matrices of varying size and density to see on what percentage of examples these methods yielded a maximum transversal without recourse to Hall's method. These results are shown in table 5.3.

The following notes can now be made.

Notes

(i) From the times in table 5.1 it is seen clearly that method 4 is substantially better than the other three methods when tested with 16 sparse matrices of order 100 and probability density 0.03. The improvement over Hall's method (method 1) being sometimes about 75%.

(ii) A similar trend is seen in table 5.2 where matrices of varying orders and densities were used. Again the times are such that the efficiency of the methods increases as one goes down the table from method 1 to method 4. The results of this table indicate that the trend shown in table 5.1 for matrices of a fixed density and order is also true over a wider range of random matrices. Method 4 is always the quickest method available being

fairly regularly about 70% or more better than method 1. The times are completely different from those of table 5.1 since the runs were made on different machines and the KDF9 times include the time taken to send the results to an output file (the same for all four methods).

(iii) It would be unfair to draw any further conclusions from table 5.2 as regards, for example, the behaviour of the various methods as the sizes and densities vary. This is because there was a fairly large variation in times over the three different examples tested at each size and density. The trend as regards the comparison of the various methods was, however, the same, thus allowing the deductions of note (ii) to be made.

(iv) Table 5.2 also clearly shows the advantage of selecting singletons when possible since method 4 is sometimes over 50% better than method 3. In many cases (i.e. when Hall's method is not invoked) these methods only differ in this respect.

(v) From the results of table 5.1, it is seen that even when method 2 failed to give a maximal transversal without the use of Hall's method, it was still much faster than using Hall's method from scratch. The results of this table also show that the 'a priori' sort of method 2 produced transversals of average length 95 (standard deviation 1.8), while method 3 required the use of Hall's method only once. In most cases this gain gave method 3 a faster overall time than method 2 in spite of having to update the element counts at each stage. In every case, in tables 5.1 and 5.2 method 4 gave a maximal transversal without recourse to Hall's method.

(vi) The results of table 5.3 continue an investigation of this phenomenon. It is seen that occasionally method 4 does require the use of Hall's method to select the maximum transversal but that this happened in less than 1% of the cases tried. On

similar matrices taken from a fairly wide distribution of densities (0.02 to 0.30) and orders (20, 40, 50, 100) method 3 required Hall's method for its completion 10% of the time while method 2 required it more than 50% of the time. (vii) The times given in the tables 5.1 and 5.2 may seem, at first glance, fairly high. The principal reason for this is that no proper advantage has been taken of the sparsity of the matrix thus making the time for all the methods dependent on $n$, the order of the matrix, rather than $\bar{r}$, the average number of non-zeros in a row of the matrix and reduced matrices. Some tests and a careful consideration of the algorithms indicate that this inefficiency biases the times fairly proportionately, and so the comparisons and general trends will still be valid using this better implementation. Since there is this little subtlety in the actual programs used in this chapter they have not been included in the appendices.

The results of this section clearly establish method 4 as the one which should be used to select the maximal transversal of a matrix before going on to the block triangularisation phase described in the next chapter.

| METHOD \ MATRIX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| METHOD 1 | 290 | 340 | 291 | 459 | 310 | 244 | 331 | 238 | 322 | 254 | 336 | 316 | 279 | 314 | 185 | 168 |
| | 96 | 93 | 95 | 95 | 93 | 98 | 94 | 97 | 96 | 94 | 95 | 95 | 97 | 98 | 94 | 97 |
| METHOD 2 | 186 | 198 | 176 | 170 | 218 | 174 | 182 | 169 | 176 | 177 | 190 | 225 | 171 | 173 | 170 | 181 |
| | | | | | | | | | | 98 | | | | | | |
| METHOD 3 | 172 | 171 | 171 | 171 | 173 | 171 | 172 | 172 | 172 | 185 | 171 | 174 | 171 | 172 | 173 | 172 |
| METHOD 4 | 121 | 107 | 114 | 119 | 117 | 117 | 114 | 123 | 121 | 121 | 120 | 116 | 118 | 121 | 123 | 122 |

Table 5.1 Comparison for run times of methods in section 4. Units are seconds of KDF9 time. Matrices are of order 100 with probability density about 0.03.

| MATRIX n. / METHOD p. | 100 0.05 | 100 0.04 | 100 0.03 | 100 0.02 | 50 0.10 | 50 0.06 | 50 0.05 | 50 0.04 | 25 0.20 | 25 0.10 | 25 0.05 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| METHOD 1 | 20.233 | 19.606 | 35.306 | 39.927 | 2.112 | 3.088 | 1.906 | 3.388 | 0.2348 | 0.2223 | 0.2960 |
| METHOD 2 | 12.747 | 8.896 | 10.042 | 8.091 | 1.373 | 1.275 | 1.222 | 1.133 | 0.2321 | 0.1329 | 0.1699 |
| METHOD 3 | 9.015 | 7.774 | 7.187 | 9.195 | 1.341 | 1.105 | 0.9872 | 1.035 | 0.2143 | 0.1516 | 0.2126 |
| METHOD 4 | 7.373 | 5.449 | 3.986 | 1.567 | 1.126 | 0.7062 | 0.4638 | 0.3683 | 0.2000 | 0.0996 | 0.0401 |
| PERCENTAGE IMPROVEMENT OF METHOD 4 OVER METHOD 1 | 63.6 | 72.2 | 88.7 | 96.1 | 46.7 | 77.1 | 75.7 | 89.1 | 14.8 | 55.5 | 86.4 |

Table 5.2 Runs of various transversal selection methods with matrices of varying orders and densities. Times are in seconds of 1906A milltime. Each run was made with three different matrices of the same size and density and the average time taken.

| METHOD | NUMBER OF MATRICES TESTED | NUMBER OF ATTEMPTS REQUIRING HALL'S METHOD |
|---|---|---|
| METHOD 2 | 170 | 90 |
| METHOD 3 | 170 | 17 |
| METHOD 4 | 368 | 3 |

Table 5.3 Runs with matrices of various orders (20 - 100) and densities (0.02 - 0.30) to investigate whether methods 2 - 4 yield a maximum transversal without the use of Hall's method.

# CHAPTER 3.

## ORDERING TO BLOCK TRIANGULAR FORM

## Section 1. Introduction.

If a general matrix A is ordered such that $a_{ii} \neq 0$ for all i, then it is apparent, from theorems 3.1 and 3.2 of the last chapter, that there exists a permutation matrix P such that $PAP^T$ is in block triangular form (although, if the matrix A is biirreducible, then this form may consist of only one block). Furthermore, apart from a reordering within these blocks, this form is unique (given that the diagonal blocks are biirreducible).

It is, of course, quite possible and is often the case in practice that systems already have non-zeros on the diagonal. If this is so, then the algorithms in this chapter can be used without prior sorting by the methods of chapter two. Although matrices arising from the discretisation of differential equations do not readily admit of such a block triangular ordering many systems arising in the operations research area and in chemical engineering do possess the property of being so reducible. This chapter indicates the importance of finding suitable permutations to reduce the matrix and also gives some idea of how to go about finding such a permutation.

The motivation for finding the permutation is given in section 2, while section 3 examines some existing algorithms in the field. These algorithms are viewed in a graph-theoretic context and from this unifying theory a new method, method 2, is developed in section 4. Section 4 also discusses a minor improvement to one of the established methods in the field. This is called method 1.

The results of section 5 indicate that method 2 is competitive with all the other methods tested, particularly if the matrix under consideration is structured or relatively dense. It has the disadvantage of being weak for very sparse random systems but improves as the density increases contrary to the behaviour of the other two better methods, those due to Steward (1962, 1965) and Sargent and Westerberg (1964). Some computational details are also given in this section.

## Section 2. Sparsity and pivoting considerations.

In this section, some reasons are given for determining the diagonal blocks of a bireducible matrix. All the reasons are based on the preservation of sparsity or the reduction of induced instability during the elimination process.

The main reason for identifying the diagonal blocks is to reduce what may be a very large problem into the sequential solution of smaller problems. Consider the block triangular system of figure 2.1.

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ & A_{22} & A_{23} \\ & & A_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

figure 2.1

The solution vector to this system can be found in the following manner:

$$\left.\begin{aligned} A_{33}\, x_3 &= b_3 \\ A_{22}\, x_2 &= b_2 - A_{23}\, x_3 \\ A_{11}\, x_1 &= b_1 - A_{12}\, x_2 - A_{13}\, x_3 \end{aligned}\right\} \quad \ldots\ldots (2.1)$$

Equation (2.1) indicates that the solution to the system in figure 2.1 can be found by solving three separate systems whose coefficient matrices are $A_{33}$, $A_{22}$, and $A_{11}$. In the case where the coefficient matrix of figure 2.1 is equipartioned, each of these submatrices are only one third as large as the original system. Thus, at any given time in the solution process, a much smaller system than the original one is being considered. Hence it is possible to use a backing store routine which only brings into core particular parts of the original system when they are required during the solution process.

This will be particularly important when the system of figure 2.1 is too large to hold in the main store even when the matrix is held in packed form.

However, even if it is assumed that there are no core storage limitations on the system, then there are still other good reasons for obtaining the diagonal blocks prior to implementing the elimination process. The principal one is easily observed by noticing what happens if full Gaussian elimination is performed on the system when its structure is unknown. For example, if a non-zero in an off-diagonal block is chosen as a pivot and an elimination is performed then the situation shown in figure 2.2 results, where the row and column of the pivot p are marked by dotted lines.

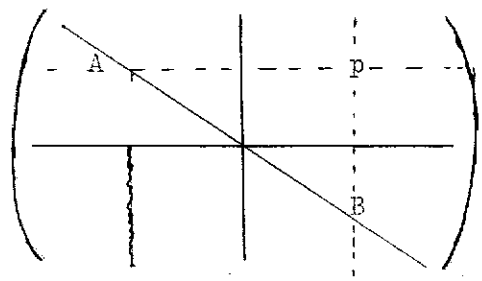



figure 2.2

With p as the pivot the column designated by a wavy line in the zero submatrix will be filled-in according to the density of the dotted column in the submatrix B. Similarly, fill-in can occur in the block containing the element p if a pivot is chosen from the diagonal blocks A or B. However, if pivots are first chosen from B and Gaussian elimination is performed only within that block thus reducing it to upper triangular form before any other operations are done on the system, then a back substitution can be carried out to reduce B to the unit matrix and the off-diagonal block containing p to zero. Proceeding in this fashion causes no fill-in in the off-diagonal blocks and leaves the unaltered submatrix A to be decomposed by Gaussian elimination again causing no fill-in outside the diagonal blocks. Of course, any pivoting strategy normally used for sparse matrices can be utilised in minimising fill-in and

element growth within the diagonal blocks themselves. It is important to note that this method requires prior use of block triangularisation techniques to determine the rows and columns of the diagonal blocks. A similar decomposition method for systems which are nearly block triangular is described in detail in the next chapter. The benefits, in terms of operation count, of using this method are obvious.

The benefits, in terms of stability, of using such a scheme can be seen by the fact that, if no fill-in occurs in the off-diagonal blocks, then no growth can occur in that part of the matrix. Therefore, it is highly likely that the induced instability is lower than if fill-in and growth occurred there since growth is directly related to instability (see Reid (1971c)). In addition, since the region in which growth can occur has been confined, it makes the task of monitoring any growth that much easier.

In the jargon of backward error analysis (Wikinson (1965)), it can be said that by pivoting within the diagonal blocks the only perturbations allowed to the matrix of figure 2.1 are those which leave the off-diagonal submatrices unchanged. This restriction of perturbation can be quantified by saying that the solution obtained is the exact solution of the perturbed system with coefficient matrix $A + E$, where $E$ is partitioned similarly to $A$ with zero blocks in the off-diagonal positions. Clearly, this restriction can only have a beneficial effect on the accuracy of the solution to the system if the other elements of $E$, in both cases, are similarly bounded.

Since the diagonal blocks of a non-singular block triangular matrix are themselves non-singular and they always have a better condition number than the complete matrix, it is not really surprising that solving the subsystems sequentially is better than solving the whole system together. As is seen, the order in which the subsystems are solved is also important.

The following interesting theorem is now presented and is referred to again at other points in this thesis.

Theorem 2.1

If $A$ is a matrix which can be partitioned into block upper triangular form with $n$ diagonal blocks $A_{ii}$ $(1 \leq i \leq n)$, then there always exists a scaling, which does not alter the condition numbers of the blocks $A_{ii}$, such that pivots will be chosen from only the diagonal blocks if partial pivoting by rows or columns or total pivoting is utilised.

Proof.

The proof is constructive and uses a scaling matrix described by Willoughby (1970, 1971a).

Let $I_i$ be the identity matrix of the same order as the diagonal block $A_{ii}$ of the system and consider the matrix $D$ of equation (2.2).

$$D = \text{diag}(\varepsilon^{-1}I_1, \varepsilon^{-2}I_2, \ldots \varepsilon^{-n}I_n) \qquad \ldots (2.2)$$

Then, if the scaled matrix $DAD^{-1}$ is considered, it is seen that the diagonal blocks (and hence their condition numbers) remain unaltered and the off-diagonal blocks $A_{ij}$ are multiplied by powers of $\varepsilon$, such that the $(i,j)$th block of $DAD^{-1}$ is equal to $\varepsilon^{j-i}A_{ij}$. Since $A$ was block upper triangular, $j$ is greater than $i$ and these powers of $\varepsilon$ multiplying the $A_{ij}$ are all positive.

Clearly, by choosing $\varepsilon$ sufficiently small the required scaling matrix given in equation (2.2) has been found and the theorem is proved.

Clearly, a similar theorem holds for block lower triangular matrices and so the adjective "upper" could be omitted from the statement of theorem 2.1 without affecting its validity.

Although the scaling given in theorem 2.1 is not a very practical

one since matrices are generally scaled so that their non-zero elements have roughly the same size, it does indicate that it is possible to scale the matrix so that, even on a strictly numerical basis, pivots would always be chosen from within the diagonal blocks. In addition, a scaling on the diagonal blocks can be superimposed on the previous one to reduce the condition number of these blocks and of the matrix as a whole.

## Section 3. A brief look at some existing algorithms.

In a similar way to the algorithms for obtaining a maximal transversal which were mentioned in the last chapter, there have been many algorithms devised over the last ten to fifteen years for finding row and column permutations which reduce a matrix to block triangular form. Unlike the transversal algorithms which are very similar to each other in nature, many of the algorithms developed for dealing with this problem do attack it from somewhat different angles. In the discussion of current methods which follows and in the next section, a more unified approach to these algorithms is developed. This unification can be interpreted in graph theoretic terminology and is used to develop a new method, method 2 of section 4.

Frank Harary (1959b) can probably take the credit for being the first person to put matrix reducibility algorithms onto a sound footing although he refers during his paper to previous ones in the same field which had made semi-formulated attempts at the problem. Harary (1959b, 1960, 1962a) establishes some results interpreting matrix reducibility in graph theoretic terms. In particular he proves the equivalence of strong components of a digraph and irreducible blocks of its adjacency matrix (see section on graph theory in chapter one for a definition of these and other graph theory terms used in this chapter). He uses this result in developing his algorithm on matrix reduction which is described in detail in his paper of 1962. This algorithm has two major defects. The first is that he appears to be unaware of the significance of first ordering non-zeros onto the diagonal (which means he cannot decompose matrices which are bireducible but irreducible) and, in addition, his method involves obtaining the invariant Boolean power of the matrix (an expensive process when $n$ is large) before continuing to the decomposition stage. Since the algorithms given in section 4,

which deal with the two defects mentioned above, have a similar interpretation to that of Harary's method, there is no great advantage to be gained by going into his algorithm in great detail here. Suffice it to say that Harary's algorithm is based on the concept of obtaining the strongly connected components of the digraph via the reachability matrix R. This matrix is in fact the invariant Boolean power of A and has the property that $r_{ij} = 1$ if and only if i and j are connected in the graph of the matrix. Dulmage and Mendelsohn (1962) also advocate the use of his algorithm.

Duff (1970, p.68) stated that Steward's approach to the problem (Steward (1965)) was faster than that of Harary because of the time involved in calculating the invariant power of a matrix of large order. This comparison is quantified in section 5 of this chapter. The basis for the method of Steward lies in the fact that any two points in the same strong component of the digraph always lie on a loop in the digraph and vice-versa. Thus, the algorithm of Steward is essentially one of loop-chasing and collapsing, a process very similar to but less systematic than the second method given in section 4 of this chapter. The method consists in first examining the matrix for any singleton rows which are ordered in the sequence in which they are found and ignoring them (and their corresponding columns) in the future steps of the algorithm. Once no more singletons can be found, the method proceeds by looking for a non-zero off-diagonal in any uneliminated row and proceeding from that row to the row indexed by the column of this non-zero. In this way, paths are found linking rows in the matrix until a row is encountered for the second time. This row and the rows in the path between the two occurrences of this row are in the same component of the digraph and so are collapsed together into one row, hereafter called a composite row. The rows which were collapsed into this row are then removed from future consideration. This condensation

is executed by considering the rows as a Boolean pattern of Os and 1s and performing a 'logical OR' operation on the two rows. A similar operation is done on the columns and the reduced matrix is then examined for singleton rows. This process continues until all the rows or composite rows have been selected as singletons (clearly possible, since, if no singletons exist in a reduced matrix, a path of the type mentioned above must occur). If the rows and columns are then ordered so that the rows or composite rows (and columns) are in the order in which they emerged as singletons and the rows collapsed into a composite row are ordered contiguously, then the matrix will have been ordered to be block lower triangular. A flow diagram for this algorithm is given in Duff (1970, p.68).

The paper of Weil and Kettler (1969), mentioned in chapter two, uses this partitioning algorithm of Steward. Some limitations of this method of Steward are mentioned in the next section.

Sargent and Westerberg's (1964) method is an improvement over Steward's method whereby there is no investigation for singleton rows at each pass of the algorithm. Their method follows a path in the graph until a loop is found (which is then collapsed) or until the path terminates implying that a singleton has been found which can then be eliminated. After this condensation or elimination, the process continues with the last uneliminated point of the loop-chasing path. A program for this method is given in the appendices and it is compared with Steward's method and method 2 in section 5 of this chapter.

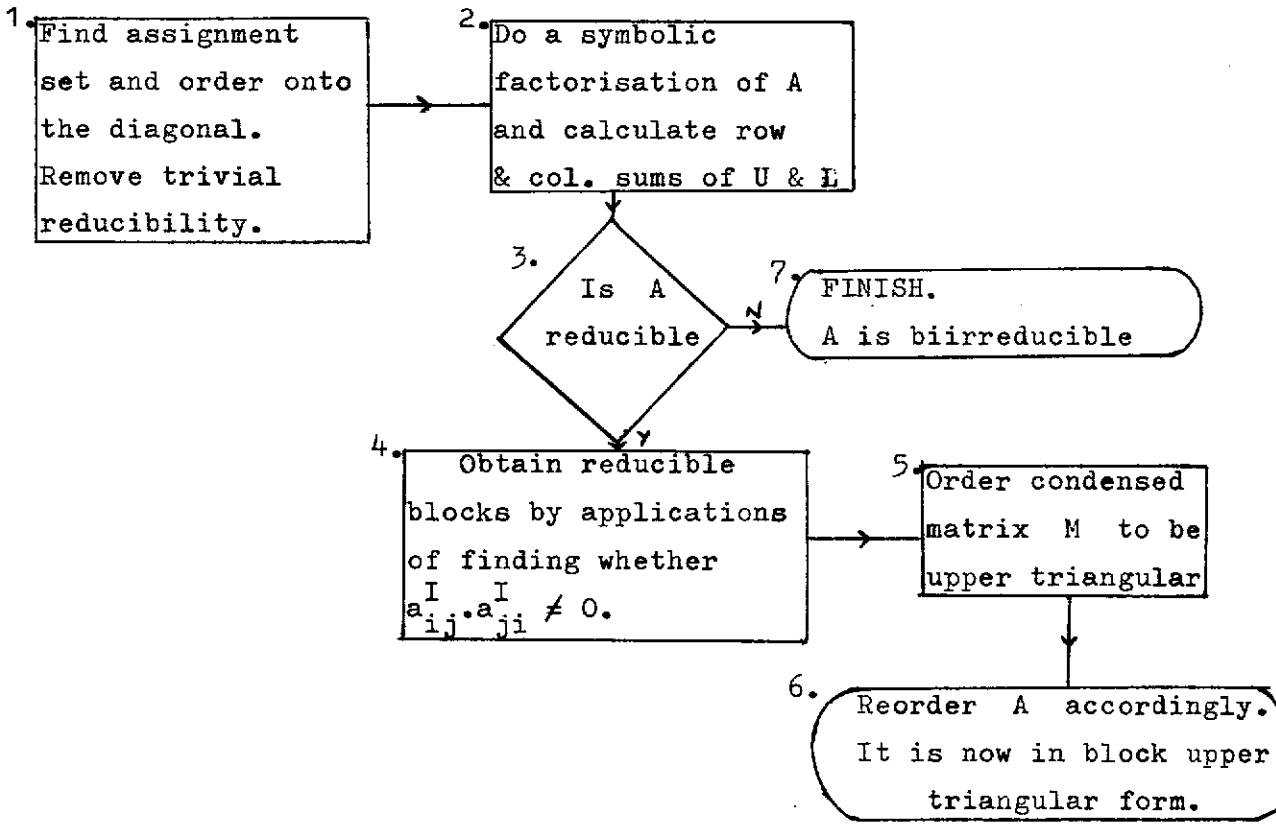
In one of the most recent algorithms to be published on matrix reducibility (Willoughby (1972)), Willoughby again uses the concept of directed graphs in obtaining his condition for two columns to be in the same block of the block triangular form. In step 4 of his algorithm given on page 57, he uses the criterion that $a^{I}_{ij}.a^{I}_{ji} \neq 0$

where $a^{I}_{ij}$ is the (i,j)th element in the Boolean inverse of A (see the section on graph theory in chapter one). This is identical to requiring that i and j be joined to each other by directed paths in the graph of the matrix (for a proof of this, see chapter one). His implementation is carried out entirely in the sparsity context of A reordered to have non-zeros on its diagonal and its LU decomposition obtained by Gaussian elimination pivoting in the natural order down the diagonal. In steps 2 and 3 of his algorithm he makes use of the following theorem on reducibility.

Theorem 3.1 (Willoughby (1970).

The elements of $A^{-1}$ are all non-zero (i.e. A is irreducible), where $a_{ii} \neq 0$, if and only if L and $U^{T}$ each have their last column as their unique singleton column.

Algorithm of Ralph A. Willoughby for matrix reducibility.



The column counts are done at the same time as the symbolic factorisation of A. In step 4, the calculation of whether or not two indices lie in the same block can be automated fairly easily. For example, if it is desired to find out which indices lie in the same block as k, say, then this is done in the following manner.

Solve $Ax = e_k$ for the kth column of $A^{-1}$

by the equations

$$Ly = e_k$$

$$Ux = y$$

Solve $z^T A = e_k^T$ for the kth row of $A^{-1}$

by the equations

$$w^T U = e_k^T$$

$$z^T L = w^T$$

then

any index i for which $x_i$ and $z_i$ are simultaneously non-zero lies in the same block as k. These rows and columns can be omitted in subsequent passes through step 4.

Although Willoughby approaches matrix reducibility from a somewhat novel angle, he has to date published no experimental results or detailed flow charts of his algorithm. Unfortunately, it is geared very much to the type of symbolic factorisation envisaged in the IBM symbolic Crout reduction (see Gustavson et al (1970)) and is not really at all feasible unless a good symbolic factorisation routine is in operation. In addition, his method of first doing a symbolic elimination without knowledge of the block triangular structure is clearly going to be very wasteful of storage because of the introduction of unnecessary fill-in as mentioned in the previous section of this chapter. Also, his method does not avoid any of the indexing problems of other methods. Because of these difficulties, this method has

not been compared in section 5 of this chapter.

Tewarson (1967c) has indicated a method whereby reduction to nearly upper triangular form can be expressed as a linear programming problem. His algorithm, in fact, does yield the block structure of the previous methods but is an unnecessarily complicated way of doing it. It does admittedly do more than just reduce a system to block triangular form but it would seem more advantageous to use one of the methods of this section to reorder the matrix to block triangular form first and then, if desired, use Tewarson's algorithm to make each block as nearly triangular as possible. This possibility is mentioned again in chapter eight on eigenvalues and eigenvectors of sparse matrices.

In the following section, new algorithms are developed from the same concepts as the original one of Harary and are compared with Harary's algorithm and the algorithms of Steward and Sargent and Westerberg in section 5 of this chapter.

## Section 4. New and modified methods for permutation to block triangular form.

Of the two algorithms in this section, one is based on the earlier mentioned one of Harary (page 54) while the other lies somewhere between the two algorithms of Harary and Steward and is described after the unified description of the partitioning methods given in the first part of this section. It is thus perhaps useful to first define the frame of reference and state a few principal theorems which are exploited by the various algorithms under consideration. Most of the theorems have already appeared in the literature although they have not always been explicitly stated. The general theory of Boolean powers of (0,1) matrices is given in Rosenblaat (1957), and some work has been done to integrate the theory of directed graphs into abstract algebra by Wenke (1964), Jaeger and Wenke (1969), and Eufinger (1970, 1971) However, most of the theorems appearing in this section can be attributed to Harary (1959a, 1959b, 1960, 1962a, 1971).

In the following discussion all matrices are considered to be Boolean matrices whose only entries are either 0 or 1. All matrix multiplications are Boolean multiplications where the Boolean multiplication of (0,1) matrices is defined in a similar manner to ordinary matrix multiplication with the exception that 1 + 1 is defined to be equal to 1.

For any (0,1) matrix of order n, a directed graph on n points is associated with it as in section 3 of chapter one where self-loops (i,i) are allowed whenever the (i,i)th element of the matrix equals 1.

The square (using Boolean multiplication) of a (0,1) matrix A, say C, is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} a_{kj}$$

where the above-mentioned 'logical OR' rule is used in the summation.

Thus, $c_{ij} = 1$ if and only if there is a subscript $k$ such that $a_{ik} = 1$ and $a_{kj} = 1$, that is if lines from $i$ to $k$ and from $k$ to $j$ are present in the directed graph. This discussion of Boolean multiplication in terms of graphs is generalised in the following theorems.

Theorem 4.1

If $A$ is an arbitrary $(0,1)$ matrix of order $n$, and if $A^k$ is defined by

$$A^k = A \otimes A^{k-1}$$

where $\otimes$ denotes Boolean matrix multiplication, then there exists a directed path of length $k$ from the point $i$ to the point $j$ in the directed graph associated with the matrix $A$ if and only if the $(i,j)$th element of $A^k$ is equal to 1.

Proof:

The proof is essentially the same as that in Harary (1959a) and is very simple. Therefore it will not be reproduced here. It should perhaps be noted that self-loops are included in the above-mentioned directed paths so that, for example, the directed path formed by the directed lines $(i_1,i_1)$, $(i_1,i_2)$, $(i_2,i_3)$, $(i_3,i_3)$, $(i_3,i_4)$ would be considered a directed path of length 5 from the point $i_1$ to the point $i_4$.

The first two corollaries follow immediately.

Corollary 1.

If $B$ is a $(0,1)$ matrix of order $n$ which has 1's on the diagonal, then, if $B^k$ is defined by

$$B^{k+1} = B \otimes B^k,$$

there exists a directed path of length $k$ or less from point $i$ to point $j$ in the directed graph associated with the matrix $B$ if and only if the $(i,j)$th element of $B^k$ is equal to 1.

Proof.

The proof is obvious if it is observed that for any $(0,1)$ matrix $A$,

$$(A + I)^n = I + A + A^2 + \cdots + A^n$$

where Boolean arithmetic is used throughout.

Corollary 2.

If $C$ is a $(0,1)$ matrix of order $n$ which has zeros on the diagonal, then, if $C^k$ is defined by

$$C^{k+1} = C \otimes C^k$$

there exists a directed path of length exactly $k$ from the point $i$ to the point $j$ in the directed graph associated with $C$ if and only if the $(i,j)$th element of $C^k$ is equal to 1.

Proof.

The proof is again trivial. The paths in the above theorem do not include self-loops, so the path mentioned after theorem 4.1 would have length 3 in the context of this Corollary.

The additional two Corollaries are required in the theoretical justification for method 2.

Corollary 3.

If $B$ and $B^k$ are defined as in Corollary 1 and the elements of a $1 \times n$ $(0,1)$ array called $DIAG_k$ are defined by

$$DIAG_k(i) = \sum_{\substack{l=1 \\ l \neq i}}^{n} b_{il} b^k_{li}$$ where the summation is Boolean,

then,

if the element $DIAG_{k-1}(i) = 0$ but $DIAG_k(i) = 1$, then the point $i$ lies in a cycle of length $k$ but not in any cycles of length less than $k$.

Proof.

The proof is again straightforward and follows from Corollary 2 since the above summation means effectively that the diagonal elements of $B$ and $B^k$ have been considered zero. These cycles do not include self-loops.

Clearly, any two points in the same cycle will lie in the same strong component of the directed graph and any point being in a genuine cycle (i.e. no self-loops) will imply that some other point also lies in a cycle (i.e. it is impossible that $DIAG_k(i) = 1$ for only one $i$) and, in particular, that this point is not a strong component by itself. The next Corollary states how the other points in the cycle or cycles may be determined.

Corollary 4.

If the point $i$ lies in a cycle of length $k$, say, then the other points in this cycle can be found by looking at the $i$th row and column of $B^t$ (for any $t \geq k-1$). Those $j$ for which $b^t_{ij}$ and $b^t_{ji}$ are simultaneously 1 and for which $DIAG_{k-1}(j) = 0$, $DIAG_k(j) = 1$ are the required points. Points $j$ which satisfy the first condition but not the second also lie on cycles from $i$ and so are also in the same directed component in the digraph.

Proof.

The proof is again self-evident.

These theorems and corollaries establish a unifying platform for all the partitioning algorithms considered here and form a basis from which method 2 of this section is developed. It is seen that, to a greater or lesser extent, all the algorithms do loop chasing

and strong component finding within the graphical framework just described. The four methods already described are now very briefly examined in this light.

The reachability matrix used by Harary is the invariant power of the matrix B of Corollary 1. Thus, all the paths between points are given by this matrix and two points i and j lie in the same loop (and hence same strong component) if and only if the corresponding entries ((i,j) and (j,i)) of the reachability matrix are 1. Since the sparsity structure of the reachability matrix is the same as that of the Boolean inverse of a matrix (see the graph theory section of chapter one), Willoughby's method admits of an identical description in this graph theoretic sense. Steward's method involves a haphazard chasing of loops in the graph. These loops are of an undetermined length (determined solely by the structure of the matrix) but certainly lie entirely in the one strong component. The collapsing procedure merely identifies those points in the same component as each other, and the final matrix obtained from selecting singleton rows or composite rows as they occur is merely a permutation to lower triangular form of the matrix of Harary's condensed graph. The description of the Sargent and Westerberg algorithm is similar to that of Steward's.

On the one hand, the first two methods develop all the paths immediately (Willoughby avoids this a little by only evaluating selected columns of the inverse of A), while the last two are unsystematic in their approach. Method 2 is developed to bridge the gap between these two extremes being systematic in only looking for loops of lengths in successive powers of two and not forming an invariant power until the matrix has been very much reduced in size.

The first step in any of the algorithms discussed in this section is to use a method of the previous chapter (preferably method 4) to permute the rows and columns of the matrix in order to put non-zeros

on its diagonal. The importance of this has been mentioned already and cannot be overstressed since it has the effect of permitting the consideration of only symmetric permutations. The restriction to symmetric permutations has indeed no effect on the ability of the algorithms to decompose the matrix since the previous discussions have shown there to be effectively no such a thing as irreducible, birreducible matrices. In the description of the two algorithms that follows, it is assumed this initial permutation has already been performed.

Method 1.

(1) As in the method of Harary (1962a), the invariant power $B$ of the Boolean representation $A$ of an arbitrary complex matrix $M$ (or a permutation of $M$ which has non-zeros on the diagonal) is first calculated.

(2) The ordering $PAP^T$ of the rows and columns of $A$ to put it into block upper triangular form is given by that ordering $P$ which orders the rows of $B$ in order of decreasing row count.

Justification of method 1.

The invariant binary power $B$ of a matrix $A$, whose graph has a point $i$ connected by a directed path to any other point $j$, will have $b_{ij} = 1$. Hence, for any $i$ and $j$ in the same strong component of the digraph $b_{ij} = b_{ji} = 1$. Further, if a strong component $I$ is connected to a strong component $J$, then all the points of $I$ are connected to all the points of $J$. Thus, if $P$ reorders $A$ to block upper triangular form it will reorder $B$ to the form given in figure 4.1.

$$\begin{pmatrix} M & M & M & M \\ & M & M & M \\ & & M & M \\ & & & M \end{pmatrix}$$

figure 4.1.

where M is a matrix all of whose entries are 1. Hence step 2 is justified. The justification of step 1 follows from the theorems given previously and is given in Harary (1962a).

The only case where this method could possibly fail to give a block triangular ordering would be that having two equal sized diagonal blocks which were totally disconnected from the rest of the matrix. However, it is normally possible to tell from the problem from which the matrix arises as to whether or not it can be decomposed to block diagonal form and subsequent care taken when performing this algorithm. If it is really desired to reduce the matrix to block diagonal form then a method similar to Tewarson's (1967c) might be used first. Method 1 could then be used on each diagonal block in turn.

The advantage of this method over that of Harary is very evident since the execution of step (2) is obviously faster than forming the elementwise product of the reachability matrix with itself and reading off the irreducible blocks from that product. This is borne out in the results of section 5. The procedure for forming the invariant power in step (1) is given in the appendices.

Unfortunately, method 1 leaves untouched perhaps the biggest defect of Harary's method since it still involves the calculation of the reachability matrix, the most time-comsuming step in either algorithm. Method 2, which is now described in detail, is an attempt to avoid this difficulty often requiring only one matrix multiplication which is done with a matrix of a very much smaller order than the initial one.

Method 2.

The algorithm is first given in step form below after which a justification follows. A flow diagram for a very slightly modified version of the algorithm is given in section 5.

(1) Let A be the Boolean representation of an arbitrary complex

matrix M permuted so that its diagonal is non-zero. Put $k = 1$.

(2) Calculate $A_{k+1} = A_k^2$ using Boolean multiplication.

If $A_{k+1} = A_k$ go to (4)

Calculate DIAG(i) from the formula

$$\text{DIAG}(i) = \sum_{\substack{l=1 \\ l \neq i}}^{n} (a_{k+1})_{il}(a_{k+1})_{li}$$

If there exists an i such that DIAG(i) is non-zero, go to (3) otherwise increase k by 1 and go to the beginning of step (2)

(3) Examine those rows and columns (index i) of $A_{k+1}$ for which DIAG(i) is non-zero. For each such i, find the subset $J_i$ such that, for each $j \in J_i$, $(a_{k+1})_{ij}$ and $(a_{k+1})_{ji}$ are simultaneously non-zero. Collapse the rows and columns of A by the following procedure. Form the logical sum of the rows i and those in the subset $J_i$. For each i, use this sum to replace row i in the original matrix and delete those other rows in $J_i$. Do the same procedure on the columns of A. A note is made of those rows and columns collapsed together, and the algorithm continues to step (2) with A now equal to this collapsed A and k again put equal to 1.

(4) The matrix has now been fully reduced and can be ordered by a permutation Q to triangular form.

If the original matrix M is then reordered (first to bring its non-zeros onto the diagonal) such that the rows and columns collapsed together in the reduction process of step (3) (note was made of them in (3)) are grouped together, the groups being in the same order as in A and the permutation Q applied to this matrix, then the total resulting permutation will render M block triangular.

Justification of the algorithm.

Step (2). From the meaning of $(a_{k+1})_{ij}$ (Corollaries 1 and 3),

it is seen that the appearance of a non-zero in position i in DIAG indicates that the point (or composite point) i lies in at least one cycle of length between $2^k$ and $2^{k+1}$ (if the cycle was of length less than $2^k$, it would have been found at the previous step and the algorithm would have gone to step (3) and collapsed this cycle). Corollary 4 is then used to find the other points of the cycle as at the beginning of step (3).

Step (3). The collapsing merely involves an identification of points found to be in the same strong component and effectively mirrors the transitivity property of directed paths. Any point (or composite point) outside the group which is being collapsed and which is linked originally to any point (composite point) of the group will be similarly linked to the point in the reduced matrix corresponding to the collapsed group.

Step (4). The diagonal elements of the final A represent the strong components (or equivalently the diagonal blocks - Harary (1960)) while each of the off-diagonal non-zeros represents a line or lines from points of one component to points of another. Clearly this matrix is acyclic otherwise lines would exist between two strong components giving a contradiction. Hence, by Harary (1959b) or by observing the graph of the matrix to be a tree and using Parter (1961), the matrix can be ordered to be triangular by means of symmetric permutations. More details concerning this are to be found in the next section.

Basically, as mentioned earlier, this method is situated computationally midway between the algorithms of Harary and Steward. It avoids the excessive number of multiplications in Harary's algorithm while only doing a very minimum amount of Steward type loop-chasing. In fact, only loops of length 2 are traced at any one time. This systematisation of Steward avoids some of the pitfalls of his method, one of which is readily demonstrated by considering the

example of figure 4.2.

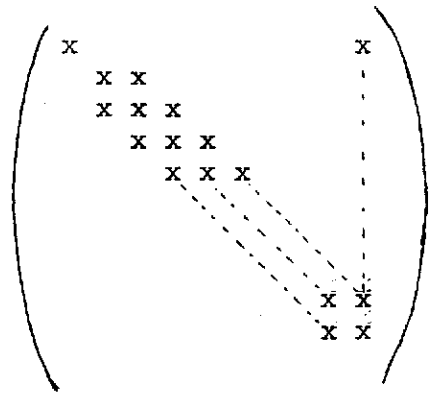


figure 4.2

Since the matrix of figure 4.2 has no singleton rows, the Steward algorithm will start loop-chasing with the element (1,n) eventually finding that rows 2 and 3 are in the same loop. The next stage ascertains that composite row (2 & 3) and row 4 are in the same loop and the process continues until the matrix of figure 4.3 is obtained.

$$\begin{pmatrix} x & x \\ 0 & x \end{pmatrix}$$

figure 4.3

Here clearly there is a singleton row and so the Steward algorithm now quickly terminates. However, the amount of work (principally in terms of comparisons) in the decomposition of this matrix by means of Steward's method is seen to be

$$\tfrac{1}{2}n^2 + O(n)$$

whereas the matrix is easily decomposed using the modification of method 2 given in section 5 since rows 2 to n are joined successively by paths of length 1 and are hence all collapsed together in one pass of the COLLAPSE procedure. It is seen that method 2 will require only

$$O(n) \quad \text{operations}$$

which is of the same order as Sargent and Westerberg's method.

Evidently because of the path of length n-1 from point 1 to point 2, log n multiplications will be needed to form the invariant Boolean power giving a total of

$$O(n^3 \log n) \quad \text{operations}$$

and, because of the method by which Boolean multiplication is carried out (see, for example, the procedure INVAR in the appendices), this will be reduced to

$$O(n^2 \log n) \quad \text{operations}$$

for the method of Harary or method 1 of this section.

Thus, it is true that while Steward's algorithm may work efficiently for some matrices whose graphs have many long paths and few short ones, while the method of Harary works well if no long paths are present (the example of figure 4.2 contradicts both of these premises), method 2 and the method of Sargent and Westerberg should be able to cope better when dealing with matrices which do not have such characteristics. The results in section 5 show that Steward's method only has an advantage over method 2 (albeit a significant one) when the matrix is very sparse and apparently unstructured. Structured or dense matrices (dense being used in a comparative sense here... the probability density of a 'dense' matrix being about 0.1 (or 10 zeros per row in a matrix of order 100)) will tend to have shorter paths joining points in the graph of the matrix and so method 2 and Sargent and Westerberg's method should triumph. This is indeed borne out by the results of section 5.

## Section 5. Computational details, results, and conclusions.

In this section, the algorithms to be tested are first described, the results are given in tables, then some conclusions are drawn. The four methods being compared are Harary's method (page 54), Steward's algorithm (page 55), Sargent and Westerberg's method (page 56), and methods 1 and 2 of the previous section.

For the two methods (Harary's and method 1) which rely heavily on first obtaining the invariant Boolean power of a matrix great care should be exercised in how this computation is performed. It is by far the longest part of these algorithms, even if good Boolean power procedures are used, and will consume vast amounts of time if done inefficiently. Four methods for obtaining this invariant power were tried. These were as follows:

Method A: Ordinary Boolean multiplications using two N x N arrays forming the powers $A^2$, $A^4$, $A^8$ until invariance is achieved, observed by finding k such that $A^{2^k} = A^{2^{k-1}}$.

Method B: Again ordinary Boolean multiplications are used but this time a second N x N array is not stored. A one-dimensional array keeps a note of elements changed from 0 to 1. This array is used to update A before the next multiplication. This procedure terminates when the one-dimensional array is left empty after a multiplication.

Method C: A procedure taken straight from one developed by Baker (1962) which uses only the original two-dimensional array. This procedure searches rows for non-zeros and performs a 'logical OR' operation with the rows corresponding to the row and column of this non-zero, replacing this row by the result. The method proceeds in this way sweeping through the matrix until invariance is obtained.

Method D: A procedure developed by the author but subsequently found to be described in a paper by Comstock (1964). This procedure is essentially identical to ordinary Boolean multiplication and consists of sweeps through the matrix looking for loops of length 2 until

invariance is obtained. It has the advantage of requiring only one copy of the array and less multiplications than method A or B.

The four methods were tried on a number of random matrices and the average time taken for each method on the KDF9 is shown in table 5.1.

| METHOD A. | METHOD B. | METHOD C. | METHOD D. |
|---|---|---|---|
| 8min. 17secs. | 9min. 55secs. | 7min. 43secs. | 3min. 10secs. |

Table 5.1. KDF9 times of invariant binary power methods. Average times of runs on 10 100×100 random matrices of density 0.03.

As is evident, method D proves to be the quickest. This method was then incorporated into procedure INVAR (see appendices) and was used in the appropriate part of Harary's method and method 1. Harary's method is described in his paper (1962a), while the modification of it used in method 1 is described on page 65.

Steward's method is described on page 56 and is not given in any further detail since it is quite similar to the method of Sargent and Westerberg. A print out of the program for this method of Sargent and Westerberg is given in the appendices.

Method 2 is modified a little from the description given on page 66 and a flow chart of this method is given on page 74. The principal changes made in the algorithm have been to allow collapsing and searching before any Boolean multiplications are carried out and to allow more than one collapsing operation to occur between each Boolean multiplication. This considerably reduces the order of matrices on which the Boolean multiplications are performed. Figure 5.1 illustrates two aspects of the algorithm, the one is the nature of the collapsing process itself and the other is the above mentioned sequence of collapsing operations causing the order of the matrix to be reduced.

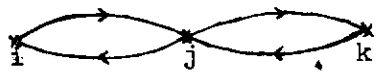



figure 5.1 (a)

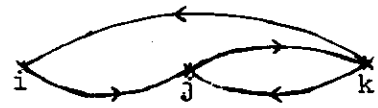
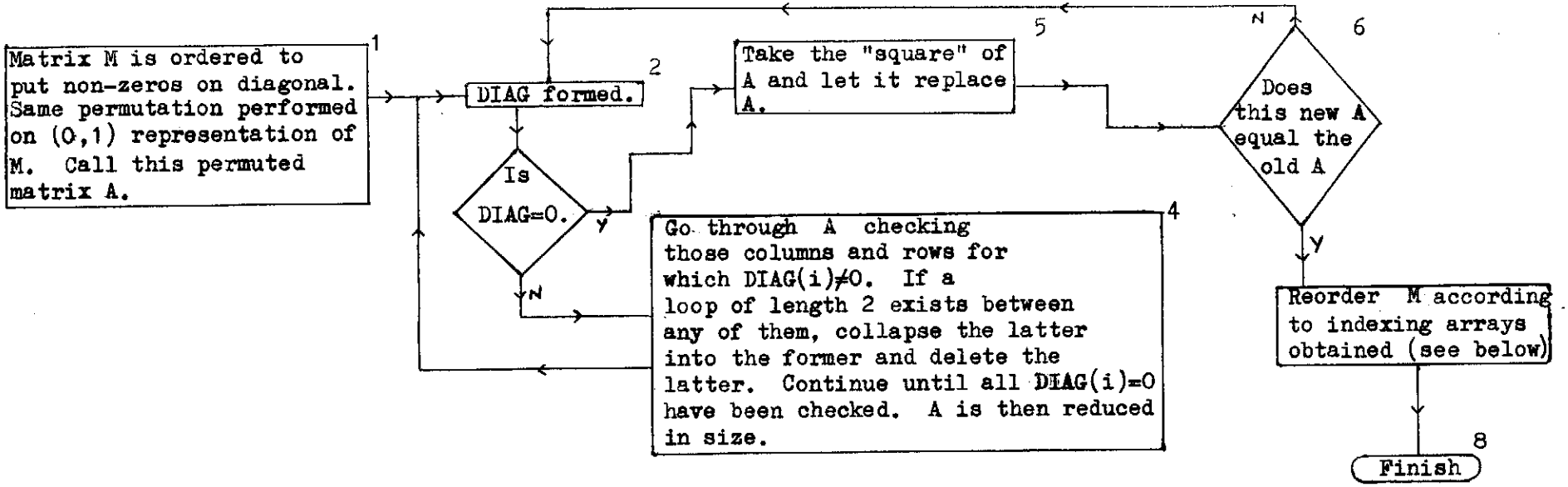



figure 5.1 (b)

In both figures, assume the ordering of the points is such that $i \prec j \prec k$. Then, in figure 5.1 (a), j is first collapsed into i giving lines (i,k), (k,i) [from lines (j,k) & (k,j)] which causes k to be subsequently collapsed into i at a later point during the same call of the collapsing procedure. The example given at the end of the last section illustrates this very case. In figure 5.1 (b), the first pass through the collapsing procedure collapses k into j and gives a new line (j,i) [from (k,i)], hence on the next pass (even without an intervening multiplication) the composite point (k/j) will be collapsed into the point i. This collapsing procedure is given in the appendices.

The Boolean multiplications, when required, are performed using the procedure BINMULT in the appendices. It is seen that this procedure does not actually produce the square of the Boolean matrix input to it but does a little more besides. This is all right since the matrix produced is still a matrix which has 1's only in positions where 1's exist in the reachability matrix of the graph of the original matrix. The procedure BINMULT is, in fact, one step of procedure INVAR and again benefits by having to carry no subsidiary arrays.




Note on indexing arrays.

Arrays used are three integer arrays S, B, and OR which are all one dimensional of order n where n is the order of the original matrix.

At any stage:

B(i) : 0 if row (column) i is a composite one in the present reduced matrix.

: j if row (column) i has been eliminated. It has been collapsed into row (column) j (which in turn may also have been collapsed into another row and column - see B(j)).

S(i) : Currently [1:m] where m is the order of the reduced matrix. The ith entry is the position of row or column i of the reduced matrix in the original one. Thus, B(S(i))=0.

OR(i) : The row (column) number of the row and column in position i in the final reordered matrix.

Although the note on indexing arrays gives some details of the algorithm, some additional comments are now made.

In step 7 the required permutation is obtained by using a theorem of Parter (1961) and Harary (1959b), possible because it is observed that the graph of this final A is that of a tree. If, for example, it is desired to permute it to upper triangular form then points with column count 1 are put at the beginning, points with row count 1 at the end, the matrix reduced by ignoring these rows and columns, and the process repeated until all the rows and columns have been so ordered. The simple tree algorithm of the papers mentioned above ensures that this procedure will terminate successfully with the matrix ordered to be upper triangular.

At the collapsing stage, step 4, it is possible to include a bit more loop searching into the algorithms to look for paths of the form $a_{ik_1}a_{k_1j}$, $a_{jk_2}a_{k_2i}$ etc. Experimental tests on this have not proved very fruitful and, in any case, these paths are often encountered on subsequent passes through step 4, even without an intervening Boolean multiplication.

The experimental results for the five methods, mentioned at the beginning of this section, are given in tables 5.2 and 5.3. The first table gives the average results of runs on random matrices of 3 different densities and 3 different orders. The second one gives results for the five methods on 4 structured matrices.

It is seen that the conjecture of section 4 is borne out by the results contained in these tables. Method 2 is consistently very much better than method 1 or Harary's method, while method 1 is always a slight improvement on the method of Harary. The method of Sargent and Westerberg is always better than that of Steward and both of these methods are better than the other ones if the matrix is very sparse and unstructured. Method 2 is better than Steward's method for relatively dense matrices and structured ones and is quite competitive

with the method of Sargent and Westerberg in such cases, sometimes even being an improvement on this method. The actual method to be preferred will be dependent on the structure of the matrix under consideration but the results show that the only really viable methods are those of Sargent and Westerberg and method 2 of section 4. For structured or relatively dense (say > 1%) matrices, either of these methods could be recommended for use as a block triangularisation algorithm but for general use the method due to Sargent and Westerberg is to be preferred.

Willoughby's method (page 57) was not compared since its successful implementation is dependent on good symbolic factorisation procedures and preliminary tests on his scheme proved it to be very uncompetitive.

Similar comments to those in section 5 of the previous chapter can be made concerning the use of storing the matrix in packed form. Again the significant factor will cease to be $n$, the order of the system, but will be instead $\bar{r}$, the average number of non-zeros per row of the matrix. All the algorithms will be speeded up proportionately, thus, in general, only affecting the absolute times but not the comparisons made in this section. The speed of the algorithms, even without this improvement, serves to indicate the feasibility of carrying out block triangularisation procedures on a sparse matrix system prior to any attempt at Gaussian elimination.

| MATRIX Order | 25 x 25 | 25 x 25 | 50 x 50 | 50 x 50 | 100 x 100 | 100 x 100 |
|---|---|---|---|---|---|---|
| Prob. Density | 0.20 | 0.30 | 0.10 | 0.14 | 0.08 | 0.10 |
| HARARY | 0.901 | 1.210 | 2.251 | 2.588 | 7.235 | 7.387 |
| METHOD 1. | 0.798 | 1.091 | 2.109 | 2.432 | 6.987 | 7.049 |
| METHOD 2. | 0.077 | 0.070 | 0.439 | 0.247 | 1.166 | 0.989 |
| STEWARD | 0.094 | 0.096 | 0.3580 | 0.361 | 1.394 | 1.415 |
| SARGENT AND WESTERBERG. | 0.059 | 0.060 | 0.200 | 0.211 | 0.740 | 0.768 |

Table 5.2. Results on matrices of various densities and orders. Average values of runs of 10 matrices at each of the 6 values. The times are in seconds of 1906A milltime.

| MATRIX / METHOD | 54 x 54 CURTIS 1 | 57 x 57 WILLOUGHBY 1 | 100 x 100 EXAMPLE SIMILAR TO ONE IN FIGURE 4.2 | 64 x 64 EXAMPLE FROM LAPLACIAN ON CUBE OF SIDE 4. |
|---|---|---|---|---|
| HARARY | 0.565 | 2.584 | 3.738 | 0.557 |
| METHOD 1. | 0.525 | 2.539 | 3.689 | 0.510 |
| METHOD 2. | 0.224 | 0.274 | 0.799 | 0.307 |
| STEWARD | 0.437 | 0.474 | 11.731 | 0.606 |
| SARGENT AND WESTERBERG | 0.225 | 0.264 | 0.926 | 0.308 |

Table 5.3. Results of running methods on structured problems.

Times are in seconds of 1906A milltime.

# CHAPTER 4.

## SOME COMMENTS ON PARTITIONING AND TEARING.

## Section 1. Introduction.

In this chapter further developments from the block triangularisation of chapter 3 are discussed and a topic known as tearing is introduced. The concept of tearing has its basis in papers by Kron (1953, 1954) and a subsequent book by the same author (1963). Many papers have been written (for example, Spillers (1965), Branin (1959) and Harrison (1963)) to define Kron's terms and put them on a secure mathematical footing. This is because his book and original papers are written in an intuitive manner from the viewpoint of a person well-versed in circuit theory and extremely capable of associating the mathematical tearing of systems with the physical tearing of circuits.

The main interest in tearing centres on whether it is possible to select elements so that their removal enables the system to be partitioned into block triangular form. In fact, it has been shown recently by Rose and Bunch (1972) that this is essentially the only case in which tearing (or the method of modification, as they call it) provides any benefit to finding a solution to a set of linear equations. Just as algorithms have been developed to put systems into block triangular form (see chapter three) so have algorithms been developed to tear systems in an optimal or near optimal fashion. Steward is the principal exponent in this field first publishing a crude algorithm in 1965 and then improving on it in subsequent papers. His algorithm splits up the diagonal blocks by selecting as torn element(s) those element(s) whose removal would break up the most loops in the system and leave the largest number of disjoint remaining loops. The subject is a very involved one, occupying most of Steward's forthcoming Ph.D Thesis, (Steward (1972)), and so no exact detail of tearing algorithms will be given here except to note that they do exist and are becoming more

computationally feasible year by year. Sargent and Westerberg (1964) also describe a method of tearing systems. A recent development is due to Hellerman and Rarick (1972a, 1972b) who obtain the torn columns (called "spikes") by a sequence of complicated row and column counts. Their algorithm promises to be much faster than that of Steward's but its optimality has not yet been analysed. In many cases, it is of course feasible that the form of the system to be solved or the physical problem from which it arises may suggest a priori which elements should be selected as the torn ones thus avoiding some of the computations mentioned in this paragraph.

In the second section of this chapter, tearing, inasmuch as it interests sparse matrix analysts, is described in very simple terms following the type of approach adopted by Steward (1965). It is subsequently shown to be basically just a way of partitioning the matrix where one of the partitions has the property of being quite exceptionally sparse.

This approach to tearing is used when describing a factored form of the inverse in section 3 of this chapter. Some numerical properties of this algorithm are also considered in this section.

In the final section, a method of partitioning and tearing due to George (1972) is described, examined in some detail, and an improvement to it suggested. Some final comments are then made on tearing and tearing algorithms.

## Section 2. A short description of tearing.

Tearing can be thought of in two different ways. The method of elimination and the method of iteration (Steward (1962)). The former is a direct method in which a finite sequence of eliminations are performed in which the torn elements play a special role. The latter is an iterative method of tackling the problem in which successively improved values for the torn elements are obtained. It is the former approach which will be followed here, the example used being a 2 x 2 block matrix although the generalisation to more blocks follows immediately.

Consider the equations in figure 2.1.

$$\begin{pmatrix} A & C \\ D & B \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

figure 2.1

With the equations partitioned as in figure 2.1 they become

$$Ax + Cy = a \qquad \ldots\ (2.1)$$

$$Dx + By = b \qquad \ldots\ (2.2)$$

In most good tearing methods the matrix $D$ is very sparse, perhaps only containing one non-zero element. The first step in any tearing algorithm would be to solve for $t$ in the subsystem given by the equation

$$Bt = b \qquad \ldots\ (2.3)$$

Then, from equations (2.3) and (2.2), the variable $y$ can be found in terms of $t$ and a few components (since $D$ is very sparse) of the vector $x$. This is shown in the equation

$$y = t - B^{-1}Dx \qquad \ldots\ (2.4)$$

where $B$ is assumed to be non-singular. (The inverse of $B$ is

not itself calculated, of course, but merely the LU decomposition of it).

If the expression in equation (2.4) for $\underline{y}$ is substituted into equation (2.1), then equation (2.1) can be solved for the vector $\mathbf{x}$, resubstitution in equation (2.4) giving $\underline{y}$ and hence the complete solution of the system in figure 2.1.

This is the process known as elimination and is described at length by Steward (1962) in his paper on information flow. The line after equation (2.3) indicates that a necessary condition for the solution process to work is that the submatrix B is non-singular. This is by no means guaranteed for an arbitrary partitioning of the kind mentioned in figure 2.1. and some mention is made of this in the section on stability.

This description of tearing in a language akin to that of partitioning leads naturally on to the algorithm for solving equations which is described in the next section.

## Section 3. Tearing and a factored form of the inverse.

The algorithm is first described in detail on a block 2 x 2 example (see figure 2.1) and then a description of how it would be performed in the general case follows. Finally, some comments on the stability of the method are made.

In figure 2.1 the torn element or elements are assumed to be in the very sparse rectangular block D. The stages of the algorithm are then as follows:

(1) Perform forward elimination on B, performing the same elementary row operations on the right-hand side and on the rows of D.

(2) Back substitute in B, reducing B to the identity and again affecting D (making it D') and the right-hand side. Continue the back substitution, reducing C to zero and changing A to A' and altering a, the upper part of the right-hand side.

(3) Perform forward elimination on A', affecting A' and the right-hand side, pivoting on the torn columns at the end of the elimination. Back substitute in A', and continue the back substitution down the torn columns in D'. Do the same row operations to the right-hand side vector, thus finally obtaining the required solution vector.

This algorithm is now examined in detail on the example of figure 2.1 where, without loss of generality, any row permutations necessary for partial pivoting have already been performed on the matrix, and only one torn element exists and is in the last column of A and in row i of B.

After steps 1 and 2, the matrix and right-hand side have the form given in figure 3.1, where A' is identical to A except for its last column, see equation (3.3).

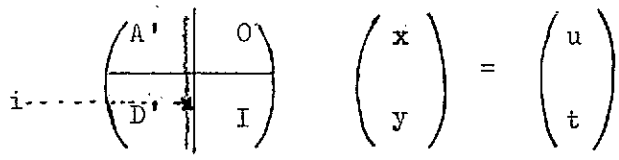



figure 3.1

**where** t is given by equation (2.3) and

$$D' = B^{-1}D \qquad \ldots\ (3.1)$$

$$u = a - Ct = a - CB^{-1}b \qquad \ldots\ (3.2)$$

and

$$A' = A - CB^{-1}D \qquad \ldots\ (3.3)$$

Figure 3.1 and subsequent equations indicate that the above-mentioned part of the algorithm is equivalent to premultiplying the matrix and right-hand side by the matrix given in figure 3.2 which is partitioned similarly to the matrix of figure 3.1.

$$\left(\begin{array}{c|c} I & -CB^{-1} \\ \hline 0 & B^{-1} \end{array}\right)$$

figure 3.2

Step (3) of the algorithm involves forward elimination on A' which produces the effect shown in figure 3.3.

$$\begin{pmatrix} U_{A'} & 0 \\ D' & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} L_{A'}^{-1}U \\ t \end{pmatrix}$$

figure 3.3

and the final back substitution gives x and y from the following equations:

$$x = U_{A'}^{-1}L_{A'}^{-1}u = A'^{-1}u \qquad \ldots\ (3.4)$$

and

$$y = t - D'U_{A'}^{-1}L_{A'}^{-1}u = t - D'A'^{-1}u \qquad \ldots\ (3.5)$$

$$= t - D'x \qquad \ldots\ (3.6)$$

but, when equation (3.6) is compared with equation (2.4), they are seen to be identical. Therefore, it only remains to show that equation (3.4) when solved for $\mathbf{x}$ is identical to solving equation (2.1) for $\mathbf{x}$ after substitution from equation (2.4) in order to show that this algorithm is identical to the partitioned description of Steward's tearing algorithm which is given on page 79.

From equations (3.3), (3.4) and (3.2)

$$(A-CB^{-1}D)\mathbf{x} = u = a-CB^{-1}b \qquad \ldots. \quad (3.7)$$

and equation (3.7) is found to be identical to substituting equation (2.4) in equation (2.1) thus proving that the algorithm mentioned on page 81 is, in fact, a tearing algorithm.

In order to discuss this factored form more closely, the example shown in figure 3.4 is now considered and a more detailed explanation of the algorithm as it works on this example is given. The shaded areas in the following figures contain non-zeros.

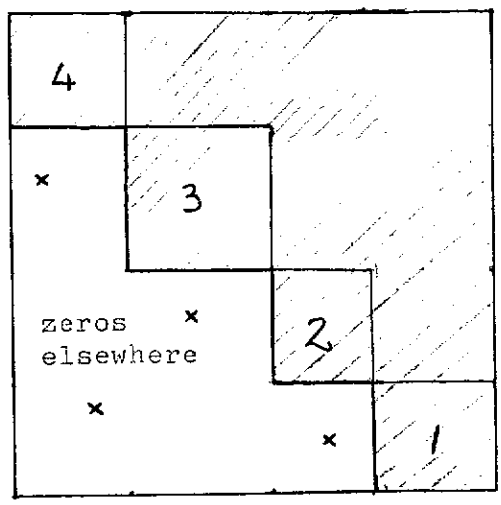



figure 3.4

Although this example has only four diagonal blocks, it is clear that the following discussion can be easily extended to cover cases with more blocks. The matrix of figure 3.4 will be

decomposed block by block starting with block 1 and ending with block 4.

After block number 2 has been reduced the matrix will look like that given in figure 3.5. The fill-in to the torn columns in this figure should be noted. A detailed explanation of what happens when block $A_3$ is decomposed is now given and serves to indicate how the method proceeds in general.

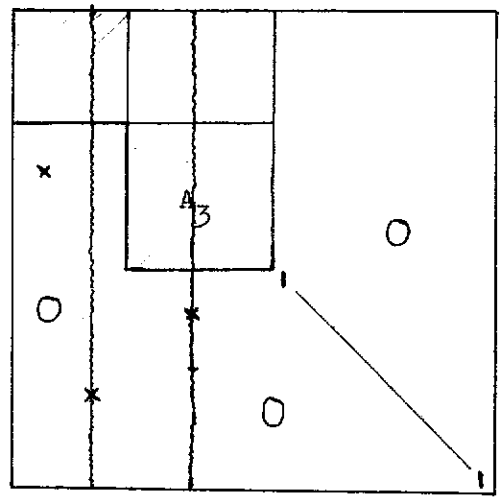



figure 3.5

Any torn columns in $A_3$ are first moved to the end of the block (or are flagged so that they will be considered last in the elimination process) and elementary row operations are performed on the whole system so that the untorn columns of $A_3$ are left in upper triangular form. Fill-in will occur in all the uneliminated torn columns and the resultant matrix is shown in figure 3.6. Any of the pivoting strategies normally used for sparse systems can be used during this elimination of the untorn columns of $A_3$. Some additional mention of this is made at the end of this section.

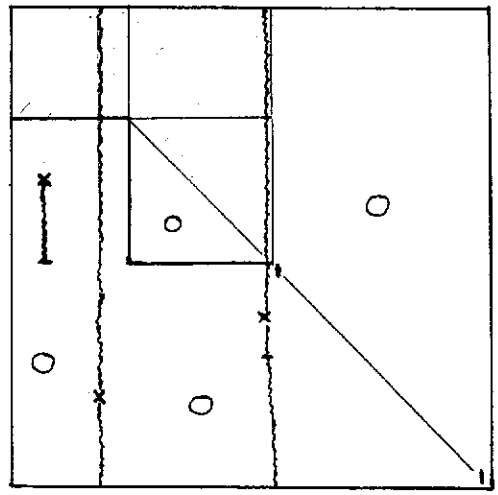


figure 3.6

Further elementary row operations are then performed to reduce elements in the torn columns of block $A_3$ beneath the diagonal to zero. This includes the part of the torn columns beneath this diagonal block. A final sequence of elementary row operations then reduces the block $A_3$ to the identity matrix and reduces the off-diagonal block above this block to zero also. All these operations cause fill-in in the torn columns yielding, at the end of this second stage, a matrix of the form given in figure 3.7.

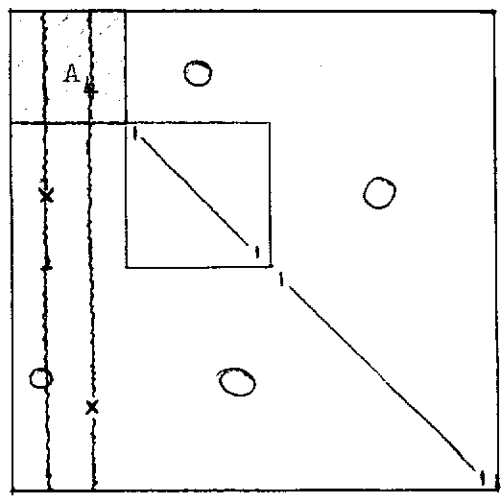



figure 3.7

The method then proceeds to step 4 where a similar elimination reduces block $A_4$ to the identity thus completing the solution of the system.

The factored forms mentioned in the title of this method are just the representations of the $2n - 1$ elementary row operations as $n$ lower triangular elementary matrices and $n - 1$ unit upper triangular elementary matrices. Only the non-trivial columns of such matrices and a pointer to their position need be held in the computer.

Some notes can now be made concerning this method and, in particular, with relation to the order in which the various operations are performed. The blocks of figure 3.4 are reduced in the order 1, 2, 3, 4, so that any fill-in occurring in the matrix is confined solely to within diagonal blocks and columns containing torn elements (which are, for even moderately good tearing algorithms, small in number). If, however, elimination is performed on the matrix of figure 3.4 (or its transpose) reducing the blocks in the opposite order, then fill-in will occur in the large shaded region above the diagonal blocks. The description of this algorithm lends itself particularly well to storing the elementary triangular matrices, representing the $2n - 1$ elementary row operations in factored form and so the normal savings of storage by using sparse storage techniques (see next chapter, section 2) can be made (in addition to that saved by ordering the blocks $1 \longrightarrow 4$). This is because the density of the factored form is more close to the density of the original matrix than that of its inverse which is normally full (see Brayton et al (1970)). The columns containing torn elements, in order to keep fill-in to a minimum, should be pivoted on last within the block in which they appear. It is, of course, feasible to leave the elimination of all the torn columns to the end but, in

addition to creating more fill-in in the torn columns themselves, it is also possible that the final block of torn elements will be singular causing a grave stability crisis for which the analysis is very difficult.

Now, when considering the stability of the algorithm, it is alarming to find a simple 2 x 2 block matrix for which this method breaks down. Consider the matrix given in figure 3.8.

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 3 | 4 |
| 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 |

figure 3.8

Clearly the determinant of this matrix is equal to 2 and so its inverse exists. However, both of the diagonal blocks are singular and so the initial LU decomposition is impossible and the method fails. Thus it can be seen that even a matrix which is well conditioned may fail to give a solution using the algorithm described. Since, by the arguments at the beginning of the chapter on block triangularisation, the solution of a block triangular system by the above method does not in itself create any induced instability, this instability must arise through the presence of the torn element or elements. By using the same diagonal symmetric scaling as in the second section of chapter 2, it is seen that the size of the non-zeros in the upper rectangular block in figure 3.8 can be made negligible at the expense of increasing the size of the torn elements correspondingly.

There is no obvious a priori method of deciding whether a particular tearing and partitioning will yield well-conditioned diagonal blocks so, when implementing the method, some monitoring strategy should be adopted to check for instability.

The type of monitoring strategy envisaged is as follows. Since the raison d'être of tearing is to reduce substantially the size of the diagonal blocks, it is feasible that full partial pivoting (or perhaps even total pivoting with the proviso that pivots are not chosen from the torn columns until the end of the elimination) could be employed within each diagonal block thus guarding against bad induced instability within these blocks. If the blocks are still so large that this is impractical and more account must be taken to preserve sparsity within these blocks, then a compromise partial pivoting with growth monitoring could be used as in the program of Curtis and Reid (1971b). The danger of induced instability through growth in the torn columns is, however, not prevented by this and some additional monitoring must be employed. It is thus necessary to keep a separate check for growth in the torn columns of the system. These columns are in some way identified at any rate since they have to be pivoted on last in the diagonal block in which they occur, and so the amount of extra work is not very large. The actual growth which could be allowed would, of course, be dependent on the accuracy of the solution required and could be set as an input parameter to any solution process adopting these procedures. A fuller discussion of this growth in the matrix elements for scaled matrices is to be found in Curtis and Reid (1971d).

Additional instability can occur in the right-hand side as illustrated in figure 3.9 and the comments following it.

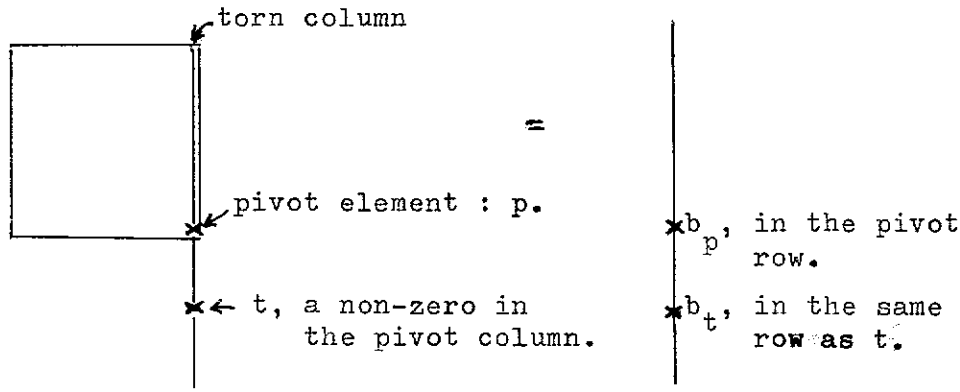



figure 3.9

Now suppose $t \gg p$, the pivot at the last stage of elimination and first stage of back substitution. Then, during the back substitution, $b_t$ is transformed as in

$$b_t \longrightarrow b_t - b_p.(t/p) \qquad \ldots\ldots (3.8)$$

This can give a fairly large growth in $b_t$ due to the presence of a large value for $t/p$ multiplying $b_p$. This will not be disastrous if the component $t$ of the solution vector is also large (it is likely to be after the computation of equation (3.8)) since the relative error is then small. Because of the possible growth from calculations like that of equation (3.8), it is important to keep a check on the size of successive right-hand side vectors. This monitoring of the right-hand side vector has to watch for growth in this vector and should print out some warning message only if this growth is followed later in the computation by a subsequent reduction in size of the same component.

These monitoring strategies while not ensuring stability, will at least give the user of these procedures some idea when his answer is likely to be inaccurate.

It should be pointed out, at this stage, that the stability difficulties envisaged in the above paragraphs are to some extent rather less likely to happen in practice than might be supposed. It is entirely possible that the original system may quite naturally split into the form of figure 3.4 where it is known a priori that the diagonal blocks are non-singular. In addition, as is often the case in tearing problems, those elements torn could well be ones which have little effect on the system (known in advance from physical considerations) and so be unlikely to cause the other type of error indicated above.

## Section 4. Towards an optimal order for Laplace and some final comments on partitioning and tearing.

George (1972) gives a fairly detailed description of an ordering which is developed for a grid on the unit square but which, as is seen later, can be viewed in terms of partitioning and tearing and has a much wider applicability. An improvement to George's ordering suggested by J.K. Reid (private communication) is also discussed at length. Finally, some additional comments are made on tearing and partitioning algorithms.

Laplace's equation is considered on a two-dimensional square grid where the function values at the boundary are unknown and the natural ordering by rows is applied. If the grid is of order $n$ ($n + 1$ points in each direction), then the matrix arising from a nine-point discretisation of the differential equation (and this is later shown to be a substage of a five-point formula) is a symmetric, positive definite, band matrix with semi-bandwidth $n + 2$ (where a matrix has semi-bandwidth $\lambda$ if $a_{ij} = 0$ for $|i-j|>\lambda$). If Cholesky factorisation is used on this matrix, the number of operations required to solve this system, $N_{Cholesky}$, and the storage required, $S_{Cholesky}$, are given by the equations

$$N_{Cholesky} = \tfrac{1}{2}n^4 + O(n^3) \qquad \ldots\ (4.1)$$

$$S_{Cholesky} = \tfrac{1}{2}n^3 + O(n^2) \qquad \ldots\ (4.2)$$

where full advantage has been taken of symmetry. Admittedly, the storage for Cholesky can be improved by order $n$ if backing store is available since only $\frac{1}{2}n^2 + O(n)$ elements need be in core at any one time. However, this does involve some minor scheduling difficulties. The ordering due to George, which is now described, successfully attempts to reduce the high power of $n$ which dominates this operation count for large $n$. This ordering is first described for a general mesh of order $2^N + 1$, and then a detailed diagramatically explained example illustrates its use for $N = 4$. Generalisation

to arbitrary mesh sizes follows easily especially when the method is viewed from a tearing angle.

Let $(i,j)$ be the point in the mesh with coordinates $i$ and $j$, so that $i,j \in [0, 2^N]$, $i,j$ integer, defines the mesh. Consider the sets $S(k)$ & $P(k)$ such that

$$S(k) = \{(i,j) : (i,j) \text{ is a mesh point and either } i \text{ or } j \text{ is a multiple of } 2^k\} \quad \ldots\ (4.3)$$

and

$$P(k) = \text{Total set of mesh points} - S(k) - \sum_{j<k} P(j) \quad \ldots\ (4.4)$$

defined for $k \in [1, N-1]$.

The sets $P(k)$ consist of $(2^{-k}N)^2$ subsets where each subset lies in a '+'. The following ordering is adopted.

Number the unknowns (points) in increasing order starting with those in $P(1)$ and then in successive $P(k)$ $(k = 2,..N-1)$ at each stage numbering for each subset in $P(k)$ first one leg of the cross, then the opposite one, and finally the line of points left. All the points have then been numbered with the exception of a principal cross and boundary points ... these remaining points are numbered by taking the points from the boundary within each square (there are four squares) in turn and finally eliminating the central vertex in a fashion similar to that for the previous $P(k)$ (see figure 4.1 (d)). Then, with this ordering, the total operation count is given by $N_{George}$ and storage requirements by $S_{George}$ (see George (1972)) where

$$N_{George} < 22.2^{3N} + 8.2^{2N}(N - 4) \quad \ldots\ (4.5)$$

$$S_{George} < 8N.2^{2N} \quad \ldots\ (4.6)$$

The following diagrams illustrate this ordering procedure on a mesh of order 17 (i.e. $N = 4$)

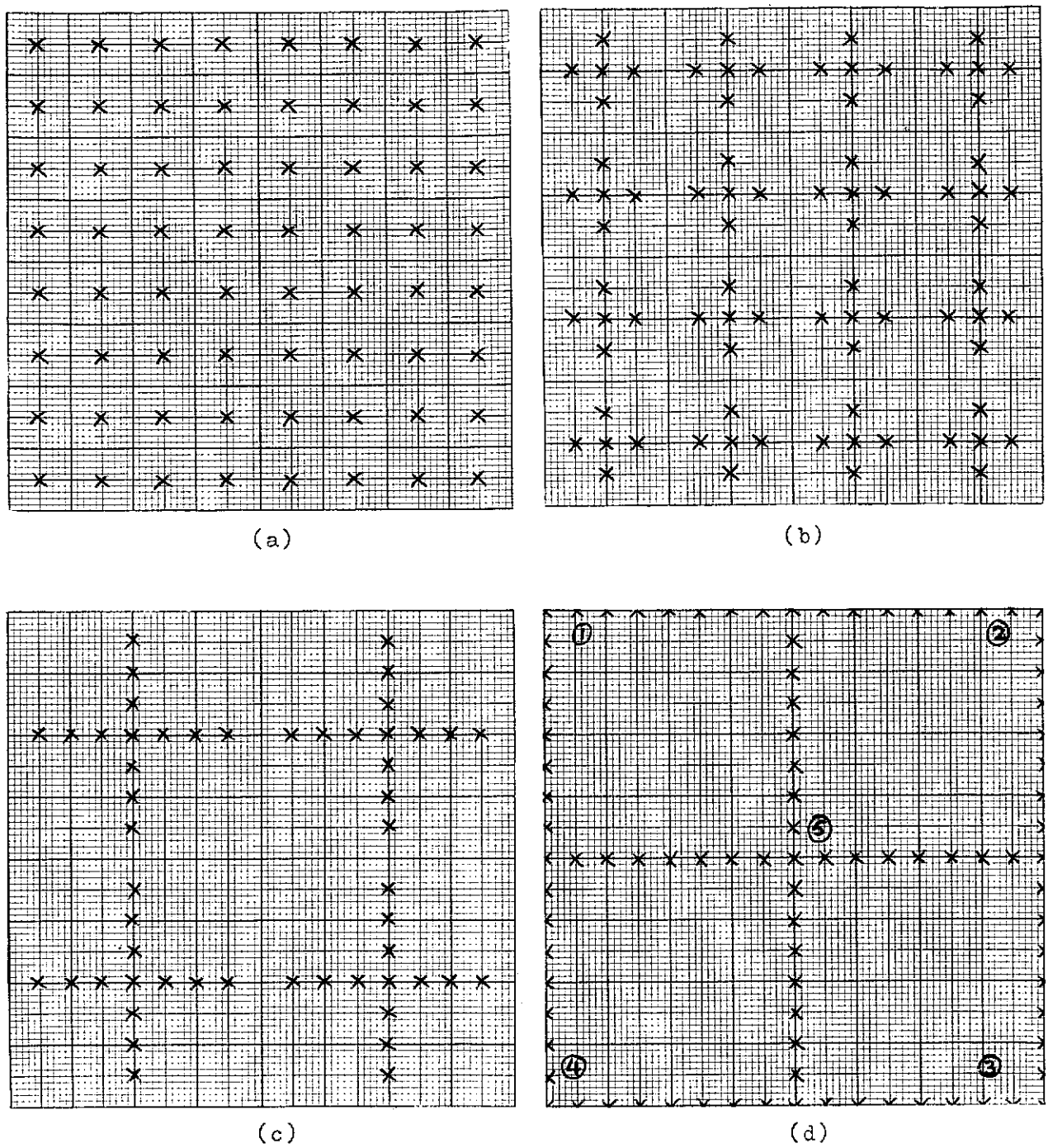



figure 4.1

It is important that the last points in d of figure 4.1 be numbered in the aforementioned manner since a numbering of the form given in figure 4.2 will give an increase in operation count and storage for that step alone equal to that given by the equations

Number of operations in stage N of George's method $\simeq 6.2^{3N} + O(2^{2N})$ .... (4.7)

Storage required for stage N of George's method $\simeq 8.2^{2N} + O(2^{N})$ .... (4.8)

Number of operations for ordering in figure 4.2 $\simeq 26.2^{3N} + O(2^{2N})$ .... (4.9)

Storage required for ordering in figure 4.2 $\simeq 159.2^{2N} + O(2^{N})$ .... (4.10)

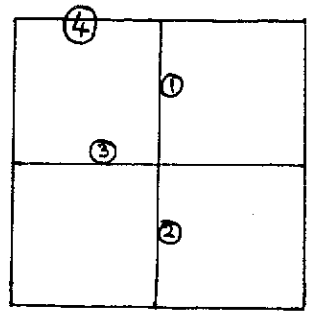



figure 4.2

It should be noted that all of the above discussion on operation counts and storage applies to when the nine-point formula is used on the grid. However, if the five point formula is considered on the Dirichlet problem on a square grid and the circled points in figure 4.3 are first eliminated, then the nine-point formula on a grid of mesh size $\sqrt{2}\times$ previous size is left (see dotted lines on figure for identification of new grid).

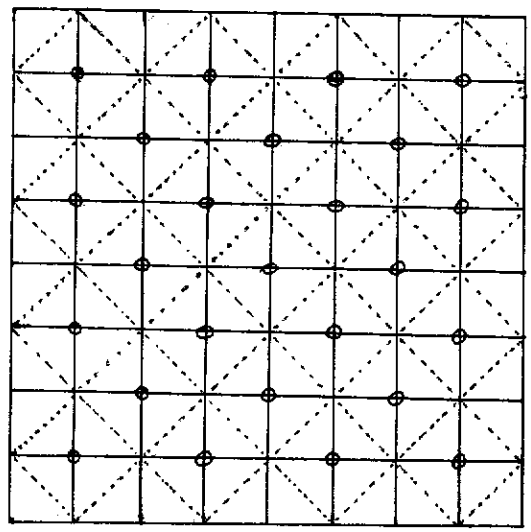


figure 4.3

This ordering eliminates $(n/2)^2 + (n/2 - 1)^2$ points immediately from a grid of order $n+1$ and is seen to be advantageous even when considering only a routine Cholesky type decomposition with a basic row ordering. The relevant operation counts are given by the equations

Number of operations using ordinary Cholesky and row ordering $= \frac{1}{2}(n+1)^4 + O(n^3)$ .... (4.11)

Number of operations by converting first to nine-point formula $= \frac{1}{4}n^4 + O(n^3)$ .... (4.12)

saving about half the number of operations.

Thus, it is seen that, without loss of generality, it is permissible to consider only the nine-point formula in the discussions on ordering for Laplace's equation.

In private discussion, J.K. Reid suggested a possible improvement to George's scheme. This is worked out in some detail below in order that a fuller understanding of the concepts involved in this ordering might be obtained and so that the extension of George's ordering to the 3-D Laplacian problem might be facilitated.

The basic building block for the operation count used in the following discussion is denoted by $O(m,n)$ where $O(m,n)$ signifies the number of operations required to eliminate $m$ strongly connected points from a network where these $m$ points are fully connected to $n$, and only $n$, other points which are not necessarily strongly connected. The basic building block for storage is concerned with how many units of store are needed to store the factored form of the decomposition of $M_1$ and $M_2$ of figure 4.4 and is denoted by $S(m,n)$. This is indicated more clearly in figure 4.4 where the submatrices $M_i$ are full.

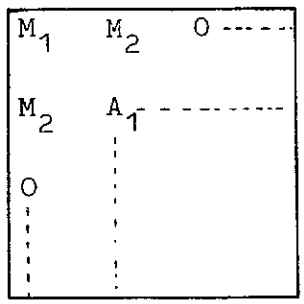



figure 4.4

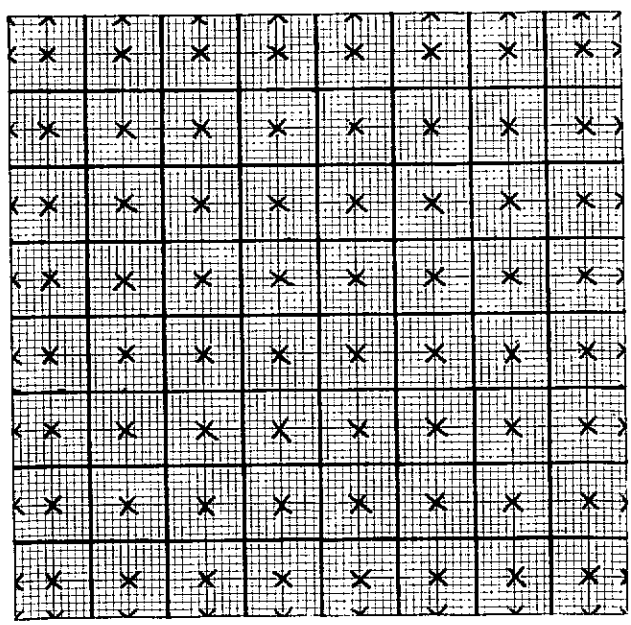
When full advantage is taken of symmetry (decomposition to $LDL^T$ being considered) it is seen that $O(m,n)$ is given by the equation

$$O(m,n) = \frac{1}{6}m(m^2 + 3m(n+1) + 3n^2 + 6n - 4) \quad \ldots\ldots (4.13)$$

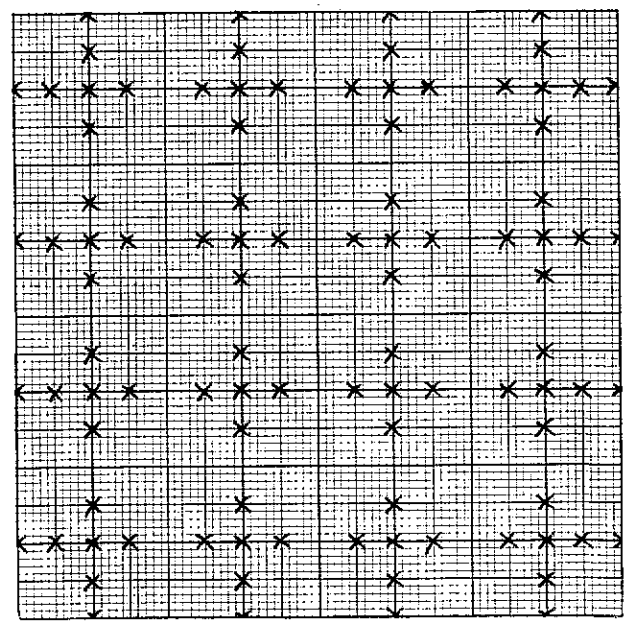
and $S(m,n)$ is given by the equation

$$S(m,n) = \tfrac{1}{2}m(m + 2n - 1) \quad \ldots\ldots (4.14)$$

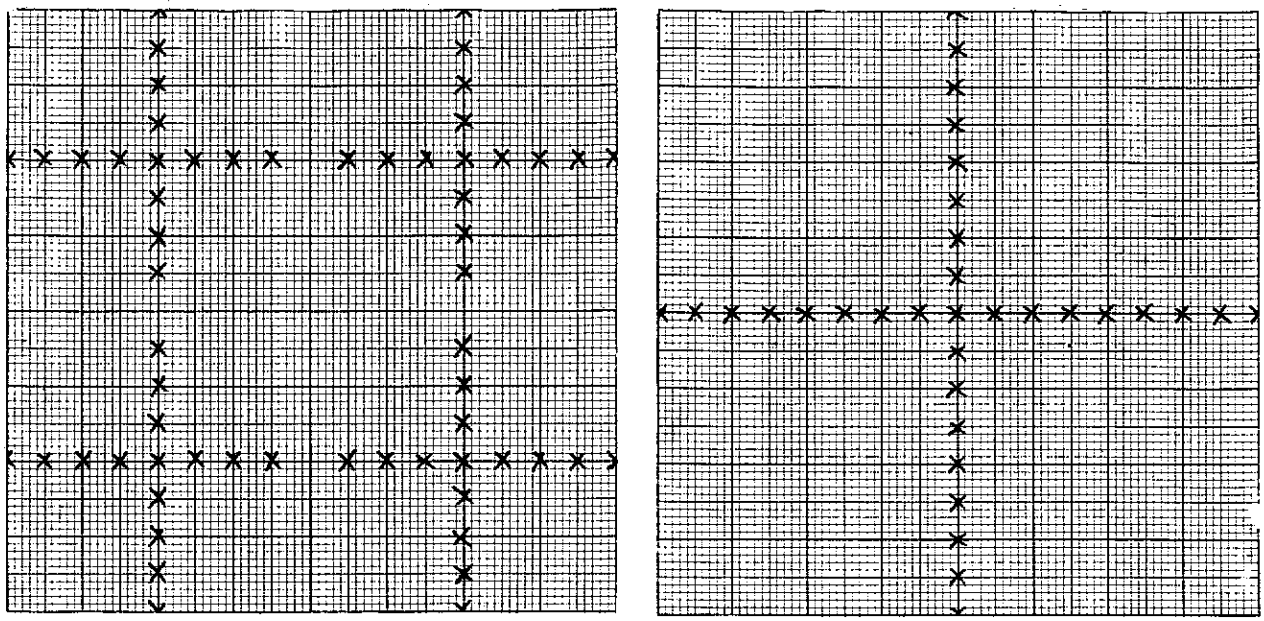
The modification to George's method consists of assuming that the boundary points are not known a priori (that is, it is not a Dirichlet problem) and recognising that these points have to be eliminated along with the other points during the elimination process. The grid of order $n+1$ is then effectively embedded in a grid of order $n+3$ and George's ordering used for a Dirichlet problem on this grid. The effect of this change in the ordering on the 17x17 example of figure 4.1 is shown in (a) to (d) of figure 4.5.



(a)



(b)



(c) (d)

figure 4.5

The operations count and storage requirements for stage k of the elimination are now examined in detail.

At stage k, there exist $(2^{N-k})^2$ independent subsets in P(k) of which 4 are what can be called corner subsets and $4.2^{N-k}$ can be called edge subsets. Clearly the counts for these three types of subset will be different while the counts within each type are the same. A typical corner subset is shown in figure 4.6

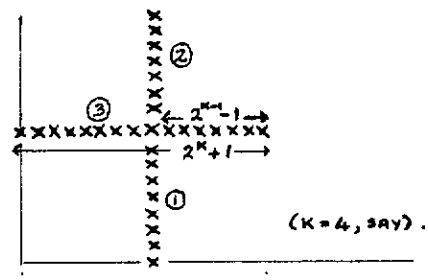



figure 4.6

Clearly the number of elements are as indicated in the diagram and so ordering in the manner indicated in the figure will give the count for corner subsets as in the equation

$$N_{corner} = O(2^{k-1}-1,\ 5.2^{k-1}+1) + O(2^{k-1},3.2^{k-1}+1) + O(2^{k},2^{k+1}+1) \quad \ldots\ (4.15)$$

In this case, and in each subsequent case, the storage count is given by the same expression with S substituted for O.

The edge subsets are shown typically by figure 4.7 and with the ordering given there, the count given in the equation

$$N_{edge} = O(2^{k-1}-1,3.2^{k}) + O(2^{k-1},2^{k+1}+1) + O(2^{k}-1,3.2^{k}+1) \quad \ldots\ (4.16)$$

is obtained.

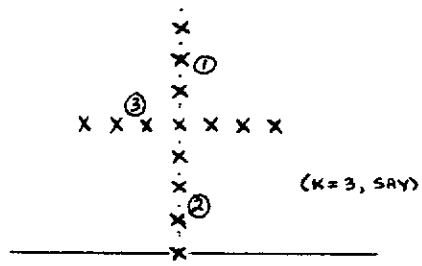



figure 4.7

Since the remaining subsets are as in George's method their count is given by the equation

$$N_{centre} = 2.O(2^{k-1}-1,3.2^{k}) + O(2^{k}-1,2^{k+2}) \quad \ldots\ (4.17)$$

and hence the total number of operations at stage k of the elimination $(2 \leq k \leq n-1)$ is given by summing equations (4.15) to (4.17), noting the number of subsets in each class. The total storage required is similarly evaluated.

In addition, for k = 1, (see figure 4.5) the count is given by the equation

$$N_{k=1} = 4.(2^{N-1}-2).\left[O(1,8) - O(2,7)\right] + 4\left[O(1,8) - O(4,5)\right] \ldots\ (4.18)$$

while at the final stage (again see figure 4.5) it is given by the equation

$$N_{final} = 4.O(2^{N}-1,2^{N}+1) \quad \ldots\ (4.19)$$

giving a total count for this modification of George's ordering as that in the equation

Total number of operations =

$$\sum_{k=2}^{N-1}\Big[4.\{O(2^{k-1},3.2^{k-1}+1) + O(2^{k-1}-1,5.2^{k-1}+1) + O(2^{k},2^{k+1}+1)\}$$

$$+ 4(2^{N-k}-2).\{O(2^{k-1}-1,3.2^{k}) + O(2^{k-1},2^{k+1}+1) + O(2^{k}-1,3.2^{k}+1)\}$$

$$+ (2^{2(N-k)}-4(2^{N-k}-1)).\{2.O(2^{k-1}-1,3.2^{k}) + O(2^{k}-1,2^{k+2})\}\Big]$$

$$+ 4(2^{N-1}-2).[O(1,8) - O(2,7)] + 4.[O(1,8) - O(4,5)]$$

$$+ 4.O(2^{N}-1,2^{N}+1) \qquad \ldots\ (4.20)$$

which sums to give the equation

$$N_{total} < 10.2^{3N} + (65-17N)2^{2N} + O(2^{N}) \qquad \ldots\ (4.21)$$

The storage required when similarly calculated is shown in the equation

$$S_{total} \simeq (8N - 24)2^{2N} + O(2^{N}) \qquad \ldots\ (4.22)$$

and so, the constant of the dominant term in George's result has been halved by means of this minor modification. Even so, although the result of George is clearly vastly superior asymptotically to normal Cholesky in terms of operation count, it is seen by comparing equations (4.1) and (4.21) that the order of the mesh has to be at least 5 for the dominant term in equation (4.21) to be less than the dominant term in equation (4.1). In fact, if the detailed expansion of the number of operations is evaluated it is seen that George's method with Reid's improvement is always superior to the normal Cholesky band method in terms of operation count.

It is, of course, possible to apply this kind of ordering to more complex grids and to the 3 dimensional Laplacian on a cubical grid, but, before this is done, it is perhaps useful at this juncture to discuss George's ordering in a wider and more tearing oriented aspect.

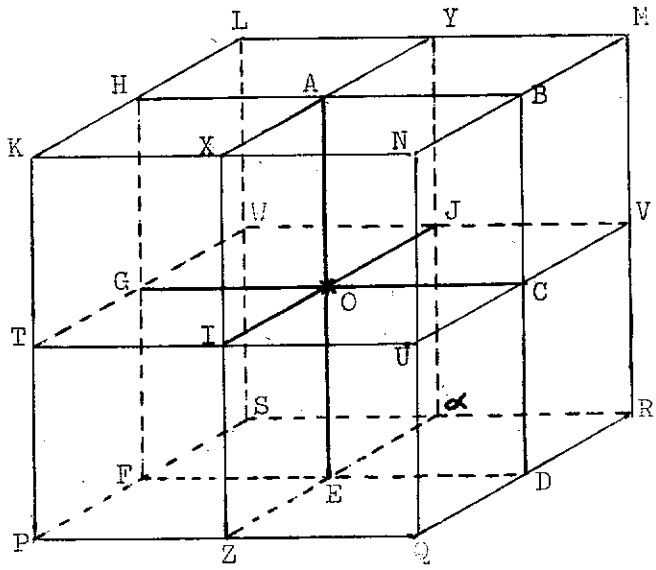
To envisage this process in terms of partitioning and tearing

consider the graph of the system and the diagrams of figure 4.5, starting with diagram (d) and working back in the reverse order. This means, in effect, that the points ordered last in the George ordering are being considered first. The removal of the points along one line of figure (d) divides the graph in two while removal of the other two lines splits the graph into four. Continued removal of points in such a fashion (akin to tearing where torn points are considered last) subdivides the graph into $(2^{N-1})^2$ units as indicated by the lines drawn in figure 4.5 (a). When the first points inside these square units are eliminated the units are in fact collapsed into one and the unit for the next stage in the elimination process is composed of four adjacent units from this stage. These four units are then collapsed into one new unit by the elimination in figure 4.5 (b). So the process continues until all points have been eliminated. At each stage, the interiors of the units are independent sets of points and, at each stage, blocks of 4 neighbouring units effectively coalesce. The splitting of the graph mentioned earlier is clearly similar to the tearing of a system where some points are selected at the beginning to be dealt with at the end.

With this description of George's method in mind it becomes easier to envisage applying the same ordering strategy to Laplace's three dimensional equation on the unit cube. Here the building block units are cubes rather than squares and the idea in pursuing this ordering is to obtain an order of magnitude calculation for the operation count to see if such an ordering would make the solution of large 3-dimensional Laplacians feasible by direct methods instead of the normal iterative techniques.

At each stage of the elimination process (except the first), it is possible to look upon this ordering algorithm as one which forms a composite fully connected cube from the 8 adjacent composite cubes left from the previous stage of the elimination process. At each

stage, the composite cube formed is hollow and full inasmuch as all the points interior to the cube have been eliminated and all the points on the sides of the cube are still present and are connected to every other point on the cube. The transformation at a particular stage is illustrated my means of figure 4.8 and the comments following it.




Composite cubes before the stage :

| | |
|---|---|
| AOIXGTKH | - 1 |
| AOIXNUCB | - 2 |
| ZIOEPFGT | - 3 |
| ZIOEUQDC | - 4 |
| JOE∝CDRV | - 5 |
| JOE∝GFSW | - 6 |
| AYJOLHGW | - 7 |
| AYJOBMVC | - 8 |

figure 4.8.

The decomposition is done as follows :

(i) Join blocks 1 & 2 → (12) by eliminating int.pts. of face AOIX

(ii) Join blocks 3 & 4 → (34) " " " " " " ZIOE

(iii) Join blocks 5 & 6 → (56) " " " " " " JOE∝

(iv) Join blocks 7 & 8 → (78) " " " " " " AYJO

(v) Join blocks (12) & (78) → (1278) by eliminating interior points of faces AHGO & ABCO & interior points of line AO.

(vi) Join blocks (34) & (56) → (3456) by eliminating interior points of faces GOEF & OCDE & interior points of line OE.

(vii) Join blocks (1278) & (3456) by eliminating points in the interior of the composite face TUVW.

Clearly, after these eliminations one is left with a hollow and full composite cube KPQNMRSL.

So, if O(m,n) is as defined in equation (4.13), then the number

of operations at stage k is given by the equation

$$\mathfrak{Z}(k) = 4 \times O(m_1,\beta) + 2 \times O(m_2,\gamma) + O(\delta,\varepsilon) \qquad \ldots\ (4.23)$$

where

$m_1$ is the number of points in interior face AOIX.

$\beta$ is the number of points in composition of cubes AOIXKTGH and AOIXNUCB.

$m_2$ is the number of points in the interior of composite face HGCB.

$\gamma$ is the number of points in the composition of cubes (12) & (78).

$\delta$ is the number of points in the interior of composite face TUVW.

and

$\varepsilon$ is the number of points on exterior of new composite cube.

Clearly,

$$\left.\begin{aligned} m_1 &= (2^{k-1}-1)^2 \\ m_2 &= (2^{k-1}-1)(2^k-1) \\ \beta &= (5.2^{2k-1}+2) \\ \gamma &= 4.2^{2k}+2 \\ \delta &= (2^k-1)^2 \\ \text{and } \varepsilon &= 6.2^{2k}+2 \end{aligned}\right\} \qquad \ldots\ (4.24)$$

Thus, it is now possible to calculate the number of multiplications when George's ordering is used to tackle this problem considering, as before, a cube with $2^N+1$ grid points in each direction. Since, at stage 1, each elimination clearly requires $O(1,26)$ operations, $\mathfrak{Z}(k)$ is seen to hold for $k=1$, (from equations (4.23) & (4.24)). Further, the number of cube compositions at stage k is $(2^{N-k})^3$ and, at the final stage, a hollow empty cube of side $2^N+1$ is eliminated giving, by $\varepsilon$ of equation (4.24), $O(3.2^{2N+1}+2,0)$ operations. So, finally, the total number of operations using George's ordering on the Laplacian on a unit cube with $2^N+1$ mesh points in each direction is equal to $N_{cube}$ given by the equation

$$N_{cube} = \sum_{k=1}^{N} (2^{N-k})^3 \mathfrak{Z}(k) + O(3.2^{2N+1}+2,0) \quad \ldots\ldots (4.25)$$

where

$\mathfrak{Z}(k)$ is given in equations (4.23) and (4.24).

This can be summed to give the equation

$$N_{cube} = 28.2^{6N} + O(2^{5N}) \quad \ldots\ldots (4.26)$$

Now, the number of operations required if normal Cholesky is used on ordinary row ordering is $\frac{1}{2}m^2n$ where $m$, the bandwidth, $\sim 2^{2N}$ and $n$, the order of the system, $\sim 2^{3N}$ giving the number of operations for Cholesky, $N_{chol}$, as

$$N_{chol} \cong \tfrac{1}{2}.2^{7N} \quad \ldots\ldots (4.27)$$

Thus, it is clear from equations (4.26) and (4.27) that the George ordering is again asymptotically better than the row ordering technique by a factor equal to the number of points in one direction of the grid, but that $N$ would require to be greater than 5 for obvious benefits to occur. All that a Reid type of improvement does is to cut the constant factor multiplying $2^{6N}$ in equation (4.26). It is therefore apparent that, even using this George ordering, direct methods are liable to be uncompetitive with iterative ones for large scale three-dimensional Laplacian problems.

As a final comment to this section, it should be mentioned that several other ordering techniques can be looked at from a partitioning and tearing viewpoint although this is not necessarily the context in which they are usually quoted. To illustrate this two matrix inversion methods are discussed. One by Nathan and Even (1967) and the other a bandwidth minimisation technique due to Arany, Smyth & Szoda (1971) and discussed in Cuthill (1972).

The strategy of Nathan and Even is explicitly derived in terms of the graph of the matrix. The idea here is to tear out a set of points such that every loop in the original graph of the matrix is

broken. The main part of their paper discusses an algorithm for finding this tearing set which is a very non-trivial matter for graphs of any size. Elimination is first performed on the untorn points of the system. If the matrix is partitioned according as to whether the points are in the tear set or not, then it can be reordered to have the form given in figure 4.9, where the first block corresponds to the untorn elements.

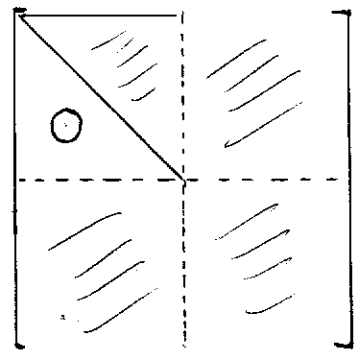


figure 4.9

The upper left hand block is seen to be trivially invertible and the lower left hand block can be considered the D partitioned block of the torn matrix as in figure 2.1, although it is somewhat more dense than originally envisaged there.

It should be noted here that, in fact, George's strategy is an example of this kind of procedure. Clearly, the complement of P(1) (the first points to be eliminated in George's method) is a set of points whose removal breaks every loop of the original graph of the system. It is certainly clear that no subset of these points possess this property and it may be true that this is the smallest number of points which can so divide the system. It is interesting to note that the improvement to George's strategy does, however, still leave some loops in P(1). (see figure 4.5(a)).

In the Arany, Smyth and Szoda paper a somewhat more normal set of tearing elements is employed. This set is a point cutset of the graph (see Harary (1969)) whose removal will result in the graph

of the matrix being reduced to a set of smaller independent graphs. Numbering to achieve a near optimal minimum bandwidth is then carried out on the separate components of the graph.

# CHAPTER 5.

## SOME COMMENTS ON THE SOLUTION OF SIMULTANEOUS LINEAR ALGEBRAIC EQUATIONS.

## Section 1. Introduction.

Whereas the last three chapters have basically been concerned with an a priori ordering of rows and columns which put a given matrix into a form in which fill-in is restricted to certain regions of the matrix, this chapter considers the use of ordering techniques to determine which particular non-zeros are chosen as pivots at each stage with a view to reducing the fill-in when Gaussian elimination is performed. This type of pivoting is called sparsity pivoting. It is generally assumed that, in the symmetric case, pivots are only chosen from the diagonal so that symmetry is preserved. This chapter places no such a restriction on the choice of pivots and, unless it is explicitly stated otherwise, the discussions of this chapter are applicable to general systems.

The results of chapter six indicate why some form of sparsity pivoting should be employed. Consider, for example, a matrix of order 100 with non-zeros on the diagonal and a probability of about 0.3 of any off-diagonal element being non-zero; if the pivots are chosen from the diagonal in the natural order, then the fill-in can be expected to be about 2500 , while good sparsity pivoting techniques can reduce this fill-in to about 450. This considerable cut in fill-in is valuable in two ways. It is evident that the less elements there are in each reduced matrix, the less operations have to be performed on it when it is being decomposed. In fact, it is seen from the results in the tables in sections 3 and 4 that the number of multiplications in this decomposition phase varies almost monotonically (very occasionally a larger total fill-in will give a lower multiplication count) with the total fill-in produced. In addition, it is obvious that the fewer non-zeros there are in the decomposed form, the less work will have to be done when solving for subsequent right-hand sides. In the normal Gaussian LU decomposition, the number of multiplications in these subsequent solutions

is identical to the number of non-zeros present in the decomposition. This fact should be borne in mind when considering methods which are inherently slower than others but produce less fill-in. Not only will the resulting ordering probably result in a faster decomposition and solution but, in addition, the very pivot selection process itself might be speeded up by having a lower fill-in. In addition to this advantage, there is the obvious reduction in the storage required to hold the decomposed form. In order to take full advantage of this it is necessary to store the matrix in a packed form holding only the non-zeros and some integers to access and identify them. A discussion of possible storage schemes is the subject of the second section of this chapter.

In most of this chapter little mention is made of using pivoting for numerical stability. It is certainly assumed that in any working procedure some safeguards have to be taken against numerical instability. This may be by using some crude form of pivot tolerance as in the algorithm of section 6 or in controlling the maximum size of the multipliers and monitoring element growth as in Curtis and Reid (1971c, 1971d). It is assumed, as was mentioned in chapter one, that all matrices have been scaled prior to the pivot selection process (see, for example, Curtis and Reid (1971a)). Often the criterion for selecting a non-zero as a possible pivot is a combination of sparsity and numerical considerations as in the ordering procedures for GNSO (Gustavson et al. (1970)) or in the program of Curtis and Reid (1971b). The validity of the comparisons for fill-in in the following sections is not destroyed by the fact that no consideration has been taken as to whether any particular selected pivot would be numerically suitable. Over very many different examples the effect of having certain elements forbidden to be pivots will cancel out and, in addition, experiments by Curtis and Reid (1971c, 1971d) where they have varied the numerical criterion indicate that little

difference occurs in the relative fill-in unless the numerical test for a particular pivot is very strict. It is worth noting that for patterns like that of figure 2.1 in chapter one, it is sometimes possible to obtain more accurate numerical results using sparsity pivoting criterion than using strict partial pivoting since the appalling difference in fill-in greatly increases the number of operations and hence sources of possible instability.

There are essentially two different types of sparsity pivoting. A priori ordering of the columns is considered in section 3, and it is shown that even the best of such orderings known and tested by the author gives relatively bad results in comparison with orderings which select the pivot from the whole remaining matrix at each stage by means of a property of the non-zeros in the matrix at that stage (this property is called a 'merit value'). Some of these second kind of orderings, called 'local' orderings, are examined in section 4. In this section, examples are given of matrices where local minimisation does not lead to global minimisation. This result is perhaps not surprising since it is well known that minimising the local multiplication count does not minimise the count globally, but examples of the phenomenon are singularly lacking in the literature. However, it seems from the results of section 4 that this local minimisation presents the best easily implemented scheme for obtaining a near optimal ordering although the time to produce an ordering by this scheme is somewhat excessive. The work of section 5 attempts to remedy this by finding methods of short circuiting the work and makes such an ordering scheme feasible and competitive. By thus minimising the fill-in, the work in decomposing the system is generally reduced and the solution of further systems speeded up. It therefore seems worthwhile to try and achieve this near minimum.

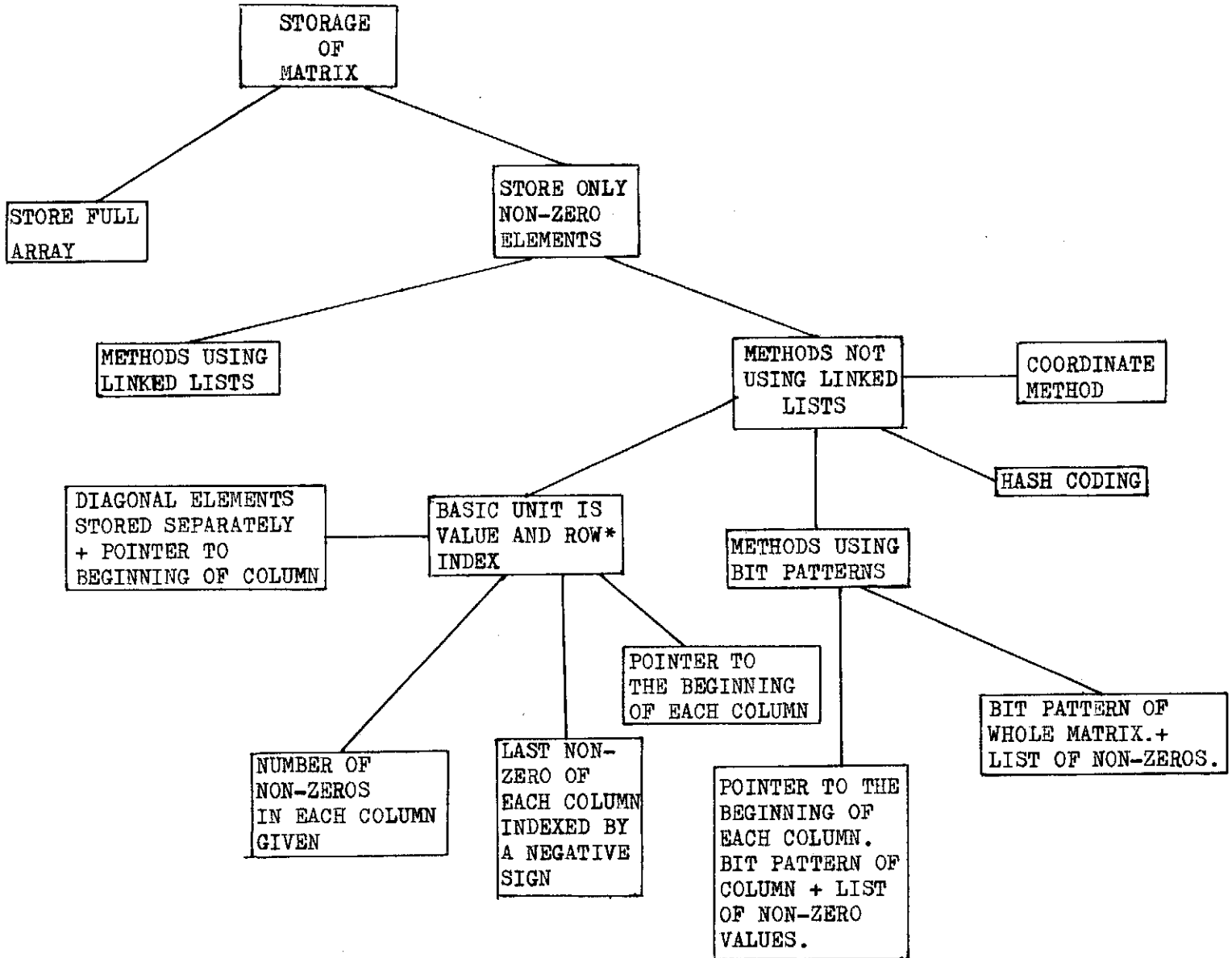
The final section discusses a sparse matrix package, designed

by the author, which attempts to illustrate some of the points of the previous sections and indicates the great gains to be made by using sparsity techniques.

## Section 2. Storage Considerations.

It is essential when dealing with sparse matrices that advantage is taken that most of the elements are zero and only the non-zero elements are held in the computer. This type of storage is referred to as holding the matrix in packed form. A unit of storage is defined as that quantity of storage required to hold a real variable in single length. Thus a unit on the 1906A would be two 24-bit words while a unit on the IBM 370/165 would be 4 bytes (8 bits to a byte) and, on both machines, an integer variable would occupy one half of a unit. Then, ideally, it would be desired to reduce the payload from the $n^2$ units required for a full matrix to $\tau$ units where $\tau$ is the number of non-zeros in the matrix. This theoretical minimum is, however, seen to be fairly impractical since it is not much good holding a set of values for the non-zeros if there is no way of telling to which entry of the matrix each of them refers. Nearly all sparse matrix storage schemes have therefore to employ, in addition to the $\tau$ storage units for the values of the non-zeros, some extra indexing arrays or pointers to identify the individual elements. An elementary way of doing this would be to associate with each non-zero two integers recording the row and column number of the element in question. This is called the coordinate method (Phillips (1970)). However, this method is still impractical if it is desired to find a particular element since a scan is required through all the non-zeros until it is found, checking the subscripts at each stage. In addition, $2\tau$ extra half-units of core are required for this method bringing the total number to $4\tau$ which usually exceeds that of $S_{basic}$ in equation (2.4), the storage count for a much more feasible type of storage scheme. Some method has therefore to be found for indexing the non-zeros so that they can be easily accessed. This can be done in a variety of ways, from holding the full array, (already discounted

because of storage being used for zeros), to merely holding a bit representation of the matrix alongside a linear list of the non-zeros. The supplementary problem of removing and inserting non-zeros is one which is present in the triangular decomposition of a system prior to its solution and raises other questions concerning indexing. One way of dealing with this problem is to use linked lists (Knuth (1968)). The distinction between methods using linked lists and those not was made very clear in the survey paper of Reid (1972a). A fuller breakdown of some currently used storage techniques is given in figure 2.1 which is followed by a short discussion on most of the types of method mentioned in the figure.




* or column (see note on page 121)

figure 2.1

Hash Coding.

Hash coding has long been a useful tool of the compiler writer (Morris (1968), Maurer (1968)) and is basically a means whereby, to any given identifier or entry to a table, a unique pseudo-random number is associated with it. This number is used to index the location in a table where information concerning this identifier is held. The principal idea is that while the number of different identifiers may be very large only relatively few are used at any one time, and so the table can be much smaller than the total number of identifiers. Since the mapping of identifiers to table positions is many-to-one, methods have been devised for dealing with collisions when some other location must be found for the colliding element. The analogy with sparse matrices has been made (Morris (1968), Jimenez (1969)) where the number of $(i,j)$ element values ($n^2$ if the matrix is of order $n$) corresponds to the number of identifiers while the number of non-zeros corresponds to those identifiers actually in use. De Villiers (1971) has experimented with hash coding the element indices $(i,j)$ for various table sizes and suggests that hash coding of these indices could be made the basis for a sparse matrix storage scheme. Certainly, the storage required for such a scheme is attractively low, but the scheme is impractical for the following three reasons. Firstly, in most hash coding work and in the codes arising from index manipulation, it is assumed that access to the elements is random which is clearly not so in most matrix operations where row and column scanning are commonplace. The most damming indictment against such a scheme is that there is no a priori way of determining in advance whether the element indicated by the indices $i$ and $j$ is zero or non-zero. Thus, when scanning a row, say, all $n$ locations indicated by the hash codes have to be separately examined to see whether the element is non-zero and, if so, to obtain its value. Finally, the time taken to calculate the hash code, particularly if codes ensuring few

collisions are employed, is not insignificant.

Bit patterns.

The use of bit patterns in sparse matrix storage schemes has been advocated several times in recent literature (Smith (1969), Gustavson et al (1970), de Buchet (1971)). However, the use of such a technique is now falling out of favour even with its own advocates and is presently regarded as being infeasible apart from matrices with non-zeros in small blocks.

The use of bit patterns involves indicating whether or not an element is non-zero by setting a single bit to one or zero accordingly. The bit pattern can be for the whole matrix (Gustavson et al (1970)) or can deal with a column at a time, a pointer being used to the head of a column and a bit pattern indicating the zero/non-zero structure from the first to last non-zero in the column. A superficial attraction of such a scheme is in the reduction in storage which it offers. The storage required, $S_{bit}$, for a Gustavson type scheme being given in the equation

$$S_{bit} = 2\tau + \left[N^2/p\right] + 1 \quad \text{half-units} \qquad \ldots\ldots (2.1)$$

where $\tau$ is the number of non-zeros.

N is the order of the system.

p is the number of bits per half word.

and $[x]$ is the greatest integer strictly less than x.

This storage is, however, dependent on the machine configuration and, in some cases, the savings are not as great as might be hoped. This can be seen if the following example and machine configuration is examined. Consider a machine of half word length $2^4$ bits and a matrix of order $2^k$ $(k \geq 3)$ with an average of $2^3$ non-zeros per row, then the respective storage using a bit scheme $(S_{bit\ 2^k})$ and a column pointer, row index scheme, as in equation (2.4), $(S_{col\ 2^k})$ is given by the equations

$$S_{bit\ 2^k} = 2^{k+4} + 2^{2k-4} \quad \ldots\ (2.2)$$

and

$$S_{col\ 2^k} = 3.2^{k+3} + 2^k \quad \ldots\ (2.3)$$

It is seen from these equations that, although savings in storage of almost 50% are made when k is small, the bit pattern scheme takes more storage for all k greater than 7. This means that, for orders of matrices greater than about 200, the bit scheme (with the density given, $2^{3-k}$) will, in fact, take more storage than that of equation (2.4). Since matrices of about this order and density are fairly commonplace, the validity of using bit patterns to save storage is very dubious indeed.

The author did some pilot tests on bit mapping schemes but found the 'bit picking' involved exceptionally laborious if high level languages were used in the implementation. Hybriding a normal package to low level language bit-picking subroutines involved an unhealthy number of procedure calls.

Partly because of these reasons and partly because of pilot tests on various methods, Gustavson (private communication) no longer advocates the use of bit strings and, in fact, the successor to the program GNSO (Gustavson et al (1970)), called GNSOIN (Gustavson (1972)), uses a storage scheme similar to that described before equation (2.4).

Thus, with the possible exception of matrices with closely grouped non-zeros in bunched blocks (de Buchet (1971)) where storage may be saved and manipulation made feasible, the use of bit patterns in sparse matrix storage schemes is not recommended. Even in this exceptional case, the use of schemes such as Jennings (1966) might be better.

Storage by columns.

One way of avoiding the search problem inherent in the coordinate storage system while at the same time avoiding the

manipulative problems of bit picking or hash coding is to order the elements of a column contiguously (if the matrix is stored by columns) indexing the beginning of each column (and the end of the last one) and carrying a reference to the row index with each non-zero. Since these indices are integers, it can generally be arranged so that they occupy only one half of a unit of core. This would give the total storage requirement for such a scheme as

$$S_{basic} = 3\tau + n \quad \text{half-units} \qquad \ldots. (2.4)$$

with $\tau$, again, the number of non-zeros in a system of order $n$.

It is clearly seen from equation (2.4) that, in order for this storage requirement to exceed that for the unpacked form, $\tau$ would have to be greater than $\frac{1}{3}(2n^2-n)$. This means that the matrix would have to be about 67% dense which is hardly of interest in a sparsity context. This fairly simple storage scheme has been used in many sparse matrix packages (see, for example, Tinney (1969b), Tinney and Ogbuobiri (1970), McNamee (1971), Curtis and Reid (1971c, 1971d), Hsieh and Ghausi (1971a), Gustavson (1972)) and has many variants.

It is possible, for example, to avoid storing the column pointers by making the last row index in each column negative (Phillips (1970)), but this assumes whole columns are stored contiguously to each other and makes the accessing of any particular column more difficult. This latter objection also applies to the scheme in McNamee (1971) where the number of elements in each column rather than pointers to the beginning of columns are held. Many variants apply to cases where the matrix is symmetric or incidence-symmetric and advantage can be taken of symmetry or structural symmetry. Most of Tinney's codes are for use on incidence-symmetric power system networks and they take advantage of that fact by storing the values of the symmetrically placed elements alongside each other in the scheme. In most of these schemes, the diagonal is stored separately along with the column pointers. Thus, the storage required for Tinney's symmetric and

incidence-symmetric matrices (Tinney (1969b) and Tinney and Ogbuobiri (1970)) is given by $S_{sym.}$ and $S_{inc.sym.}$ in the equations

$$S_{sym.} = \tau + 2n + \tfrac{1}{2}(\tau - n) \quad \text{half-units} \quad \ldots\ldots (2.5)$$

and

$$S_{inc.sym.} = 2\tau + n + \tfrac{1}{2}(\tau - n) \quad \text{half-units} \quad \ldots\ldots (2.6)$$

So, the saving over $S_{basic}$ for an incidence-symmetric matrix of order 100 with 400 non-zeros would be about 20%. It is possible, of course, to utilise the additional information of incidence-symmetry in facilitating the triangular decomposition procedures rather than in saving storage as in Zollenkopf (1971).

Linked lists.

The previously mentioned methods of storage are satisfactory for static matrix operations defined by the fact that the non-zero pattern of the matrix does not change during the operation. Such a static operation is the back substitution phase in Gaussian elimination or in the solution of successive right-hand sides by an already decomposed matrix. Thus, algorithms like method Z of Curtis and Reid (1971d) can efficiently use such storage schemes. However, the problem with all of the methods discussed above is that quite a great deal of work (for example, shuffling or reordering existing non-zeros) has to be done if it is desired to add a non-zero element to the matrix. This is a common occurence in the forward elimination phase of Gaussian elimination and these methods can only really be satisfactorily employed if some other information is available, as in the case of Gustavson et al (1970) and Gustavson (1972) where the non-zero structure of the LU decomposition is known in advance, or if some restriction is imposed, as in the case of Curtis and Reid, method X, (1971d), where the order of the pivot columns is known a priori and they can be operated on in some predetermined order.

In order to combat this problem, the normal technique is to use linked lists (Knuth (1968)). A linked list is a list consisting,

in addition to some information concerning the elements of the list, of elements which are themselves pointers to other elements in the list. In this way, an order or partial order can be superimposed on elements in the list. An example of this kind of structure is given in figure 2.2(a) and figure 2.2(b) where (b) is a linked list corresponding to the structure in figure (a).

a→b→c

d→e

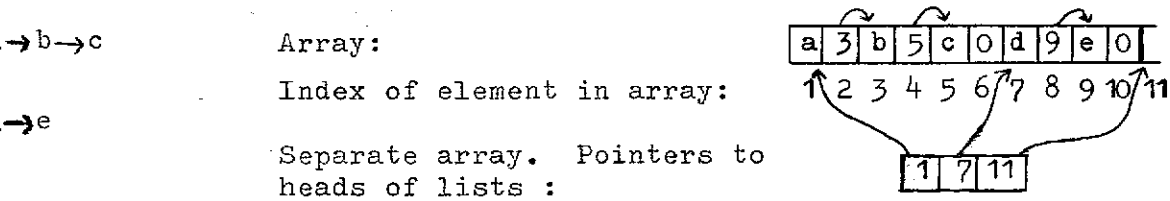



figure 2.2.(a) figure 2.2.(b)

A more detailed example of how the linked list concept is used to store a sparse matrix is given later in this section and in chapter eight when a scheme developed by the author for symmetric matrices is described. Linked lists, as employed in the storage of sparse matrices, can be generalised as follows. The description is developed for column ordering (that is, ordering rows within columns) but it is equally possible to use such schemes for row ordering. With each non-zero is associated some combination of the following :

(i) Row index.

(ii) Column index.

(iii) Pointer to the next element in the column.

(iv) Pointer to the next element in the row.

} forward pointers.

(v) Pointer to the previous element in the column.

(vi) Pointer to the previous element in the row.

} backward pointers.

As before, an array of pointers is used to designate the beginning of each column.

Clearly, the number of the above attributes it is desired to associate with each non-zero will depend on how much storage is available (for example, if all the above six integer attributes were used then, assuming an integer takes half the space of a real, the

matrix would have to be about 25% dense or less for the above scheme to be better, as regards storage requirements, than storing the matrix in full), and on how the particular operations to be performed on the matrix are organised (for example, algorithm Z of Curtis and Reid (1971d) uses two different schemes at different parts of the solution process). Some comments on which attributes are better used are now given.

Most solution schemes can be organised in such a way that backward pointers are a luxury rather than a necessity and so attributes (v) and (vi) are not used in any packages known to the author. Working methods, however, do use most other combinations of attributes. Method Y of Curtis and Reid (1971d) uses all four, while a scheme suggested by Ogbuobiri (1970) and implemented by Duff (see section 6 of this chapter) uses only a row index and column pointer with each non-zero. The defects of Ogbuobiri's scheme which saves only $\tau$ units of storage over the scheme in method Y is seen in table 2.1 where times for a sparse matrix procedure programmed using these two storage schemes are given. These procedures are given in the appendices. Table 2.2 illustrates this even more clearly by giving times of runs for various matrices of a program to calculate the fill-in when the pivot for local minimum fill is chosen at each stage (see section 5 of this chapter). In view of this fairly significant effect on the time taken for procedures utilising these two schemes it is generally the case (unless the system is critically memory bound) that the use of both row and column pointers is very desirable. It is really only by this means that the factor $n$ can be replaced by $\bar{r}$ (the average number of non-zeros in a row during the elimination process) in the various operation counts for matrix processes. A storage scheme similar to that for method Y of Curtis and Reid (1971d) but developed for use with symmetric matrices is now described.

Figure 2.3 shows the portion of the matrix stored and the way in which the pointers link the various elements which are stored.

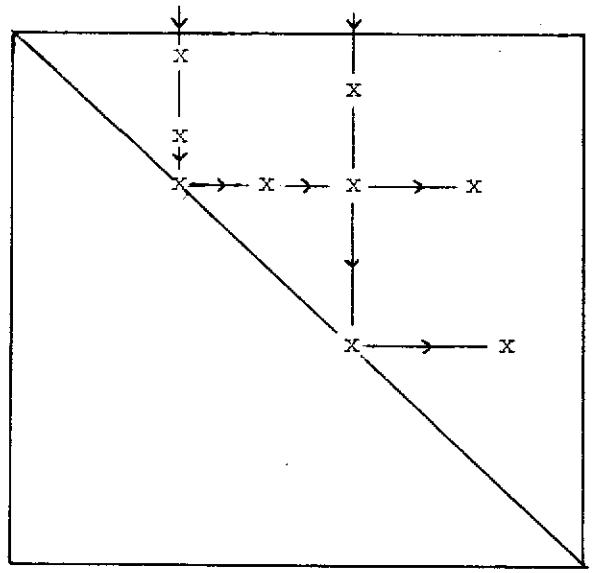


figure 2.3.

Hence, with each non-zero there is again associated two pointers, one to the next element in the column (-1 if the element is on the diagonal) and one to the next element in the row (0 if the element is the last element of a row). The row and column indices are stored with each non-zero and column pointers to the beginning of a column (or row) are also stored, making the total storage required for this scheme as that given by the equation

$$S_s = 3\tau + 4n \qquad \text{half-units} \qquad \ldots\ (2.7)$$

Implementation of this scheme is shown in chapter eight and a program written for that chapter is given in the appendices. The only difficulty left is that of determining whether the element being accessed is the element (i,j) or the element (j,i). A single integer variable (called L or K in the program given) determines this.

It is, of course, possible to omit the row and column indices from the non-zeros being stored thus saving $\tau$ units of storage. The normal means of then finding out these attributes of a particular non-zero is to chase down its row (column) links to the last element

of the row (column) where the pointer has been replaced by the negative row (column) index. The time penalty for doing this is not all that large since most rows and columns have only a very few non-zeros. Curtis and Reid (1971d) incorporated this into their algorithm Z and found, by experiment, that such a change caused a time penalty of about 20%.

In all of the above schemes, with or without links, it is important that the storage is implemented in such a way that there are few, if any, half-units left unused. This can be done in two ways. The integer information concerning the non-zeros can be stored in an integer array separate from the real array containing the values of the elements. This, however, tends to be clumsy and leads to some indexing problems which, although easily solved, waste time calculating indices during the solution procedures. A better way (unfortunately, only of use when there is an even number of integer attributes per non-zero) is to store the information in one array, equivalencing the integer array of attributes to the real one of element values. The effect of this is shown in figure 2.4

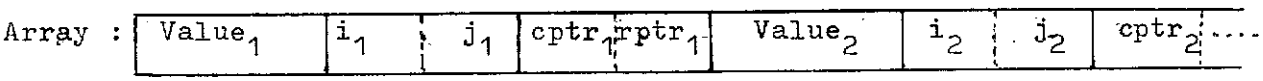



figure 2.4.

If the real array V(3Υ) is equivalenced to the integer array INDEX (6Υ), then successive element values are found in V(1), V(4), V(7) etc,

row indices in INDEX (3), INDEX (9) etc.

column indices in INDEX (4), INDEX (10) etc.

row pointers in INDEX (6), INDEX (12) etc.

column pointers in INDEX (5), INDEX (11) etc.

Throughout this section, it is important to realise the duality between rows and columns. As mentioned in chapter one, once the decomposition of A is known that of $A^T$ follows trivially. Hence,

in all of this section the words row and column can be interchanged without affecting the validity of any of the comments. It should be understood, however, that the time an algorithm takes may depend on whether the matrix is stored by rows or columns since the algorithm is likely to have a bias towards accessing the matrix elements in one of these directions. In fact, the storage scheme is somewhat less separable from the particular solution process used than might be imagined from the discussion in this section, the best scheme being determined by the structure of the matrix (for example, the variable bandwidth structure of Jennings (1966)) or the solution process envisaged (for example, a column ordering would be inadvised for row Gaussian elimination (Gustavson (1972)). It is worth remembering that the storage requirements given earlier in this section are only those for the physical storage of the matrix. When actually implementing an algorithm, there are sometimes fairly significant overheads for arrays indicating row counts or pivot order, some of which are integral to the solution process and others which merely facilitate it. Most algorithms would not normally require more than about 10n half-units for these overheads - only insignificant if the matrix is fairly dense or is very large with many non-zeros.

The following notes concerning storage can now be made:

(i) Bit mapping techniques and hash coding are not generally recommended.

(ii) The structure of the matrix should be examined to see if it can be stored economically by partitioning or variable bandwidth techniques.

(iii) Unless a priori knowledge is known concerning the system (for example, the pivot order or fill-in characteristics), then a linked list structure is recommended.

(iv) The use of backward pointers should be regarded as a luxury and avoided if possible.

(v) Both row and column pointers are recommended since they give a great advantage in speed without much increase in storage.

(vi) Row and column indices should be used if storage permits. The penalty is by no means excessive if they have to be dropped.

(vii) Always choose a storage scheme compatible with the method of solution.

(viii) Look out for the possibility of using different storage schemes at different parts of a solution process. In this, some or all, of the links may be dropped, thus releasing storage for other use or users.

| MATRIX order / Procedure density | 100<br>0.10 | 100<br>0.06 | 100<br>0.03 | 50<br>0.20 | 50<br>0.10 | 50<br>0.02 |
|---|---|---|---|---|---|---|
| COUNT | 0.0597 | 0.0375 | 0.0188 | 0.0542 | 0.0157 | 0.0054 |
| COUNT 2 | 0.0087 | 0.0064 | 0.0025 | 0.0140 | 0.0034 | 0.0012 |

Table 2.1. The times (in seconds of 1906A milltime) for the execution of one call of procedures (see appendices) to evaluate fill-in when a non-zero is chosen as pivot in Gaussian elimination. Procedure COUNT uses column links only while COUNT 2 uses row and column links.

| MATRIX order / Structure density | 100<br>0.02 | 50<br>0.10 | 50<br>0.05 | 50<br>0.02 | 25<br>0.20 | 25<br>0.10 |
|---|---|---|---|---|---|---|
| COLUMN LINKS ONLY | 37.698 | > 100 secs | 13.661 | 0.371 | 18.520 | 2.487 |
| COLUMN AND ROW LINKS | 8.939 | 49.594 | 5.054 | 0.271 | 8.086 | 1.240 |

Table 2.2. A comparison of times (in seconds of 1906A milltime) of fill-in procedures using different matrix storage schemes. One matrix for each order and density was used.

## Section 3. An examination of a priori methods of sorting a matrix with a view to reducing fill-in in Gaussian elimination.

This section is concerned with deciding how to arrange the columns of a matrix so that when the matrix is pivoted on with the columns chosen in this predetermined order, the growth of non-zeros is kept satisfactorily small during the elimination process. The term "satisfactorily" is dependent on the amount of core available in the computer and on the loss in sparsity in using such an a priori method in contrast to one which determines the pivot on a 'local' basis.

The reasons for using some kind of sparsity pivoting have already been pointed out in the introduction. There are several reasons why this sparsity pivoting should be of the kind mentioned above. Firstly, it is a very much quicker process than using a 'local' ordering. Once the initial column ordering has been done, the selection at each stage has only to be made from those non-zeros in the uneliminated rows of the given column rather than having to consider all the remaining non-zeros at each stage. It is also possible to economise on storage by only storing the matrix by columns and operating on each column as it becomes pivotal during the elimination process. Thus the need for row links is obviated. In addition, the use of such an a priori ordering makes the use of backing store more feasible. This is done by making each column, when it becomes pivotal, be still in its original state and doing all the operations to it at this stage. This necessitates only the issue of read instructions to get information from the backing store, the only write instruction being the structure of the column after it has been pivotal.

Several methods have been suggested for the a priori ordering of the columns. An obvious choice is to order them in order of ascending column count where, as was mentioned in chapter two, the column count of a column is the number of non-zero elements in it.

This is called method CJ in what follows. Several authors have suggested more sophisticated techniques. Tewarson (1967b) has advocated ordering the columns in order of increasing $\delta_j$ where $\delta_j$ is defined as the total number of non-zeros in all the rows of the matrix which have a non-zero in column j. It is to some extent a measure of the amount of the matrix influenced by any one column. Methods using this are denoted by DJ. Tewarson (1967b) has also suggested using as a measure for each column the maximum fill-in that could occur if a non-zero in the column were selected as pivot averaged over all the non-zeros in the column, that is using

$$\gamma_j = 1/c_j \cdot \left\{ \sum_i^1 (r_i-1)(c_j-1) \right\} \qquad \ldots\ldots (3.1)$$

where the summation is over those rows with non-zeros in column j. This can be written in terms of the $\delta_j$ of method DJ as

$$\gamma_j = (\delta_j - c_j).(1 - 1/c_j) \qquad \ldots\ldots (3.2)$$

The method obtained by arranging the columns in order of ascending $\gamma_j$ will be called method GJ. Orchard-Hays (1968) has suggested using as the column measure the maximum possible fill-in which can occur when the non-zero in the column with minimum row count is selected as pivot. This measure is

$$h_j = (c_j-1)\left\{\min_i (r_i-1)\right\} \qquad \ldots\ldots (3.3)$$

where the minimum is over these rows which have a non-zero in column j. HJ will denote the method obtained by ordering the columns in ascending order of $h_j$.

A comparison of these four methods against some structured matrices and a number of random ones was made and the results are shown in table 3.1. It should be noticed that in the program implementation of these methods and in all the programs written for sections 3, 4 and 5, the sparse matrix under consideration is stored in a linked list packed form, with row and column pointers and indices (see section 2). At each stage during the elimination the pivot

selected from each column was that non-zero in the column lying in the row with minimum row count. It is seen from the table that, in spite of their sophistication, the other methods do scarcely any better than method CJ, while method HJ generally does noticeably worse. In some ways this is hardly surprising. Method DJ does not use much more information about the fill-in than CJ. It will correctly place columns with a few non-zeros in heavily populated rows near the end of the column order but will perhaps fail to place singleton columns at the beginning. As might be imagined the results show that DJ and CJ have similar characteristics in the amount of fill-in produced. Method GJ, on the other hand, assumes that each element of the column is equally likely to be selected as pivot. Even in a random matrix this is untrue because of the strategies used to select a pivot from within a column. This will probably mean that the early columns will have non-zeros with low row counts selected as pivots while later in the elimination the only non-zeros left will be those with high row counts. However, this is an averaging process and is therefore unlikely to give bad results. It will certainly select singleton columns first and demote columns with few non-zeros in heavily populated rows. This is in agreement with the results from table 3.1 which show GJ to be, on the whole, about the same as both CJ and DJ. In method HJ the problem is that it is increasingly unlikely after the first elimination that the non-zero with initial minimum row count will be available for selection as a pivot, since that row (particularly in view of its low row count) is quite likely to have already been eliminated at some previous stage in the process. Thus, it is hardly surprising that the results shown in table 3.1 for method HJ are generally worse than the other methods, particularly when denser matrices are considered. Therefore, it is concluded that only methods DJ, CJ, and GJ, from those discussed, should be considered as reasonable a priori methods for column ordering, and

that, in general, method CJ works best on the majority of examples, obviously is the quickest to implement, and should be used when a priori techniques are being considered.

In order to illustrate the importance of using some form of a priori ordering on the columns, similar calculations using the natural order of the columns were made and the results are shown in table 3.1 also. These results show clearly that very significant savings in fill-in and multiplication count are made by using any of the above mentioned a priori methods.

The second problem when implementing a priori techniques is the criterion to be used in selecting a pivot in a given pivotal column. Clearly, from sparsity considerations, the best one to use is generally that which has minimum row count (if it is here desired to minimise local fill then it may well be asked why an a priori technique is being used) as was done for the results in table 3.1. This method is called method ROW. However, it is not always possible to do this easily since in, for example, method X of Curtis and Reid (1971d) the pattern of fill-in is unknown for all the columns which have not yet been pivotal. There are several other possibilities open. One is to select pivots on the basis of the original row counts in the matrix. This is called method R2. (Thus, Orchard-Hays pivot ordering combined with selecting pivots on the basis of initial minimum row count is denoted by HJR2). Another is to base the selection on choosing the non-zero of a minimum row count, where this row count is obtained from the original one by subtracting for all the non-zeros already eliminated. This method is termed R1. Finally, Tewarson (1966) has suggested assuming a random distribution of the non-zeros and updating the row count at each stage by using the row count of the pivot row and the row itself before the stage began. This update is, for stage k,

$$r_j^{k+1} = r_i^k + r_j^k - (r_i^k - 1)(r_j^k - 1)/(n-k) - 2 \qquad \ldots\ (3.4)$$

where i is the row of the element selected as pivot at stage k and the update is only done for those rows j with non-zeros in the pivot column. This method is denoted by PROB.

Results for these four selection procedures with a basic column ordering of CJ are given in table 3.2. It is verified that it is best to use the full updated row count if it is available, failing which the probability update is next best although it is, in general, worse than the full update and, as might be expected, is noticeably worse if structured matrices are considered. This result is hardly surprising but, at first sight, the difference between methods R1 and R2 seems larger than would be expected. It seems that zero is a better estimate of the change in row count than the number of non-zeros in the already eliminated part of the rows. This is indeed seen to be the case since every time some non-zeros are added to a row one is removed, and this difference in the number added from the number removed will be partially balanced by the number of times an element is removed without any being added. This type of argument provides a kind of justification for the results of table 3.2. Certainly, if any method were to be recommended, then it is suggested that the full update of the row counts be used since this almost always gives the best results. However, if this is not possible because of the nature of the solution procedure, then the use of Tewarson's probability update, equation (3.4), even for structured matrices is found to be an improvement on methods R1 and R2. The use of this carries, however, an additional time penalty above the other methods as is seen in table 3.3. This is not too drastic (almost always less than 5%) but should be borne in mind when implementing this method. If it is decided, because of the nature of the matrix or this time penalty, not to use this update then it is strongly advised to use no update at all rather than the partial update of method R1.

A full list of the fill-in produced when the 16 combinations

of CJ, DJ, GJ and HJ with ROW, R1, R2, and PROB is given in table 3.4. This indicates that, in most cases, the use of CJ with full row update is as good as any other method and it is therefore recommended as a general purpose a priori method. The table also indicates that there is no other combination of a method inferior to CJ with one inferior to ROW which gives better results and so tables 3.1 and 3.2 contain the best results for the 16 possible combinations.

In the foregoing discussion a comparison of only a priori methods has been undertaken and the question of their value as against 'local' methods has not been fully considered. This is remedied in table 3.5 where the method CJ with full row count update, the best of the 16 methods considered above, and a non a priori method using Markowitz's criterion (1957) are compared with respect to time, fill-in, and multiplication count. The percentage increase in the final total number of non-zeros of the best method over the fill-in due to Markowitz is also given. It is seen that, in some cases, the fill-in obtained using even the best a priori method is considerably greater than that using the Markowitz criterion. In addition, the increase in time required to implement the local method is not always that great. It is thus arguable that a priori methods are not good ones because of their much greater fill-in without necessarily being substantially quicker to implement. Most sparse matrix packages therefore adopt local criterion for pivot selection while a priori methods are generally only used in linear programming codes (see, for example, Larsen (1962)).

If the fill-ins in table 3.5 are compared with those in table 2 of Curtis and Reid (1971d), it is seen that they are compatible with pivoting using a low level of numerical control (the 'u' in the Curtis and Reid paper tending to zero). The Markowitz fill-ins from table 3.5 are a little lower than those with $u = 1/64$ in Curtis and Reid while the a priori results (method X of Curtis and Reid (1971d)

uses CJR2) are slightly further apart, as might be expected. Runs performed on the structured matrices using the Harwell program (Curtis and Reid (1971b)) with the parameter u set to zero indicate agreement of fill-in to within differences due to different tie-breaking procedures.

| METHOD \ MATRIX | | RANDOM MATRICES | | | | | | | | STRUCTURED MATRICES | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | order | 100 | 100 | 100 | 50 | 50 | 50 | 25 | 25 | 54 | 57 | 199 |
| | density | 0.05 | 0.03 | 0.02 | 0.10 | 0.05 | 0.02 | 0.20 | 0.10 | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 |
| CJROW | FILL-IN | 1528 | 567 | 192 | 436 | 124 | 4 | 102 | 41 | 118 | 105 | 1236 |
| | MULTN'S | 14950 | 3319 | 683 | 3228 | 535 | 69 | 706 | 181 | 852 | 788 | 5759 |
| CJPROB | FILL-IN | 1571 | 574 | 206 | 442 | 123 | 4 | 106 | 45 | 240 | 156 | 1428 |
| | MULTN'S | 15635 | 3374 | 734 | 3325 | 527 | 69 | 742 | 195 | 1554 | 1037 | 7337 |
| CJR2 | FILL-IN | 1848 | 612 | 222 | 456 | 140 | 5 | 106 | 44 | 167 | 171 | 1711 |
| | MULTN'S | 20816 | 3608 | 762 | 3541 | 628 | 70 | 735 | 191 | 1063 | 1076 | 9044 |
| CJR1 | FILL-IN | 3497 | 1606 | 922 | 812 | 277 | 5 | 174 | 69 | 200 | 204 | 6015 |
| | MULTN'S | 59824 | 15991 | 4507 | 8240 | 1333 | 70 | 1255 | 291 | 1300 | 1260 | 97856 |

Table 3.2. Runs using different local row selection criteria. The results for the random matrices are the average of five runs at each size and density.

| METHOD \ MATRIX | | RANDOM MATRICES | | | | | | | | STRUCTURED MATRICES | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | order | 100 | 100 | 100 | 50 | 50 | 50 | 25 | 25 | 54 | 57 | 199 |
| | density | 0.05 | 0.03 | 0.02 | 0.10 | 0.05 | 0.02 | 0.20 | 0.10 | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 |
| CJROW | | 7.727 | 2.433 | 1.519 | 1.849 | 0.573 | 0.321 | 0.459 | 0.224 | 0.967 | 0.926 | 5.987 |
| CJPROB | | 8.313 | 2.577 | 1.638 | 1.932 | 0.599 | 0.325 | 0.496 | 0.243 | 1.1713 | 0.928 | 6.444 |

Table 3.3. Times of runs of method CJ using proper row update (CJROW) and using probability update (CJPROB). These times are in seconds of 1906A milltime.

| METHOD | MATRIX order / density | RANDOM MATRICES | | | | | | | | STRUCTURED MATRICES | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 100 0.05 | 100 0.03 | 100 0.02 | 50 0.10 | 50 0.05 | 50 0.02 | 25 0.20 | 25 0.10 | 54 CURTIS 1 | 57 WILLOUGHBY 1 | 199 WILLOUGHBY 2 |
| CJROW | FILL-IN | 1528 | 567 | 192 | 436 | 124 | 4 | 102 | 41 | 118 | 105 | 1236 |
| | MULTN'S | 14950 | 3319 | 683 | 3228 | 535 | 69 | 706 | 181 | 852 | 788 | 5759 |
| DJROW | FILL-IN | 1593 | 563 | 190 | 439 | 122 | 4 | 98 | 40 | 118 | 88 | 1402 |
| | MULTN'S | 16318 | 3280 | 680 | 3286 | 526 | 72 | 683 | 183 | 866 | 708 | 7291 |
| GJROW | FILL-IN | 1635 | 563 | 191 | 438 | 123 | 4 | 98 | 41 | 127 | 92 | 1226 |
| | MULTN'S | 16885 | 3280 | 691 | 3274 | 523 | 72 | 681 | 188 | 886 | 725 | 6833 |
| HJROW | FILL-IN | 1998 | 630 | 164 | 501 | 139 | 3 | 127 | 40 | 158 | 83 | 1262 |
| | MULTN'S | 23615 | 4002 | 614 | 3917 | 661 | 71 | 870 | 185 | 1079 | 683 | 6936 |
| NATURAL COLUMN ORDER ROW | FILL-IN | 2204 | 967 | 288 | 594 | 226 | 18 | 161 | 73 | | | |
| | MULTN'S | 25629 | 5659 | 934 | 4697 | 929 | 104 | 1072 | 278 | | | |

Table 3.1. Runs using different a priori column selection criteria. The results for random matrices are the average of five runs at each size and density.

| MATRIX order | density | NZ | CJROW Fill-in | CJROW Multn's | CJROW Time | BEST OF THE 16 A PRIORI METHODS Fill-in | BEST OF THE 16 A PRIORI METHODS Multn's | BEST OF THE 16 A PRIORI METHODS Time | MARKOWITZ Fill-in | MARKOWITZ Multn's | MARKOWITZ Time | % DIFFERENCE IN FINAL DENSITY BETWEEN BEST A PRIORI AND MARKOWITZ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 0.05 | 595 | 1528 | 14950 | 7.727 | 1528 | 14950 | 7.727 | 1280 | 13525 | 12.090 | 13.3 |
| 100 | 0.03 | 397 | 567 | 3319 | 2.433 | 563 | 3280 | 2.659 | 470 | 2834 | 4.312 | 10.7 |
| 100 | 0.02 | 298 | 192 | 683 | 1.519 | 164 | 614 | 1.623 | 129 | 541 | 1.675 | 8.2 |
| 50 | 0.10 | 295 | 436 | 3228 | 1.849 | 436 | 3228 | 1.849 | 390 | 3102 | 2.891 | 6.7 |
| 50 | 0.05 | 172 | 124 | 535 | 0.573 | 122 | 526 | 0.630 | 95 | 455 | 0.801 | 10.1 |
| 50 | 0.02 | 99 | 4 | 69 | 0.321 | 3 | 71 | 0.398 | 1 | 73 | 0.255 | 2.0 |
| 25 | 0.20 | 145 | 102 | 706 | 0.4588 | 98 | 681 | 0.4854 | 98 | 727 | 0.653 | 0.0 |
| 25 | 0.10 | 95 | 41 | 181 | 0.224 | 40 | 183 | 0.239 | 36 | 176 | 0.294 | 3.0 |
| 54 | Curtis 1 | 291 | 118 | 852 | 0.967 | 118 | 852 | 0.967 | 89 | 742 | 1.351 | 7.6 |
| 57 | Will.1 | 281 | 105 | 788 | 0.926 | 83 | 683 | 0.702 | 28 | 495 | 1.002 | 17.8 |
| 199 | Will.2 | 701 | 1236 | 5759 | 5.987 | 1226 | 6833 | 5.8035 | 714 | 3376 | 16.306 | 36.2 |

Table 3.5. A priori methods judged against a common local criterion one. NZ = number of non-zeros in original matrix. The results for the random matrices are the average of five runs at each order and density. The times are in seconds of 1906A milltime.

| MATRIX order | density | NZ | CJ ROW | DJ ROW | GJ ROW | HJ ROW | CJ PROB | DJ PROB | GJ PROB | HJ PROB | CJ R2 | DJ R2 | GJ R2 | HJ R2 | CJ R1 | DJ R1 | GJ R1 | HJ R1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 0.05 | 595 | 1528 | 1593 | 1635 | 1998 | 1571 | 1631 | 1654 | 2003 | 1848 | 2001 | 2029 | 2352 | 3497 | 3633 | 3584 | 3304 |
| 100 | 0.03 | 397 | 567 | 563 | 563 | 630 | 574 | 651 | 649 | 713 | 612 | 780 | 780 | 859 | 1606 | 1568 | 1491 | 1177 |
| 100 | 0.02 | 298 | 192 | 190 | 191 | 164 | 206 | 203 | 205 | 170 | 222 | 231 | 232 | 230 | 922 | 797 | 666 | 495 |
| 50 | 0.10 | 295 | 436 | 439 | 438 | 501 | 442 | 465 | 462 | 535 | 456 | 501 | 500 | 576 | 812 | 838 | 873 | 816 |
| 50 | 0.05 | 172 | 124 | 122 | 123 | 139 | 123 | 128 | 131 | 154 | 140 | 154 | 154 | 195 | 277 | 273 | 291 | 238 |
| 50 | 0.02 | 99 | 4 | 4 | 4 | 3 | 4 | 4 | 4 | 3 | 5 | 5 | 5 | 4 | 5 | 5 | 5 | 3 |
| 25 | 0.20 | 145 | 102 | 98 | 98 | 127 | 106 | 106 | 106 | 142 | 106 | 114 | 114 | 139 | 174 | 162 | 159 | 166 |
| 25 | 0.10 | 95 | 41 | 40 | 41 | 40 | 45 | 43 | 42 | 42 | 44 | 45 | 59 | 41 | 69 | 53 | 60 | 55 |
| 54 | Curtis1 | 291 | 118 | 118 | 127 | 158 | 240 | 242 | 250 | 271 | 167 | 176 | 177 | 191 | 200 | 211 | 208 | 198 |
| 57 | Will.1 | 281 | 105 | 88 | 92 | 83 | 156 | 155 | 158 | 150 | 171 | 185 | 189 | 167 | 204 | 200 | 196 | 185 |
| 199 | Will.2 | 701 | 1236 | 1402 | 1226 | 1262 | 1428 | 1499 | 1436 | 1450 | 1711 | 1843 | 1852 | 1713 | 6015 | 6120 | 5890 | 5575 |

Table 3.4. Fill-in for the 16 a priori methods. NZ = number of non-zeros in the matrix. The results for the random matrices are the average of five runs at each order and density.

## Section 4. Sparsity pivoting based on local criteria.

This section is concerned with the feasibility of obtaining an optimal or near optimal ordering in the sense that an ordering of the rows and columns of the matrix is determined so that the number of non-zeros introduced in the elimination process is kept as low as possible when pivoting down the diagonal of the permuted form. In the last section, it was demonstrated that a priori methods are unlikely to approach this aim and so this section looks more closely at algorithms for achieving such an ordering. These will be based on associating, at each stage in the elimination process, a merit value to every non-zero in the uneliminated part of the matrix. A common choice for this merit value is that used by Markowitz (1957). For element $(i,j)$ of the matrix this gives the merit number

$$MV_{ij} = (r_i-1).(c_j-1) \qquad \ldots\ (4.1)$$

where the $r_i$ and $c_j$ are the row and column counts of chapter two calculated on the rows and columns of the uneliminated part of the matrix. At each stage in the elimination process, the non-zero element with minimum merit value (subject, in practical cases to some numerical tolerance) is chosen as pivot. The merit value of equation (4.1) causes the pivot choice to be that which minimises the maximum number of possible fill-ins at each stage and is very similar to choosing that element which would minimise the number of multiplications at each stage. This would be given (see comments on operation count in chapter one) by using the merit value

$$MV_{ij} = r_i.(c_j-1) + 1 \qquad \ldots\ (4.2)$$

The results in table 4.1 indicate that there is really no difference as to which merit value is chosen, and, for future comparisons, the one due to Markowitz is selected.

Another obvious candidate for the merit value of element $(i,j)$ is the number of elements changing from zero to non-zero if $(i,j)$

were chosen as pivot. This is obviously a more complex criterion than those previously mentioned, and such a selection criterion, based on choosing the element with this minimum merit value at each stage, is equivalent to minimising the local fill-in at each stage. It has been suggested that such a property is convex (Tewarson (1970c)) and will lead to a global minimum given some suitable tie-breaking procedures. Although this is shown not to be the case by examples in this section, it is seen from table 4.2 that this choice of merit value will almost invariably lead to a lower fill-in than that given by Markowitz's criterion. In the same table, however, it is very evident that the time taken for such a local fill-in minimisation ordering would make such a method prohibitively expensive unless the value of saving core storage was rated so highly that this extra time could be considered relatively insignificant. In the next section, some methods are described for significantly reducing this appalling time difference between these two methods.

It is seen, from table 4.2., that the minimum local fill-in algorithm fails to give global minimum fill-in in a matrix arising from a discretisation of Laplace's equation on a cube of side 4 but this is perhaps due to the tie-breaking procedures involved. This might therefore not contradict the hypothesis that local minimum fill-in will, given a certain procedure for breaking ties, produce global minimum fill-in. Experiments were tried on random matrices of orders 10 to 100 to see if a better global fill-in could be obtained by selecting, at some stage in the elimination process, a pivot giving local non-minimum growth. The effect of different tie-breaking procedures was then carried out on any examples giving a lower total fill-in for the latter procedure. In all, over 300 matrices were thus tested and there was none which yielded a counter-example to the hypothesis. However, it was possible to construct the following counter-example in figure 4.1 which is a development of an example in Rose (1971) which depended on a certain tie-breaking procedure for

its validity.

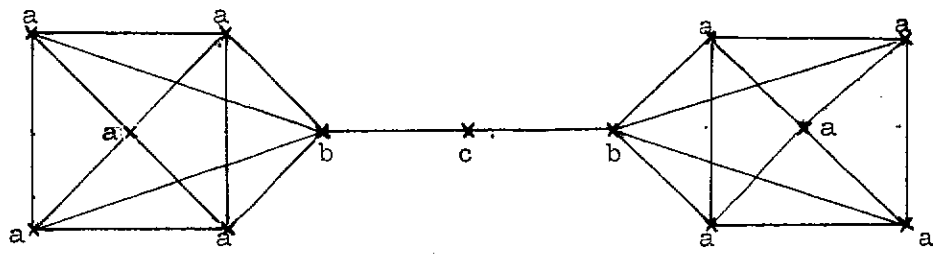



figure 4.1.

In the example of figure 4.1, where each line is equivalent to two directed lines in opposite directions, c has minimum merit value of one, all other points having merit value two. However, if c is chosen as the first pivot, then the total fill-in to result is five. Whereas, if all the points in each of the clusters in the graph (points a) are eliminated first, the total fill-in is only four, thus producing a counter-example to the hypothesis. It should be observed that, even if it is permitted to choose pivots from among the off-diagonal non-zeros in the symmetric adjacency matrix of the graph in figure 4.1, this counter-example still holds good. This example of order 13 is certainly a minimum one for structures of the type shown in figure 4.1 and is believed to be a minimum counter-example to the hypothesis.

Because of this fact that local minimisation of fill-in does not necessarily lead to a global minimisation, other methods have been suggested for choosing a pivot order for the elimination process. Rose (1970,1971) proposes a minimum triangulation algorithm which has some defects which would render it somewhat unsuitable for the purposes of this chapter. It requires an initial triangulation of the graph (see section on graph theory in chapter one), applies only to undirected graphs of symmetric matrices, will generally take an even longer time than the minimum local fill-in algorithm, and guarantees only to produce a triangulation which is minimum only in the sense that no proper subset of the lines in the triangulation form a triangulation themselves. Another approach which would again

work successfully on the example of figure 4.1 is due to Bree (1964). He splits the graph into subgraphs using cut vertex sets (see chapter one) and eliminates within each graph in turn before considering the points in the cut vertex sets. His algorithm will clearly take a long time to implement since, in addition to using minimum local fill-in criteria on the subgraphs, it requires the use of a procedure to find cut vertex sets (Paton (1971)), a lengthy process if the system is large. It is also a non-trivial matter to extend this algorithm of Bree to cover unsymmetric systems.

It is considered that such refinements are not strictly justified in terms of the time required as against the reduction in fill-in on using them. In addition, while these techniques work well on certain structures, they do not always guarantee a global minimum fill-in and often do no better than the method of minimising the local fill-in. An example of a matrix for which these refinements will not help is given in figure 4.2. The unsymmetric matrix of order 22 in that figure was developed from one of the 300 randomly generated matrices mentioned earlier in this section.

If the merit value being used is that of the local fill-in which would be caused were the non-zero used as a pivot, then the numbers on each non-zero of figure 4.2 represent the merit value of the corresponding element. The zeros marked with an 'x' are filled in after the elements (6,6) and (1,20) have been used as pivots in either order. If element (6,6) is eliminated first then clearly there is an additional fill-in in position (1,4) but there is no fill-in in the astericked positions in the column of the pivot. These two astericked positions would however, be filled-in were element (1,20) chosen as pivot first. Thus, the fill-in, if the pivots are chosen in the order (6,6), (1,20), is one less than the fill-in if they are chosen in the reverse order and, in both cases, the structure of the 20×20 matrix left after these eliminations is identical.

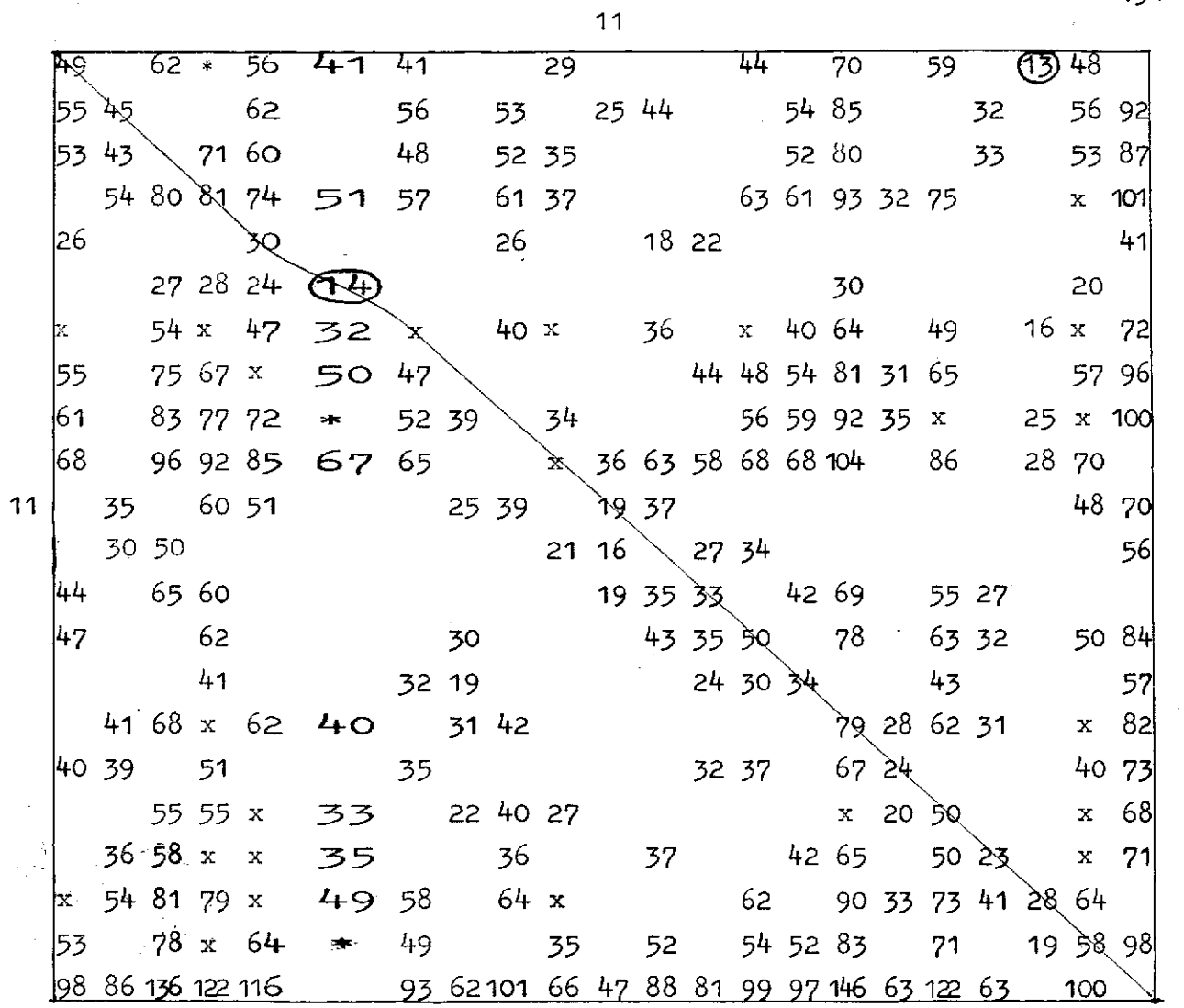



figure 4.2

The example of figure 4.2 constitutes another counter-example to the hypothesis that selecting pivots on the basis of local minimum fill-in will, given a suitable tie breaking procedure, lead to global minimisation of fill-in. In addition, the example of figure 4.2 does not readily admit to a graphical decomposition as required by the methods of Rose (1971) or Bree (1964).

It now remains to consider whether it is worthwhile to spend the extra time involved implementing an algorithm based on minimum local fill-in as opposed to using Markowitz's criterion. It could be claimed that Markowitz's criterion essentially minimises the multiplication count rather than the fill-in and that this is perhaps a more desirable thing to do. It is, however, evident from table 4.2 that the use of Markowitz's criterion as merit value does not even minimise the number of multiplications in the decomposition. In fact.

even in symmetric systems, minimisation of the local multiplication count (for example, Spillers and Hickerson (1971)) does not necessarily minimise the global multiplication count as can be seen in the counter-examples produced by Bertelè and Brioschi (1971). In addition, the example of Rose (1971), given in figure 4.3, illustrates that it is possible to create arbitrarily more fill-in using Markowitz's criterion than using local fill-in minimisation.

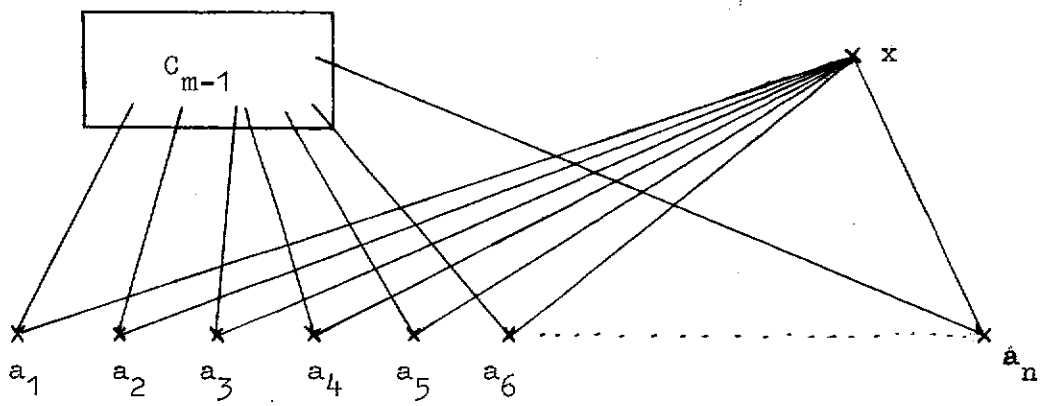



figure 4.3

$C_{m-1}$ is a complete subgraph on m-1 points and it is assumed that $m > n$. Each $a_i$ is adjacent to every point of this subgraph. It is clear that the point x would be chosen using the Markowitz criterion giving a total fill-in of $n(n-1)/2$ whereas, if the $a_i$ are ordered first, (as they would be if local minimisation were employed) the total fill-in would only be m-1.

This example, the preceeding comments, and the results in table 4.2, make it worthwhile to examine the possibility of speeding up the process whereby local minimisation is used as the criterion for selecting the pivots for Gaussian elimination. This speeding up is necessary because of the appalling time differences in the methods of table 4.2 and is the subject of the next section of this chapter.

| MATRIX / METHOD (Local Criterion) | | RANDOM MATRICES | | | | | | | | STRUCTURED MATRICES | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | order | 100 | 100 | 100 | 50 | 50 | 50 | 25 | 25 | 54 | 57 | 199 | 64 |
| | density | 0.05 | 0.03 | 0.02 | 0.20 | 0.10 | 0.05 | 0.20 | 0.10 | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | LAPLACIAN ON CUBE OF SIDE 4 |
| MARKOWITZ | FILL-IN | 1280 | 470 | 129 | 390 | 95 | 1 | 98 | 36 | 89 | 28 | 714 | 652 |
| | MULTN'S | 13525 | 2834 | 541 | 3102 | 455 | 73 | 727 | 176 | 742 | 495 | 3376 | 5117 |
| | TIME | 12.09 | 4.312 | 1.675 | 2.891 | 0.801 | 0.255 | 0.653 | 0.294 | 1.351 | 1.002 | 16.31 | 11.13 |
| MINIMISATION OF LOCAL FILL-IN | FILL-IN | NOT AVAILABLE | | 108 | 340 | 84 | 1 | 70 | 22 | 67 | 11 | 662 | 692 |
| | MULTN'S | TIME TAKEN WAS | | 401 | 2490 | 382 | 71 | 498 | 125 | 666 | 443 | 2984 | 5586 |
| | TIME | MORE THAN 60 secs | | 8.939 | 49.60 | 5.054 | 0.271 | 8.086 | 1.240 | 10.70 | 5.385 | 175.4 | 100.3 |
| % DECREASE IN FINAL NUMBER OF NON-ZEROS WHEN USING FILL-IN MIN. | | | | 5.2 | 4.9 | 4.3 | 0 | 13.0 | 12.0 | 6.1 | 5.8 | 3.9 | - 4.0 |

**Table 4.2** . A comparison of Markowitz's criterion and that of minimising the local fill-in at each stage. The results for random matrices are the average of five runs at each order and density. Times are in seconds of 1906A milltime.

| MATRIX / METHOD (local criteria) | | RANDOM MATRICES | | | | | | | | STRUCTURED MATRICES | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| order | | 100 | 100 | 100 | 50 | 50 | 50 | 25 | 25 | 54 | 57 | 199 |
| density | | 0.05 | 0.03 | 0.02 | 0.10 | 0.05 | 0.02 | 0.20 | 0.10 | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2. |
| MINIMUM MULTIPLICATION COUNT | FILL-IN | 1283 | 462 | 129 | 392 | 95 | 1 | 100 | 34 | 87 | 27 | 716 |
| | MULTN'S | 13597 | 2725 | 542 | 3120 | 453 | 73 | 731 | 170 | 738 | 492 | 3376 |
| MARKOWITZ | FILL-IN | 1280 | 470 | 129 | 390 | 95 | 1 | 98 | 36 | 89 | 28 | 714 |
| | MULTN'S | 13525 | 2834 | 541 | 3102 | 455 | 73 | 727 | 176 | 742 | 495 | 3376 |

Table 4.1. A comparison of fill-ins for two local criteria. Results for random matrices are the average of five runs at each order and density.

## Section 5. The feasibility of minimising the local fill-in.

In order that local minimisation be a feasible technique for ordering a system prior to Gaussian elimination it is necessary that some methods for speeding up the selection process be adopted. This is the subject of this section.

In the Markowitz method, it is clearly possible to update the two $1 \times n$ row and column count vectors at each stage in the elimination and from there recalculate merit values for the remaining non-zeros. This process is, however, much costlier if the associated merit value is the actual amount of fill-in which the element would cause if selected as a pivot. In terms of an approximate operation count, the amount of computation required for the two methods is given in the equations

$$\text{Operations}_{\text{Markowitz}} = 2r\tau \qquad \ldots\ (5.1)$$

$$\text{Operations}_{\text{Min-fill}} = 2r^2\tau \qquad \ldots\ (5.2)$$

where $r$ is the average number of non-zeros in a row or column and $\tau$ is the total number of non-zeros.

If the update of the last paragraph is used then the number of operations required by Markowitz at each stage is given by the equation

$$\text{Operations}_{\text{M update}_1} = 2r + 3\tau \qquad \ldots\ (5.3)$$

Of course, it is possible to short circuit this procedure in two ways. The first is to hold an integer array of merit values and only update those affected by the previous stage in the elimination. The second is to develop means whereby only a certain subset of the $\tau$ non-zeros are examined as possible pivots at each stage. The first short circuit reduces the number of operations required by Markowitz to that given by the equation

$$\text{Operations}_{\text{M update}_2} = O(r^2) \qquad \ldots\ (5.4)$$

where $r$ is again the average number of non-zeros in a row, and the constant of this dominant term is about 6 depending on the particular implementation. The second short circuit can be implemented by carrying a running minimum count only considering those non-zeros which lie in columns whose column count is less than this running minimum. This latter device is used by Curtis and Reid (1971b, 1971c, 1971d).

In order to reduce the very large operation count in equation (5.2) similar tactics have to be invoked in the case of local minimisation. If a merit value array is kept then it is possible to consider the calculation of new merit values for only the elements whose merit values are changed by the elimination of a pivot. However, in this case, rather more than the $r^2$ elements of equation (5.4) have to be considered. This is seen by looking at figure 5.1.

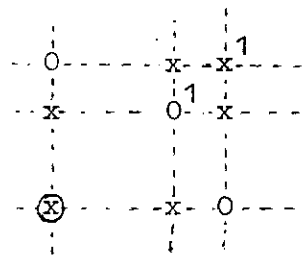



figure 5.1.

Elimination of the circled element in figure 5.1 causes the merit value of element $x^1$ to be reduced by one due to fill-in in position $0^1$. However, the element $x^1$ is not one of the $r^2$ elements in a row (column) which has a non-zero in the pivot column (row). In all, as will be seen from the subsequent discussion, $r^3$ elements could have their merit values affected by such an elimination thus yielding the number of operations for such an update as that in the equation

$$\text{Operations}_{\text{Min update}_1} = O(r^5) \qquad \ldots\ldots (5.5)$$

where the constant of this dominant term will depend on the implementation but should be about 2. It is certainly evident

that although this will, in general, be a marked reduction on the count of equation (5.2) it is still high in comparison with the operation count given in equation (5.4). An attempt to systematise this updating procedure as well as to reduce this operation count given in equation (5.5) is now described in some detail.

Update of merit value array for local minimum fill-in

Step 1.

Look along pivot row for non-zeros. In the figures below ⓧ represents the pivot element while ☒ represents an element which is filled in when this pivot is used in Gaussian elimination.

Step 2.

Look at the column of such non-zeros in the pivot row in conjunction with the pivot column to determine fill-ins. Occurrences of this kind are marked by an oval ring in figure 5.2.

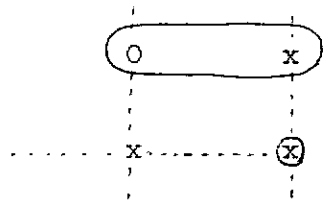



figure 5.2

Concentration is now focussed on this filled-in element (for example, element O of figure 5.2)

Step 3.

Look along row of filled-in element to find non-zeros not in the pivot column. Then check the column of this non-zero for events like those marked by an oval in figure 5.3

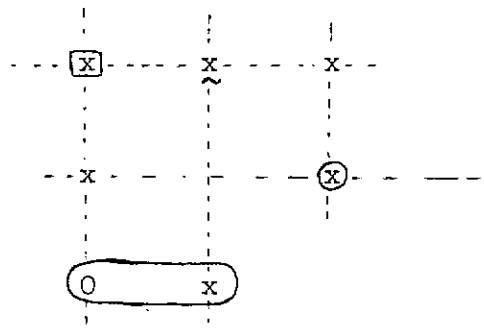



figure 5.3

For such events the merit value of element $\underset{\sim}{x}$ is increased by one.

Step 4.

Look along the column of filled-in element to find any non-zeros not in the pivot row. The rows of these elements are then examined for events like those marked by oval lines in figure 5.4.

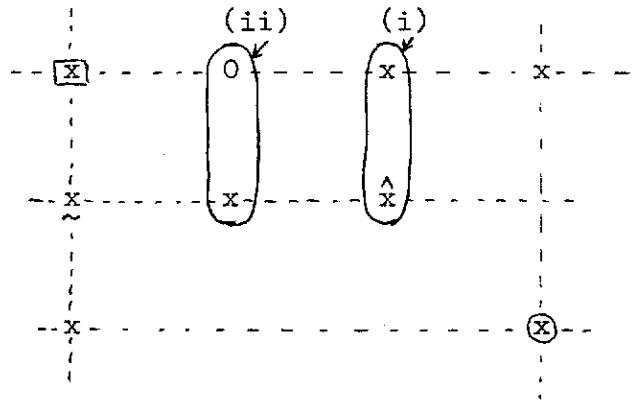



figure 5.4

For events marked (i), the merit value of $\hat{x}$ is reduced by one.
For events marked (ii), the merit value of $\underset{\sim}{x}$ is increased by one.

The total effect on the merit values of matrix elements due to the filled in element [x] has now been evaluated.

Step 5.

At the same time as the pass in step 2, the event shown in figure 5.5 can be looked for.

A subsequent scan of the row of $\underset{\sim}{x}$ in figure 5.5 is shown in figure 5.6 and may yield events like that circled by an oval in that figure.

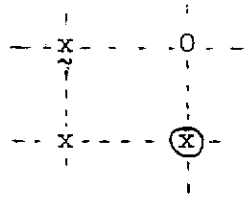



figure 5.5

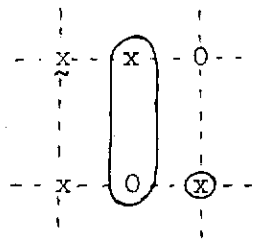



figure 5.6

In each such case, the merit value of the element $\underset{\sim}{x}$ is reduced by one.

Step 6.

A separate pass is now done along the column of the pivot element looking for non-zeros. A scan is then made along the row of such non-zeros for other non-zeros (for example, $\underset{\sim}{x}$ in figure 5.7) not in the pivot column. A final scan is then made down the column of such a non-zero for events of the kind shown in figure 5.7.

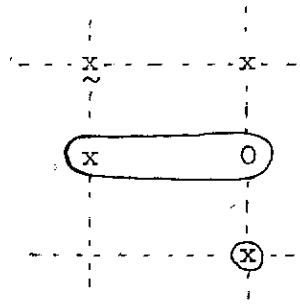



figure 5.7

If such an event occurs, then the merit value of element $\underset{\sim}{x}$ is reduced by one.

Steps 5 and 6 make all the necessary changes due to the elimination of the pivot row and column from the matrix.

The merit values of the added elements are calculated from scratch. It makes little difference whether these are evaluated at the end of the procedure or whenever they are filled in in step 2.

This method has been successfully programmed on the 1906A and is given as a procedure in the appendices.

A rough operation count for the updating method can be evaluated as follows. Step 1 takes r operations and within it steps 2 to 5 take place. Steps 2 and 5 between them take $\sim 2r$ operations and within step 2, steps 3 and 4 take place, step 3 taking about $2r^2$ operations while step 4 takes about the same. Step 6 can be conducted in about $2r^2$ operations giving a total approximate operation count for the procedure as that in equation (5.6).

$$\text{Operations}_{\text{Min update}_2} = 4r^4 + O(r^2) \qquad \ldots\ldots (5.6)$$

This is generally a saving over the operation counts in equations (5.2) and (5.5) and is more comparable with the count given in

equation (5.4).

The results of several runs are shown in table 5.1. The reason for the difference in times being larger than might be expected from equations (5.4) and (5.6) is due to the much larger overheads involved in calculating the merit values for the $\tau$ elements at the beginning of the procedure.

Finally, a method has been devised to effectively perform a short circuit similar to the second short circuit mentioned in connection with the use of Markowitz's criterion. This method will be called the method of partial lists. It is now described in general terms as follows.

Method of partial lists.

Let S be a list of values $s_i$ , i = 1,$\tau$, and allow the list to be dynamic in the sense that elements may be added to or removed from the list as the process continues. Consider a list (partial list) of a subset of this set selected by means of some criterion function on the original list. This subset will be called PL.

Then, consider a process which selects an element of S on the basis of the same criterion function. This element will be in the list PL and so would be selected by a search on this subset.

After the selection it is assumed that the values of some of the $s_i$ will change. The values of the $s_i$ thus changing are then examined to see if they would have been included in PL using the criterion function had they had that value before the elimination. If so, they are added to the list PL and the process continues with a further search of the new PL by means of the criterion function.

If the list PL is exhausted before sufficient selections are made then a new list PL has to be generated from S using the criterion function before the process can continue. This is called a regeneration of the partial list.

The above process continues until the required number of elements

of S have been selected in the order determined by the criterion function.

The procedure can be easily applied to the case being considered. Here, the list S is a list of merit values of the $\tau$ non-zeros, the criterion function on the list chooses i such that $s_i$ has the minimum value of all elements of the list, the values are updated according to the procedure mentioned previously, and the number of selections to be made from list S is n, the number of pivots required.

If no partial list structure exists then an approximate count for the number of table lookups to establish the full pivot set is given by the equation

$$N_1 = n\tau \qquad \text{.....(5.7)}$$

while, if the partial list PL has length l and is regenerated m times, then the number of table lookups is given by the equation

$$N_2 = l\, m\tau + n\, l \qquad \text{.... (5.8)}$$

For a suitable choice of l, dependent on the size and density of the matrix under consideration, $N_2$ could be less than $N_1$.

This improvement to the local minimisation algorithm by using partial list techniques is shown in table 5.2 for various lengths of partial list and for the three structured matrices used throughout this thesis. When it is borne in mind that a substantial part of the total computing time is spent in generating the original $\tau$ merit values, the savings made by using a well-suited partial list are not insignificant.

Thus, from the results of tables 5.1 and 5.2, it is seen that the feasibility of using a minimum local fill-in criterion for selecting pivots has become real and the method is more competitive with the Markowitz procedure than might be first envisaged. It is really a question as to whether the almost guaranteed saving in storage justifies the extra expenditure in time involved.

| METHOD \ MATRIX | RANDOM MATRICES | | | | | | STRUCTURED MATRICES | | |
|---|---|---|---|---|---|---|---|---|---|
| order | 100 | 50 | 50 | 50 | 25 | 25 | 54 | 57 | 199 |
| density | 0.02 | 0.10 | 0.05 | 0.02 | 0.20 | 0.10 | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 |
| OLD MINIMISATION | 8.939 | 49.60 | 5.054 | 0.271 | 8.086 | 1.240 | 10.701 | 5.385 | 175.4 |
| NEW VERSION USING UPDATES | 3.954 | 15.75 | 2.850 | 0.283 | 5.372 | 0.654 | 4.654 | 2.476 | 58.73 |
| MARKOWITZ | 1.675 | 2.891 | 0.801 | 0.255 | 0.653 | 0.294 | 1.351 | 1.002 | 16.31 |

Table 5.1. A comparison of times (in seconds of 1906A milltime) for the new form of local minimisation updating scheme described in this section with times for the old local minimisation scheme and a method using Markowitz's criterion.

| size of partial list \ MATRIX | 54 x 54 CURTIS 1 | 57 x 57 WILLOUGHBY 1 | 199 x 199 WILLOUGHBY 2 |
|---|---|---|---|
| 5 | 4.118 | 2.151 | 57.324 |
| 10 | 4.120 | 2.310 | 52.766 |
| 15 | 4.580 | 2.400 | 61.021 |
| 20 | 4.732 | 2.532 | 58.168 |
| 40 | | | 55.260 |
| 60 | | | 59.560 |
| 100 | | | 65.505 |
| 200 | | | 74.294 |

Table 5.2. A comparison of times when using the partial list scheme of this section. Times are in seconds of 1906A milltime.

## Section 6. A sparse matrix package.

The procedures described in this section were written for use on the 1906A in the spring of 1971. Although they all have been tested successfully on several examples, no claim is made that the procedures are optimal. They do, in fact, represent the first steps towards writing a sparse matrix package for the 1906A for use in the OXLIB1900 library and subsequent inclusion in the NAG library (the Nottingham algorithms group was a group initially formed to develop a numerical library for use on the ICL 1906A but the scope of the library has since been extended to cover many other university systems). Further development of this package was abandoned at the end of 1971 when it was decided to adapt the well-tested algorithm of Curtis and Reid (1971b) which was already in the Harwell subroutine library. This subroutine has now been translated into Algol and 1906A Fortran and is currently in OXLIB1900 awaiting acceptance to the next mark of the NAG library.

A short description of the author's algorithm, even although it is far from optimal, has been included here for two reasons. The first is that it illustrates some of the points made in this chapter and, in addition, demonstrates that even a relatively inefficient algorithm utilising sparsity techniques produces accurate results much faster, and using considerably less core, than the normally available library routines.

Listings of the procedures are given in the appendices and a pilot program for solving a set of linear equations is also included to illustrate the procedure calls.

The three processes of the solution of linear equations, evaluation of the determinant, and inversion of a matrix (procedures SOLN, DETERM, and INVERT) are all preceeded by the decomposition of the matrix to the product of lower and upper triangular matrices using Gaussian elimination with pivoting (procedure DECOMP). The matrix is stored by columns in a linked list with pointers to the

head of each column. With each non-zero is associated three items : the value of the non-zero element, its row index, and a pointer to the next element of the column (zero if it is the last one). The columns are sorted a priori in order of ascending $\delta_j$ (see section 3) and, at each stage, the pivot selected is that which has minimum row count in the reduced matrix. This selection is subject to a tolerance which is set in the procedure but could equally well be made an input parameter. This method of achieving some small guard against numerical instability is that which has been used in the past in some linear programming codes (see Proc. of IBM Symposium, Willoughby (1969)) and is suitable for scaled fairly well-conditioned matrices - a relatively large subset of the cases considered. It is simple to put in greater safeguards and checks by, for example, making sure that the largest multiplier is well-bounded in the same sense as the term is used in Curtis and Reid (1971b). This check would be inserted at the same point in the program as the tolerance check is at present.

The solution procedure involves a forward elimination using the factored form of the lower triangular matrix followed by a back substitution using the upper triangular matrix again held in factored form. The inversion procedure merely involves the solution of successive equations with columns of the unit matrix as the right-hand side, while the evaluation of the determinant consists in multiplying the values of the pivots chosen in the decomposition phase.

The routine to evaluate the determinant has been tried on several test matrices of orders between six and twenty and has been found to be working correctly to machine accuracy on these examples. The inversion routine has been tested on a similar set of matrices and has, in addition, been tested on a number of matrices of order 100 whose non-zero elements were generated by a random number generator and whose density was about 3%. The inverse thus obtained was then multiplied by the original matrix and the result compared to the

identity matrix. In each case, the product differed from the identity matrix by at most $10^{-9}$ in some elements. The solution procedure was tested against all of the above examples and was found to give good results. In addition, it was used to solve equations arising from various finite difference formulae applied to ordinary differential equations and gave answers as accurate as the formulae would allow. The author is indebted to L.J. Hazlewood (Oxford Computing Laboratory) for supplying these examples. Finally the procedures were used to decompose and solve a set of equations arising from a cost model in the agricultural division of I.C.I. The order of this ag.cost model matrix was 560 and the number of non-zeros was 2055 only increasing to 2467 after the decomposition. The results of runs on several right-hand sides again confirmed the validity of the algorithm. This example was supplied by W. Phillips (Oxford Computing Laboratory).

Tinney (1969a) mentions the great gains that can be made using sparsity techniques over those normally adopted for full matrices even if these sparsity techniques are not optimal. This is borne out by the results of the above experiments. For example, the average time taken by this algorithm to decompose a matrix of order 100 and to solve one right-hand side is 10 seconds, the time for subsequent right-hand sides being about 1 second. This is over 14 times better than routine F04AAA in the NAG library which solves the equations using Crout reduction with pivoting. The time taken to solve the ag. cost model using this sparsity technique (one right hand side and decomposition) was 50 seconds. The times given in the NAG manual for the decomposition and solution of systems of various orders indicate that the dependence of time on order is given by the equation

$$t(n) = (1/8\ n^3 + \tfrac{3}{4}\ n^2) \times 10^{-3} \text{ seconds} \qquad \ldots. (5.1)$$

If this dependence held for $n$ equal to 560 then the similar computation using procedure F04AAA would take over six hours computing

time an increase of over 400 times that using the sparsity method. This is, of course, on the assumption that the matrix could be held in the core of the computer. This would require 627 K of 24 bit 1906A words (each real is stored in two words in the 1906A) while the present daytime limit at Oxford is only 60 K (June 1972). The sparsity procedures require only 24 K of store to perform the entire calculation.

The suboptimality of this sparsity method is seen when it is compared with the time that the translated version of the Curtis and Reid algorithm takes on the 560 x 560 ag. cost model matrix. These are 9.8 seconds for decomposition and 0.2 seconds to solve a right hand side. This is over 2,000 times faster than the projected F04AAA procedure !!

# CHAPTER 6.

## A PROBABILISTIC STUDY OF FILL-IN.

## Section 1. Introduction.

This chapter studies the fill-in properties of Gaussian elimination on random matrices. Graph theoretical ideas are introduced and the problem is examined using the concept of random graphs.

This concept of using random graphs is not a new one. They are used as a tool in the development of existence theorems in graph theory (Erdös (1967)) and some work has been done into the structure of a graph where the number of lines is a function of the number of points, the analysis holding in the limit of the number of points tending to infinity (Palásti (1966,1970)). Ogilvie (1968) has shown that these asymptotic results hold where the graph has only a small number of vertices (about 10) but none of this work is directly concerned with matrix decomposition or indeed with the adjacency matrix of the graph. Heap (1966) has extended the notion of random graphs to examine properties of their adjacency matrices. However, his results on matrix reducibility are only valid for comparatively dense matrices (probability density $\gtrsim 0.15$). A serious attempt to tackle the problem of this chapter has been made by Hsieh and Ghausi (1971b) where they have tried to make theoretical predictions on the results of Brayton et al (1970). Their work has, however, several serious defects. Although they make more allowance for interaction than the discussion in section 2 of this chapter, their formulae only include correlation between elements in the same column ignoring the correlation which exists between elements in the same row. In addition, some of their formulae break down if the density of the matrix increases above 20% and their correlation factor, H, is chosen empirically to obtain a fit with Brayton's data rather than determined theoretically. The impetus for writing this chapter came from remarks by Robert Brayton in the panel at the end of the IBM symposium (Rose and Willoughby (1972)) when he stated that the above problem, although itself not perhaps of devastating importance, had consumed vast

amounts of manpower in its attempted solution. This chapter is dedicated to releasing these manhours for more productive work.

Section 2 of this chapter makes an 'obvious' initial approach to the problem, indicates some of the difficulties involved in its solution, and gives an upper bound to the quantity it is desired to evaluate. In the following section, the abovementioned notion of random graphs is introduced and the elimination problem is interpreted in graph theoretic terms. This interpretation is used in section 4 to derive formulae relating to the fill-in caused by Gaussian elimination on random graphs. It is seen, in section 5, that these formulae give results which are in good agreement with experimental data generated by the author. The concluding section gives a consequence of the results of this chapter and indicates other problems which may be tackled by such techniques.

## Section 2. A statement of the problem and an initial approach.

The problem to be considered in this chapter is that of Gaussian elimination performed on a matrix which is characterised by having all its diagonal elements non-zero, and with all the off-diagonal elements having the same probability $p$, say, of being non-zero. The elimination on this random matrix is performed by pivoting down the diagonal in the natural order, and the interest is centred on the probable number of fill-ins that occur. This number will, of course, be some function of $n$, the order of the matrix, and $p$ only.

Because of the nature of the elimination process, it is immediately seen that a probability $P_i$ can be ascribed to the likelihood of elements in the $i$ th row (column) of $U$ ($L$) being non-zero after the decomposition has taken place. This $P_i$ will be the same for all elements in the same row (column).

It is perhaps worthwhile to indicate a natural way to tackle this problem and, by showing its failure, indicate the way in which many 'simple' solutions fail to answer this problem.

$P_1$ will always be equal to $p$, the initial probability of the off-diagonal matrix elements being non-zero.

An element in the second row of $U$ (col.of L) will be non-zero if:

(i) it is initially non-zero : probability $p$.

(ii) it is initially zero and the elements at the head of its row and column in the original matrix are both non-zero thus causing a fill-in to occur. : probability $(1-p)p^2$, since the events are mutually independent.

It thus follows that the relation

$$P_2 = p + (1-p)p^2$$

is true.

Now consider the following argument. (This is done for $U$, but

the same argument follows for L).

An element in the third row of U will be non-zero if it is either non-zero initially or after the first elimination stage (probability $P_2$), or if it is zero then, but is filled in at the second stage of the elimination (probability $(1-P_2)P_2^2$), giving the equation

$$P_3 = P_2 + (1-P_2)P_2^2.$$

and, by a similar argument, the following recurrence relation holds for $P_i$ $(i=1,\ldots n-1)$.

$$P_{i+1} = P_i + (1-P_i)P_i^2 \qquad P_1 = p \qquad \ldots\ (2.1)$$

That this result is false is obvious experimentally since it is observed that the $P_i$ given by equation (2.1) are always greater than those obtained by empirical tests on a set of generated random matrices (see section 5). It is now indicated why the calculation for $P_3$ as given by equation (2.1) is wrong and then a theorem regarding the $P_i$ generated by equation (2.1) is stated and proved.

The reason why equation (2.1) fails to give the correct value for $P_3$ is as follows. Figure 2.1 should be consulted.

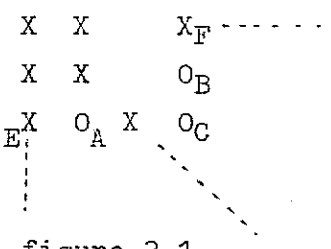



figure 2.1

It is true to say that the probability of element C being zero after the first stage is $(1-P_2)$ and that the probability of A and B both being non-zero after the first stage is $P_2^2$ (since events are independent), but it is certainly not true to claim that these two events are independent as can be seen in figure 2.1, where both the $(1-P_2)$ and the $P_2^2$ term depend on the two elements E and F. Thus, if

Prob (C is zero after first stage) $= (1-P_2)$

and Prob (A and B non-zero after first stage) $= P_2^2$

then

Prob (C zero and A and B non-zero after the first stage) $\neq P_2^2(1-P_2)$.

This argument is formalised in the following theorem.

Theorem 2.1

If $\bar{P}_i$ is defined by the relation

$$\bar{P}_{i+1} = \bar{P}_i + (1-\bar{P}_i)\bar{P}_i^2 \qquad \bar{P}_1 = p$$

and if $P_i$ is the probability of an element in row i of U (col i of L) being non-zero after the decomposition then the $\bar{P}_i$ form an upper bound for the $P_i$.

In fact,
$$\begin{aligned} \bar{P}_i &\geq P_i \qquad (1 \leq i \leq 2) \\ \bar{P}_i &> P_i \qquad (3 \leq i \leq n-1) \end{aligned} \qquad \ldots\ (2.2)$$

Proof:

Certainly,

$$\bar{P}_i \geq P_i \qquad i = 1,2$$

and relation (2.2) is assumed to hold for all $i \leq r \leq n-1$, then,

$\bar{P}_{r+1}$ is given by the relation,

$$\bar{P}_{r+1} = \bar{P}_r + (1-\bar{P}_r)\bar{P}_r^2$$

and since,

$x+(1-x)x^2$ is a strictly increasing function of x, for $x \in [0,1]$,

the induction hypothesis gives the equation,

$$\bar{P}_{r+1} \geq P_r + (1-P_r)P_r^2$$

Now, since

$P_{r+1}$ = Prob { element in the r + 1 th row of U (col of L) is filled in after r th stage}

= Prob { element in the r th row of U (col of L) is filled in after (r-1)th stage } + Prob { element is filled in at r th stage given that it is zero at (r-1) th one }

$$P_{r+1} \leq P_r + (1-P_r)\, P_r^2$$

by theorem on conditional probabilities (see, for example, Feller (1968)) and fact that

Prob { element is zero at (r-1) th stage } $= (1-P_r)$

Prob { element is filled in at r th stage } $= P_r^2$

and so,

$$\bar{P}_{r+1} \geq P_{r+1}$$

and the result follows by induction.

Since, $\bar{P}_3 > P_3$ it is true that

$$\bar{P}_i > P_i \qquad i = 3, \ldots n-1$$

because of the strict monotonicity of the function $x+(1-x)x^2$.

## Section 3. A graph theoretic description of the problem.

In this section and the following one, directed graphs as defined by Harary, Norman and Cartwright (1965) are used to analyse the fill-in problem. The terminology used is identical to theirs, the only additional term being that of 'linked'. Point a is linked to point b in a directed graph if there exists a directed line of the graph from a to b. A point a is connected to a point b if there exists a directed path in the graph from a to b. Thus linked points are connected but not vice-versa.

If a global view is taken of the problem, it is natural to ask what possible sequence of events would cause the element $a_{ij}$ to be filled in. By the discussion of the last section, it is possible, without loss of generality, to consider the case $a_{i,i+1}$. Obviously only the first $i-1$ stages of the elimination could cause fill-in to this position in the matrix and so only these need be considered in the following discussion.

A random graph on n points is defined to be a directed graph on n points where each line has a probability p ($p \ll 1$, in general) of existing, this probability being the same for all the lines. This directed graph is seen to be none other than the adjacency matrix (Harary (1962b)) for the random matrix described in the first section of this chapter.

If the theory of graphs in the elimination process is examined (for example, Tewarson (1967a), Harary (1962a) and Parter (1961), it is seen that one point will be linked to another point in the graphs of subsequent reduced matrices if there exists a path in the initial graph connecting that particular point to the other which does not pass through an as yet uneliminated point (the term 'legal' path is used to describe this and is attributed to Karp (Willoughby (1971c))).

Since the addition of a line in the graphs of the reduced matrices is in 1-1 correspondence with the fill-in of zeros in the original matrix, the following theorem holds.

Theorem 3.1.

The probability of the element in the (i,i+1)th position in the matrix being non-zero after matrix reduction is equal to the probability that there exists a path from the point i to the point i+1 in a random graph on i+1 points if the probability of any line existing in the random graph is the same as the probability of an off-diagonal element of the matrix being non-zero.

## Section 4. The derivation of the formula.

Let the random graph on $n$ vertices with probability $p$ of any edge existing be denoted by $G(n,p)$. Now let $P_x(n,p)$ be the probability that a point $x$ is connected by a directed path to every other point in the graph $G(n,p)$. Thus, for example, $P_x(1,p) = 1$ and $P_x(2,p) = p$.

The following lemma is now stated and proved.

**Lemma 4.1.**

If $P_x(n,p)$ is the probability that $x$ is connected to every other point in the random graph $G(n,p)$, then the relation

$$P_x(n,p) = 1 - \sum_{k=1}^{n-1} {}^{n-1}C_{k-1}\, P_x(k,p).(1-p)^{k(n-k)}$$

holds for any $n$ and $p$.

**Proof.**

Let the points of the graph be partitioned into two subsets, a $k$-set containing $k$ points and an $(n-k)$-set containing the remaining $(n-k)$ points and choose the $k$ points such that $x$ is one of them. Then the probability that $x$ is connected to all the other points in the $k$-set but to none of the other points is given by the formula

$$P_x(k,p).(1-p)^{k(n-k)}$$

the second part of the formula being present since it is necessary that no lines exist from the points of the $k$-set to the points of the $(n-k)$-set. It is possible to select ${}^{n-1}C_k$ subsets from the points of $G(n,p)$ such that $x$ is always in the $k$-set. Furthermore, since the events of $x$ being connected to all points of the $k$-set but none of the points in the $(n-k)$-set are independent for different selections of $k$-sets, the probability of $x$ being connected to $k-1$ other points in the graph $G(n,p)$ is given by the formula

$${}^{n}C_{k-1}\, P_x(k,p).(1-p)^{k(n-k)}$$

Now, the probability of $x$ being connected to $n-2$ or less points

in the graph is given by the formula

$$\sum_{k=1}^{n-1} {}^{n}C_{k-1}\, P_x(k,p)(1-p)^{k(n-k)}$$

since the events of $x$ being connected to $k-1$ and only $k-1$ points are independent as $k$ varies from 1 to $n-1$.

But this is the probability that $x$ is not connected to all the other points of the graph and so the lemma has been proved.

The idea of connectedness can now be extended in the following manner. If $P_{xy}(n,p)$ is the probability that there exists a directed path from $x$ to $y$ in $G(n,p)$ then the following lemma holds.

Lemma 4.2.

If $P_{xy}(n,p)$ is the probability that $x$ is connected to $y$ in the random graph $G(n,p)$ then

$$P_{xy}(n,p) = 1 - \sum_{k=1}^{n-1} {}^{n-2}C_{k-1}\, P_x(k,p)(1-p)^{k(n-k)}$$

where

the $P_x(k,p)$ have the same meaning as in lemma 4.1.

Proof.

If, as in the proof of lemma 4.1, the points of the graph are thought of as being partitioned into two subsets, a $k$-set containing $x$, and an $(n-k)$-set containing $y$, then the probability that $x$ is connected to every point in the $k$-set but to none in the $(n-k)$-set (including $y$) is

$$P_x(k,p).(1-p)^{k(n-k)}$$

as before.

But this time the $k-1$ other points of the $k$-set must only be selected from the $(n-2)$ points of the graph omitting $x$ and $y$, and so the probability that $x$ is connected to any $k-1$ points of the graph and only $(k-1)$ points and is, in addition, not connected to $y$ is given by the formula

$$^{n-2}C_{k-1}\ P_x(k,p).(1-p)^{k(n-k)}$$

and so the probability that x is connected to (n-2) other points or less but is not connected to y is given by the formula

$$\sum_{k=1}^{n-1} {}^{n-2}C_{k-1}.P_x(k,p).(1-p)^{k(n-k)}$$

and, since this is just the probability that x is not connected to y in G(n,p), the lemma has now been proved.

The following theorem follows immediately from theorem 3.1 and lemma 4.1 and 4.2.

Theorem 4.1.

If $P_i$ is the probability that an element in the i th row of U (col of L) is non-zero after the L/U decomposition of a random matrix, as defined in this chapter, then the following relation holds.

$$P_i = 1 - \sum_{k=1}^{i} {}^{i-1}C_{k-1}.H_k(1-p)^{k(i+1-k)} \quad \ldots\ (4.1)$$

where the $H_k$ are given by the equation

$$H_k = 1 - \sum_{j=1}^{k-1} {}^{k-1}C_{j-1}.H_j.(1-p)^{j(k-j)} \quad \ldots\ (4.2)$$

The next theorem then follows immediately.

Theorem 4.2.

The expected value for the total number of non-zeros in the L/U decomposition of a random matrix, as in theorem 4.1, is given by the formula

$$2.\sum_{i=1}^{n-1} (n-i)P_i + n \quad \ldots\ (4.3)$$

where the $P_i$ are given by equations (4.1) and (4.2) and the unit non-zeros in the diagonal of U are excluded since they are not stored.

## Section 5. Experimental results.

In order to examine the defects in using the upper bound approximation of equation (2.1) as an estimate for the probability density of the LU decomposition of a random matrix, the results of computing the recurrence relation for various starting values of $p$ were compared with the probabilities obtained from equations (4.1) and (4.2). These results are shown in graphs 5.1 to 5.4 for values of $p$ equal to 0.03, 0.06, 0.10 and 0.50 respectively. It is seen immediately from these results that the $\overline{P}_i$ generated from equation (2.1) give an upper bound to the true value for the probability density. The strict inequality indicated by theorem 2.1 is not always evident because of the scale of the graph but the computed results from which the graph was obtained do show such an inequality. For the lower value of $p$ in graph 5.1 it is seen that the difference only becomes evident as $i$ increases in size. For the middle values of $p$ in graphs 5.2 and 5.3 the difference between the upper bound for $P_i$ and its true value becomes evident at lower values of $i$ while by the time $p$ is as large as 0.5, the effect is nearly dwarfed by the increase in size of $P_i$ and $\overline{P}_i$ for increasing $i$.

The effect of using these two sets of values for the probability on the estimate of fill-in to a random matrix is shown in table 5.1. In both cases, equation (4.3) was used to calculate the values given in the appropriate columns. In order that the theoretical predictions of this chapter might be compared with experiment, some runs were done performing Gaussian elimination on random matrices, twenty runs being made at each value of order and density. The average number of non-zeros in the decomposed form and the standard deviation from this mean were calculated and the results are shown in the appropriate columns of the table. It is seen that, in all cases, the number of non-zeros predicted by the theory is in very close

agreement with that obtained from the runs on random matrices, and the predicted value, in every case, falls well within the distribution peak defined by the experimental standard deviation. As expected, the use of the upper bound for $P_i$ gives an overestimate of the fill-in which is particularly marked when $n$ is high, except for the case $p = 0.5$ when the fill-in in both cases is almost total.

An examination of the results in the paper of Brayton et al (1970) indicates that most of his experiments were performed with random matrices of a higher order and lower density than the ones used in this chapter. In addition to the core storage restrictions for large systems (there is also a limitation if the matrix is held in packed form because of the amount of fill-in during the elimination process), another difficulty was encountered in the calculation of fill-in to matrices of low initial density. This difficulty was caused by rounding errors in the computation of $P_i$ from equations (4.1) and (4.2). These errors are introduced by the subtraction of the sum in equation (4.2) from 1. This particular error is relatively a very large one since the value of the summation is very close to 1. For example, if it is assumed that the computer can store up to 10 decimal digits, then a summation of value $1-10^{-7}$ will give an error in $H_k$ in the third significant figure; and summation values of $1-10^{-20}$ are obtained if $p = 0.01$!! The use of double precision merely delays the catastrophe.

In an attempt to avoid this calamity, the author has developed a set of procedures to deal with the calculation of $P_i$ from equations (4.1) and (4.2) in rational arithmetic. These procedures are given in the appendices and work to infinite precision (to within the core size of the machine). They are annotated fully in the appendices and need not be described further here. J.H. Griesmer of IBM has also computed values for $H_k$ on a machine at Yorktown Heights using a

rational arithmetic package in a system called SCRATCHPAD (Griesmer and Jenks (1971)). His results have shown exact agreement with those calculated by the procedures in the appendices. Unfortunately, both the IBM system and the procedures in the appendix take a long time to evaluate $H_k$ for large $k$ (because at stage $k$, $k-1$ infinite precision terms have to be calculated for the summation in equation (4.2)), and so it is impractical to try to test the results in Brayton et al (1970) for low density and high $n$. For $p = 0.01$, the infinite precision procedures give a value for $P_{30}$ equal to 0.13920 compared with the upper bound of 0.13953 from equation (2.1). Thus, it is evident that $i$ has to be quite large to obtain any appreciable variation from this upper bound as is seen in the figures for fill-in in a matrix of order 100 in table 5.1.

## Section 6. Conclusions and a possible consequence.

This chapter has indicated some of the power of graph theory in the analysis of large sparse random systems. It is quite probable that other problems concerning singularity, reducibility, and other elimination schemes can be tackled in a similar fashion. One thing which the results of this chapter do indicate is the need for sparsity pivoting in Gaussian elimination. The fill-in of a matrix from an average of 3 off-diagonal non-zeros per row to more than 20 is rather drastic and would be reaching the limit of advisability of packed storage were it much larger.

A possible consequence of the results of this chapter might be to obtain some estimate of the possible error introduced in the elimination process. The error analysis of Wilkinson deals in terms of upper bounds rather than estimates of errors and his factor of $n^2 2^{-2t}$ (Wilkinson (1965), p.248) is not very helpful for large sparse systems. It is evident from these calculations of Wilkinson that the determining factor for these bounds is the number of operations being performed on the various positions of the matrix.

If it is possible to calculate a probablistic estimate of this number of operations, then it is feasible that a more useful estimate of the actual error involved in the solution of sparse matrix equations could be obtained than the crude upper bound given above.

The following theorem is a first step in this direction.

Theorem 6.1.

The expected number of operations on an element in row i, col j, $j > i$ is given by the formula

$$\sum_{k=2}^{i} (P_k - P_{k-1})/(1 - P_{k-1})$$

where the $P_k$ are defined in equation (4.1)

Proof.

The probability that an element is non-zero before stage k

= The probability that it is non-zero before stage k-1

+ The probability that it is zero before stage k-1 and is filled in at stage k-1

Hence

$$P_k = P_{k-1} + (1 - P_{k-1})X$$

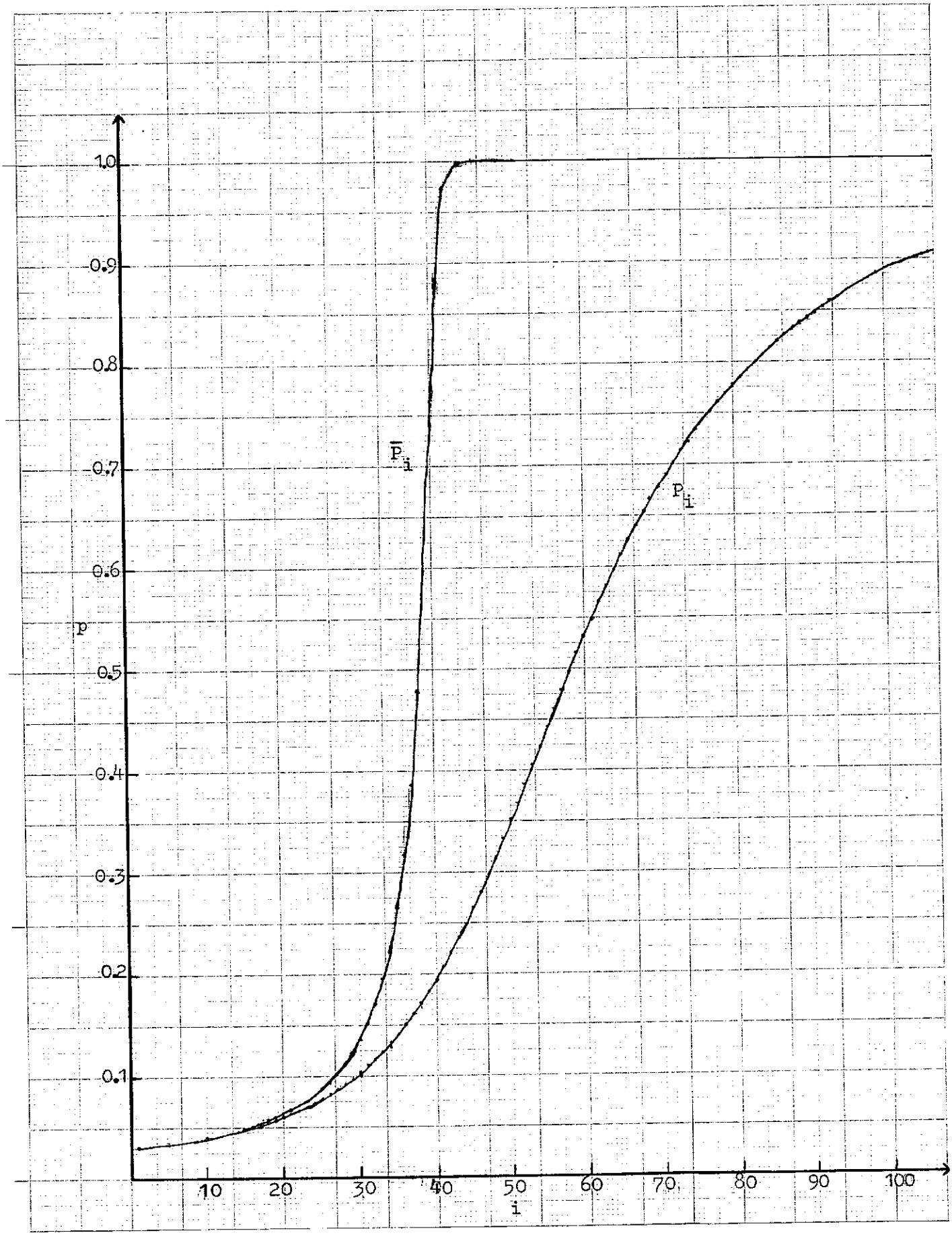
where

X is the probability that the element is filled in (or operated on) at stage k-1.

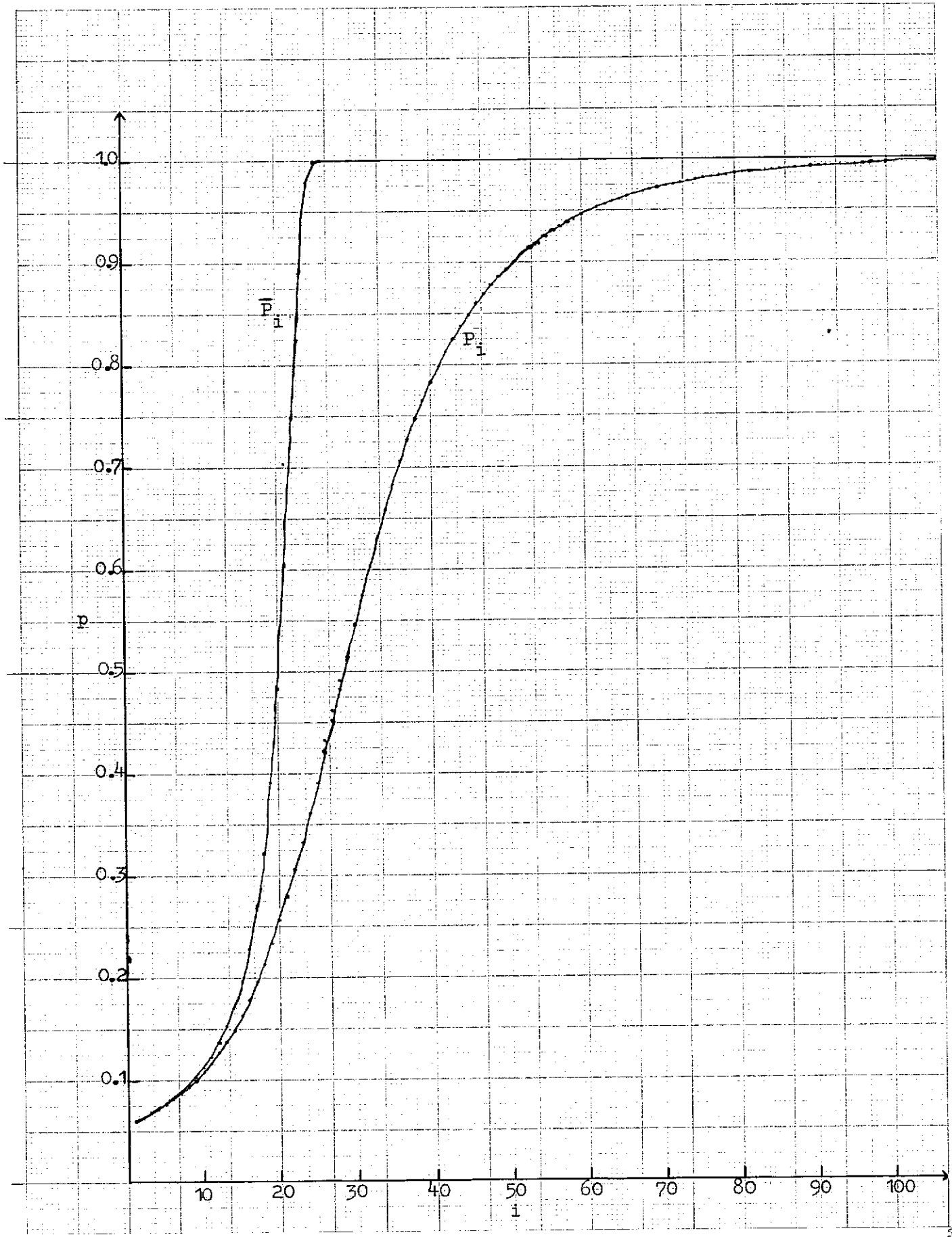
Hence result.

| n | $p_1$ | NUMBER OF OFF-DIAGONAL NON-ZEROS IN FINAL FORM | | | STANDARD DEVIATION OF EXPERIMENTAL RESULTS |
|---|---|---|---|---|---|
| | | $\bar{P}$ | P | EXPERIMENTAL | |
| 100 | 0.01 | 176 | / | 160 | 14.8 |
| 50 | 0.01 | 30 | / | 29 | 6.25 |
| 30 | 0.01 | 10 | 10 | 10 | 2.54 |
| 25 | 0.01 | 6 | 6 | 6 | 0.89 |
| 20 | 0.01 | 4 | 4 | 4 | 1.80 |
| 100 | 0.03 | 4259 | 2253 | 2309 | 239 |
| 50 | 0.03 | 290 | 168 | 162 | 33.4 |
| 30 | 0.03 | 39 | 38 | 37 | 6.18 |
| 25 | 0.03 | 24 | 24 | 22 | 3.45 |
| 20 | 0.03 | 14 | 14 | 13 | 1.06 |
| 100 | 0.06 | 6763 | 5204 | 5211 | 334 |
| 50 | 0.06 | 1062 | 656 | 641 | 71.3 |
| 30 | 0.06 | 182 | 119 | 123 | 33.8 |
| 25 | 0.06 | 86 | 68 | 69 | 17.1 |
| 20 | 0.06 | 39 | 36 | 33 | 5.33 |
| 100 | 0.10 | 7966 | 7022 | 6925 | 246 |
| 50 | 0.10 | 1553 | 1212 | 1193 | 77.5 |
| 30 | 0.10 | 388 | 274 | 264 | 34.9 |
| 25 | 0.10 | 222 | 157 | 161 | 22.4 |
| 20 | 0.10 | 105 | 79 | 82 | 13.0 |
| 100 | 0.50 | 9662 | 9623 | 9637 | 47.5 |
| 50 | 0.50 | 2334 | 2315 | 2317 | 28.3 |
| 30 | 0.50 | 802 | 792 | 788 | 14.6 |
| 25 | 0.50 | 544 | 536 | 537 | 15.5 |
| 20 | 0.50 | 336 | 330 | 333 | 10.9 |

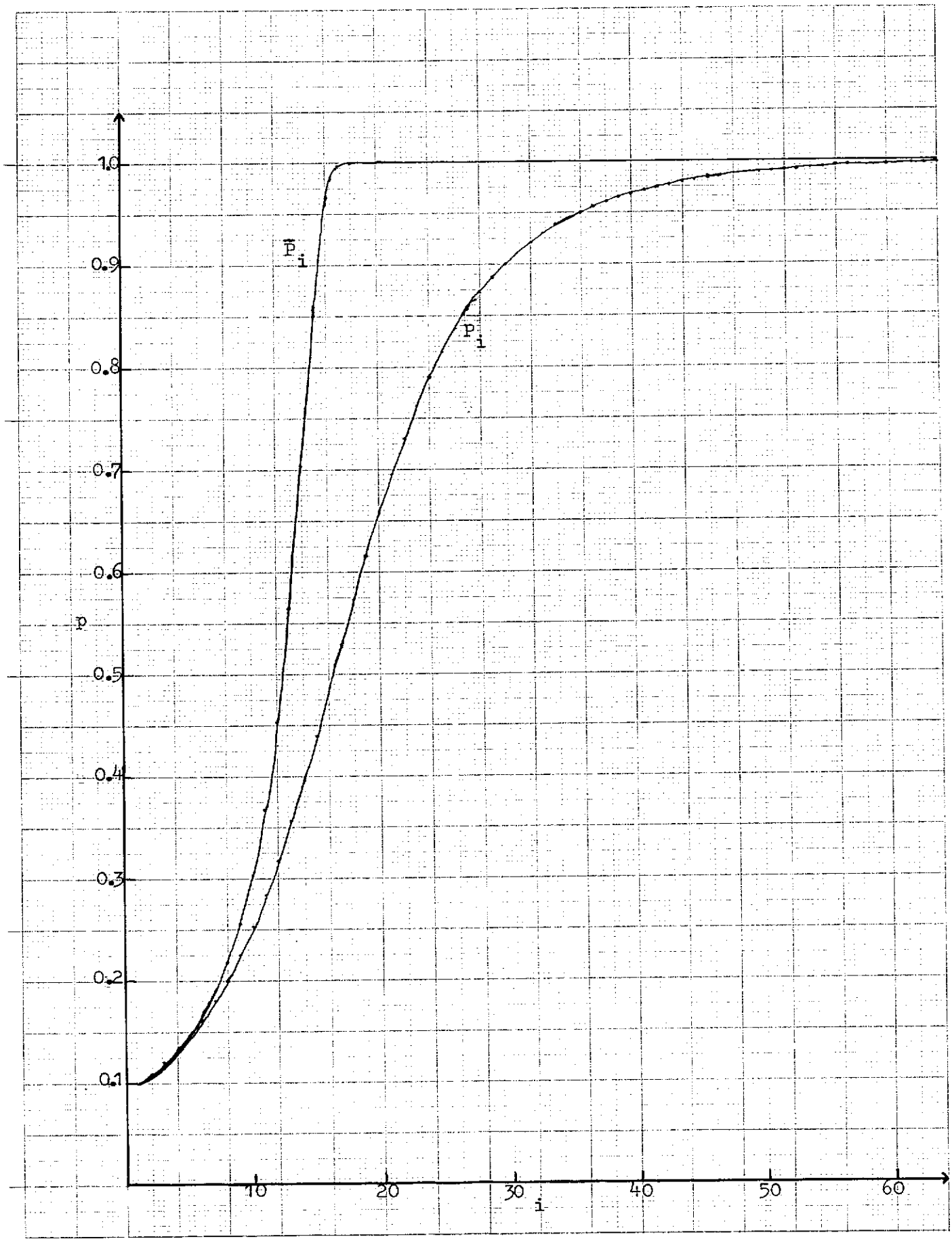
Table 5.1. A comparison of the average fill-in caused by Gaussian elimination on several different random matrices (average of 20 runs at each order and density) with the theoretical prediction given in theorems 4.1 and 4.2 and the upper bound given by equation (2.1).



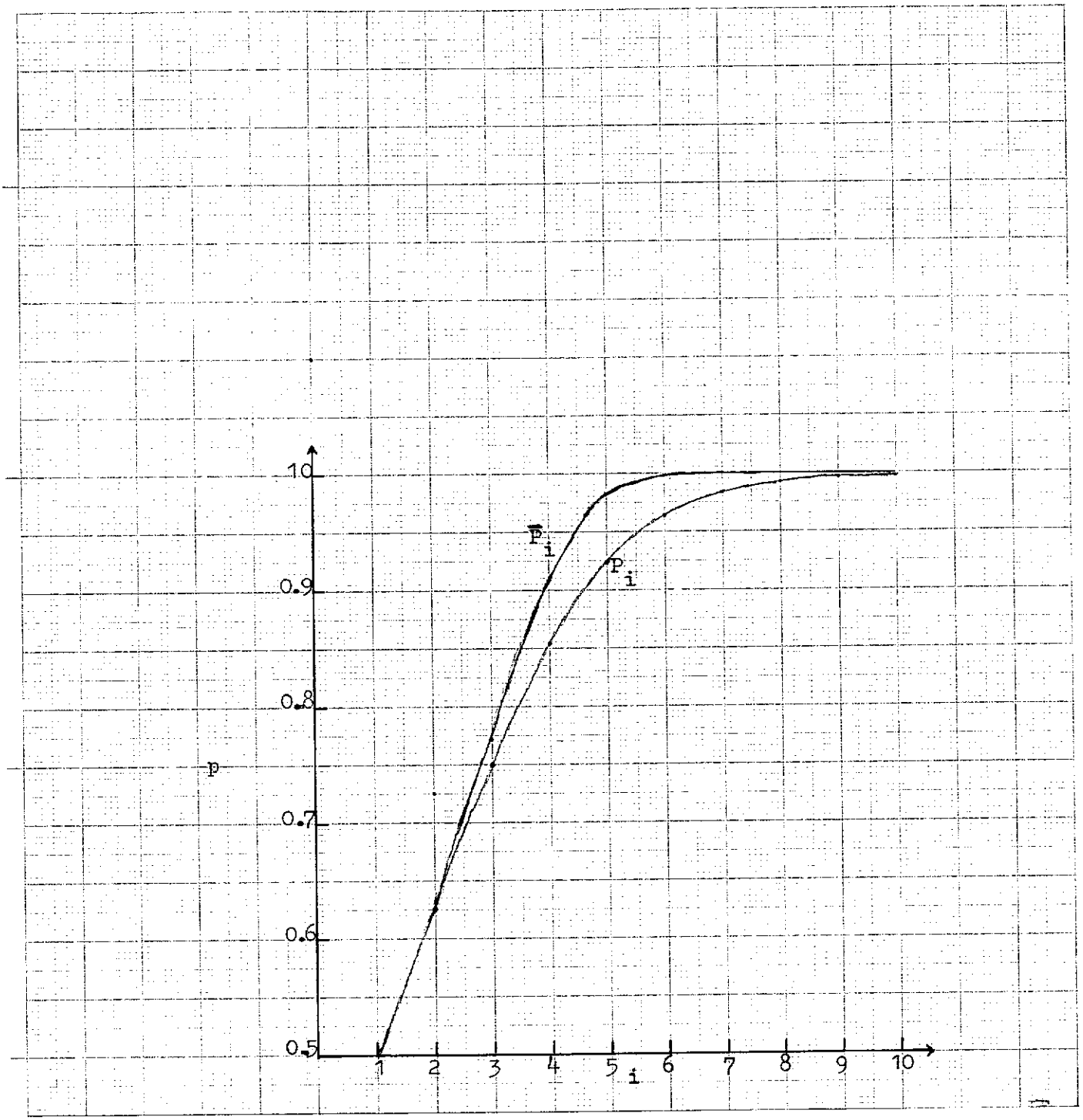

Graph 5.1. A comparison of the true probability density of the reduced form $(P_i)$ with that derived from the recurrence relation (2.1) $(\bar{P}_i)$ with starting value of $p = 0.03$.




Graph 5.2. A comparison of the true probability of the reduced form (equation (4.1)) with that derived from the recurrence relation (2.1) with starting value of $p = 0.06$.




Graph 5.3. A comparison of the true probability density of the reduced from (equation (4.1)) with that derived from the recurrence relation (2.1) with starting value of p = 0.10.




Graph 5.4. A comparison of the true probability density of the reduced form with that derived from the recurrence relation (2.1) with starting value of $p = 0.50$.

# CHAPTER 7.

## OVERDETERMINED SYSTEMS.

Section 1. Introduction.

Section 2. A brief discussion of the methods under consideration.

Section 3. A discussion of row orderings in Givens' method and a comparison of Givens' and Householder's methods.

Section 4. Some theoretical reflections on a comparison of the various methods.

Section 5. Some experimental observations on a comparison of the various methods.

## Section 1. Introduction

This chapter is concerned with the solution of the overdetermined set of equations

$$Ax = b \qquad \ldots\ldots (1.1)$$

where $A$ is an $m \times n$ $(m \geq n)$ sparse matrix and $x$ and $b$ are column vectors of length $n$.

It is assumed throughout this chapter that the rank of $A$ is $n$. Although some of the discussion can be easily extended to cover matrices of defective rank, it is more difficult to extend the experimental results on fill-in since there the linear dependence of the columns would have to be reflected in some numerical cancellations during the reduction process. It is also possible to extend the results to an underdetermined set of equations where $m \leq n$ in equation (1.1) above. In this case, the roles of the matrices which factorise $A$ are generally reversed but most of the analysis still holds.

It is concerned here to consider methods which solve equation (1.1) in the $l_2$ norm. That is to say, it is a least-squares problem which is being considered where it is desired to find $x$ such that

$$\left\{ \sum_{i=1}^{n} r_i^2 \right\}^{\frac{1}{2}}$$

is minimised where $r_i$ is the i th component of the vector $b - Ax$. When $A$ is of full rank this solution will necessarily be unique. The same problem, in the $l_\infty$ norm, is known as the minimax solution of sets of equations as in (1.1) (Cheney (1966)), but has not been considered here.

The methods considered for solving the system in equation (1.1) are that of forming the normal equations, that of performing premultiplications of $A$ by orthogonal matrices to reduce it to upper

triangular form, and a method based on LU factorisation of A due to Wilkinson and Peters (1970). These methods are described in section 2 of this chapter.

In section 3, the orthogonalisation methods utilising the plane rotations of Givens (1958) or the unitary transformations of Householder (1958) are examined in some detail. Different methods of row ordering within a Givens major step are discussed as well as various criteria for selecting the pivots. It is observed that although Givens' method with the best orderings produces less fill-in than Householder's method (thus supporting the theorems in section 2 of chapter eight), the method of Householder is to be preferred in a sparsity context.

Sections 4 and 5 deal with a comparison of Householder's technique with the other methods mentioned in this section. It is seen in section 4 that it is difficult to make any hard and fast rules about the advantage of one method over the others since it is shown that the structure of the matrix can be a vital factor. This conclusion is borne out by the experimental results of section 5 where the three methods under consideration have been tested empirically against variety of structured and random matrices.

Finally, the experimental and theoretical results of this chapter establish that it is, in most cases, advisable to use the method of Wilkinson and Peters as a general routine for solving sparse sets of overdetermined equations. This method combines the sparsity advantages of the method of normal equations with the numerical stability of the orthogonalisation procedures.

## Section 2. A brief discussion of the methods under consideration.

As was mentioned in the introduction, this section discusses some methods for solving the overdetermined system of linear equations (1.1) where the solution is found in the $l_2$ norm. Each method is considered in turn and table 2.1 sets out the features of the methods which are important in a sparsity context and are integral to the remainder of this chapter.

Historically, one of the original ways of solving the least squares problem has been to form the normal equations and to solve the resulting system whose coefficient matrix is square and positive definite. This process is indicated in the equations

$$Ax = b \qquad \ldots\ldots (2.1)$$

$$A^TAx = A^Tb \qquad \ldots\ldots (2.2)$$

where the set of equations in (2.2) are solved by using the $LDL^T$ decomposition of the positive definite $A^TA$, where $L$ is unit lower triangular and $D$ is a diagonal matrix whose entries are positive. Wilkinson and Peters (1970) have a simple proof that this use of the normal equations does indeed give a solution to the least squares problem. This method has tended to be out of favour since the premultiplication of equation (2.1) by the matrix $A^T$ effectively squares the condition number of the coefficient matrix and often leads to the set of equations (2.2) being very ill-conditioned. Another stability problem can arise due to bad scaling in $A$ (unlike the linear case, it is not possible to scale A as one would like). This is caused by the premultiplication of the right-hand side by $A^T$ where the presence of a small number (less than the rank of A) of heavily weighted rows of $A$ can lead to disastrous information loss in the right-hand side. This method is still, however, used in several routines (for example, Reid (1972b)) since the storage is kept low if the matrix $A$ is still held for other reasons, as would

be the case in linear programming problems. Another reason for using the method of normal equations has been that, for full matrices, the number of operations is low (see table 2.1). However, the work of this chapter shows that this is not necessarily true in the sparse case. This method becomes numerically feasible, if it is used together with iterative refinement or if some check is to be made on the accuracy of the solution obtained.

Wilkinson and Peters (1970) suggest that one method of avoiding this ill-conditioning is to utilise the ordinary LU decomposition of A. With A decomposed as

$$A = LU \qquad \ldots\ (2.3)$$

where L is a unit lower trapezoidal $m \times n$ matrix and U is an upper triangular $n \times n$ matrix, the normal equations become

$$U^T L^T L U x = U^T L^T b \qquad \ldots\ (2.4)$$

Since A is of full rank, $U^T$ is non-singular and can be cancelled from equation (2.4) to give

$$(L^T L) U x = L^T b \qquad \ldots\ (2.5)$$

Now the matrix $L^T L$ is square and positive definite of order n and can therefore be written as

$$L^T L = L_1 D L_1^T \qquad \ldots\ (2.6)$$

where $L_1$ is a unit lower triangular $n \times n$ matrix and D a diagonal matrix with positive entries. The solution process then consists in decomposing A as in equation (2.3) and solving equation (2.5) by means of the decomposition in (2.6) and several forwards and backwards substitutions. It is the cancellation of $U^T$ in equation (2.4) which removes the ill-conditioning present in the normal equations method.

Another method of overcoming the ill-conditioning in the normal equations is to utilise the decomposition

$$A = QU \qquad \ldots\ (2.7)$$

where $Q$ is an orthogonal matrix of order $m \times n$ and $U$ is a square upper triangular matrix of order $n$. This orthogonal decomposition is generally performed using the plane rotations of Givens (1958) or the unitary transformations of Householder (1958). A full description and error analysis of these methods in the case of linear equations is to be found in Wilkinson (1965). Golub (1965) and Businger and Golub (1965) describe in some detail how Householder's method can be used in the solution of least squares problems.

With this decomposition, the normal equations become

$$U^T(Q^TQ)U\ x = U^TQ^T\ b \qquad \ldots\ (2.8)$$

Now, if $A$ is of full rank, $Q^TQ$ is the identity matrix of order $n$ and $U^T$ is again non-singular and can be cancelled to give

$$U\ x = Q^T\ b \qquad \ldots\ (2.9)$$

Thus, the solution of the least squares problem has been reduced to performing the $QU$ decomposition of $A$ as in (2.7) and solving the triangular set of equations given in (2.9). As mentioned by Wilkinson and Peters (1970), the extra work done in the decomposition of equation (2.7) over the decomposition (2.3) is to some extent compensated by the simplification of equation (2.9) over (2.5). The extent of this compensation in a sparsity context is seen in section 5. It is possible in this method to take advantage of the retention of $A$, if it is kept for other reasons, by solving, instead of (2.9), the equation

$$U^TU\ x = A^T\ b \qquad \ldots\ (2.10)$$

This method, suggested for use in linear programming by Gill and Murray (1970) and viewed in a sparsity context by Saunders (1972), has the advantage that there is no need to store the $Q$ of equation (2.7) for the solution of subsequent right-hand sides, but has the disadvantage that the premultiplication of $b$ by $A^T$ can lead to the same trouble as mentioned in the normal equations method if the matrix $A$ is badly scaled. It should be noted that this source of

instability is removed if methods involving the solution of equations (2.5) or (2.9) are used provided that pivoting is performed during the initial decomposition phase of equations (2.3) and (2.7) respectively. For LU decomposition suitable pivoting strategies are discussed by Wilkinson (1965) while Powell and Reid (1968) discuss strategies for use in the QU decomposition by Householder transformations.

Table 2.1 gives values for some fairly important counts for the four methods described in this section. A few notes are now made concerning this table. The first two columns are particularly relevant for sparse matrices, the last three for full ones. The values given in columns 3 and 4 can be considered upper bounds for the quantities in columns 1 and 2 respectively. It should be noted that the $Q$ and $U_1$ for the two orthogonal reduction methods will not be the same and will, in general, not even have the same densities. A theorem in section 2 of the next chapter conducts a comparison between these different $Q$s and $U_1$s. Although it is impossible, without reference to the structure of the initial system, to give an a priori estimate for the number of operations to decompose a particular sparse system, this operation count for full systems can be evaluated and is given in the last column of the table. A description of how this count is obtained, in each case, serves to illustrate the manner in which such counts have been calculated by the programs producing the results given in the tables of section 3 and section 5.

Normal equations : There are $\frac{1}{2}mn^2$ operations for the premultiplication of $A$ by $A^T$ followed by $\frac{1}{6}n^3 + \frac{1}{2}n^2 + O(n)$ operations for the symmetric decomposition of $A^TA$, the number of operations for the first step in the reduction being $\frac{1}{2}n(n + 1) - 1$.

Givens' method : The total given in table 2.1 is the number of operations necessary for the QU decomposition of $A$. The number of operations for the first major step is $4n(m-1)$. There are, in addition, $mn - \frac{1}{2}n^2 + O(n)$ sine-cosine pairs to be evaluated if this

method is used.

Householder's method : The count given in table 2.1 is the number of operations necessary for the QU decomposition of A by this method. Each major step corresponds to premultiplying the matrix by a unitary transformation of the type $I - ww^T/2K^2$ . To multiply a vector by such a matrix, one first takes the vector and multiplies by the row vector $w^T$ to obtain a scalar which is divided by $2K^2$ and used to multiply the components of the vector w. The resulting vector is then subtracted from the original one. This process will take $2m+1$ multiplications if the vectors concerned are of length m, thus giving, for the first major step of Householder reduction, a total of $(2m+1)(n-1)$ operations. In addition, the computation of $2K^2$ and w requires a further multiplication, m squares, and one square root. Hence, in addition to the operation count given for Householder's method, one has also to evaluate $mn - \frac{1}{2}n^2 + O(n)$ squares and n square roots.

Wilkinson and Peters method : The value given for this operation count is the sum of several counts from the different steps of this procedure. The initial LU decomposition requires $\frac{1}{2}mn^2 - \frac{1}{6}n^3 + \frac{1}{2}mn - \frac{1}{2}n^2 + O(m) + O(n)$ operations, the number of operations at the first stage of the reduction being, since L is unit trapezoidal, $n(m-1) + 1$. The multiplication of L by $L^T$ requires $\frac{1}{2}mn^2 - \frac{1}{3}n^3 + \frac{1}{4}n^2 + O(n)$ operations while the final symmetric decomposition of $LL^T$ takes $\frac{1}{6}n^3 + \frac{1}{2}n^2 + O(n)$ operations similar to the decomposition of $A^TA$ in the normal equations case.

There now follows some comments on the operation counts for full matrices. These comments should be contrasted with the remarks of section 4 and the notes on table 5.4. In the discussion which follows it is only the high order terms in the counts which are being considered.

It can first be observed what happens in the limiting case when

$m = n$ and the system under consideration is square. In this case, the method of Householder appears to have the edge while the normal equations and Wilkinson and Peters methods are about equal with Givens doing very badly on operation counts and not compensated by gains in storage.

As the ratio $m/n$ increases (the system becomes more rectangular) the method of Givens will become correspondingly worse and eventually the increase of this ratio causes the counts for Householder's method to become unacceptably high in comparison with the methods of normal equations and Wilkinson and Peters. These two methods are themselves comparable on all counts except that for the initial decomposition where an increase in the ratio $m/n$ eventually causes Wilkinson and Peters method to take about double the number of operations of the normal equations method.

The above arguments on operations counts give part of the reason for the historical popularity of the method of normal equations. As mentioned earlier, the operation count, even for full systems, only paints part of the picture and it is stability difficulties which have prevented this method being totally dominant. It is part of the raison d'être of this chapter to indicate that, for sparse systems, not even does this operation count argument so dramatically favour the method of normal equations.

| METHOD \ COUNT | TOTAL STORAGE REQUIRED | NUMBER OF MULTIPLICATIONS FOR SOLN. OF SUBSEQUENT RIGHT-HAND SIDES | VALUES IF MATRIX IS FULL (and m x n) | | |
|---|---|---|---|---|---|
| | | | TOTAL STORAGE | MULTIPLICATIONS FOR RHS. | MULTIPLICATIONS IN DECOMPOSITION PHASE. |
| Normal Equations $A^TA=L_1DL_1^T$ | $A + L_1$ | $A + 2L_1 - n$ | $mn + \frac{1}{2}n^2$ | $mn + n^2$ | $\frac{1}{2}mn^2 + \frac{1}{6}n^3$ |
| GIVENS $A = QU_1$ | $2Q + U_1 - 2n$ | $4Q + U_1 - 4n$ | $2mn - \frac{1}{2}n^2$ | $4mn - \frac{3}{2}n^2$ | $2mn^2 - \frac{2}{3}n^3$ |
| HOUSEHOLDER $A = QU_1$ | $Q + U_1 + n$ | $2Q + U_1 + n$ | $mn$ | $2mn - \frac{1}{2}n^2$ | $mn^2 - \frac{1}{3}n^3$ |
| WILKINSON AND PETERS $A = LU$ $L^TL = L_2DL_2^T$ | $L_2 + L + U - n$ | $2L_2 + L + U - 2n$ | $mn + \frac{1}{2}n^2$ | $mn + n^2$ | $mn^2 - \frac{1}{3}n^3$ |

Table 2.1. A table of operation counts for the methods described in section 2. In the first two columns of the table each capital letter stands for the number of non-zeros in the corresponding matrix and, in the case of Q and L, it is the number of non-zeros in their factored from which is being referred to. In every case, the elements (i,i) ($1 \leq i \leq n$) of the matrix under consideration are included in this number. In the last three columns, giving counts for full matrices, only the high order terms have been included.

## Section 3. A discussion of row orderings in Givens' method and a comparison of Givens' and Householder's methods.

The arguments of the next chapter indicate that the use of Householder's procedure can never give less fill-in than using the procedure due to Givens' when it is desired to use orthogonal transformations to reduce a matrix to Hessenberg form, if the same pivots are used in each case. This is because the fill-in in Householder's method is related to the total Boolean sum of all the rows under consideration at a particular major stage of the reduction while the Givens fill-in is only related to the Boolean sum of the rows already processed in this major stage. These arguments carry over in a similar fashion to this chapter. However, the gains of Givens over Householder have to be quite substantial before it can be considered the better of the two methods since the actual number of operations performed in multiplying the right-hand side of the equations by the orthogonal matrix $Q$ is roughly twice that for Givens' method if the fill-ins are the same. A similar comment also applies to the storage. Ideally, it would be desirable if the fill-in due to Givens were half that of Householder and this section is an attempt to reduce this fill-in of the Givens' method.

Clearly, the degree of freedom which is available to the Givens' method but not to the Householder one is that of ordering the rows within a major step of the forward elimination phase. The order of rows within Householder's reduction does not affect the number of non-zeros created but such an order can influence the fill-in due to Givens as is seen clearly in figures 3.1(a) and 3.1(b).

| | |
|---|---|
| x o o ... | x o o ... |
| x x x ... | x o o ... |
| x o o ... | x x x ... |
| figure 3.1(a) | figure 3.1(b) |

Elimination, by Givens' method, using the pivot x with the order of

the rows indicated in figure 3.1(a) gives a fill-in of 4 while the ordering of figure 3.1(b) gives a fill-in of 2. For both orderings, Householder's method gives a fill-in of 4. The effect of this on a complete reduction is shown in figures 2.11(a) and 2.11(b) of the next chapter.

The problem to which it is desired to find a solution is one of finding that row ordering which would minimise the total fill-in in one stage of the Givens reduction after a pivot has been selected. The problem of selecting this pivot is considered later in this section but, for the meantime, it is assumed that some criterion exists for selecting it and that this criterion is the same irrespective of the row ordering strategy used. The discussion and examples of the next few paragraphs illustrate the difficulty of obtaining such an optimum order.

Theorem 3.1.

If the rows are ordered in increasing row count, then it is untrue that the fill-in for the whole major stage of the Givens reduction is necessarily minimised.

Proof.

This theorem is proved by means of a counter-example. Consider the submatrix of figure 3.2 where x represents the pivot element.

| | |
|---|---|
| x x x x o o | |
| x o o o x x | 1 |
| x x x x o o | 2 |

figure 3.2

Ordering on minimum row count gives the ordering shown in the figure with a total fill-in of 7. If, however, the rows (1 and 2) are taken in the reverse order the total fill-in is 5, thus yielding the counter-example and proof of the theorem.

At each minor stage of the reduction, the pivotal row and the row being operated upon each take the sparsity pattern of the union of the

two. It therefore seems sensible to keep the number of non-zeros in the pivotal row small as long as possible. The strategy of minimising the fill-in in the pivotal row at each stage will be called $\mathfrak{R}$ . It is seen that $\mathfrak{R}$ correctly orders the rows of the submatrix in figure 3.2 to give minimum total fill-in but the following theorem indicates that this is not necessarily the case.

Theorem 3.2.

If the rows are ordered by the criterion $\mathfrak{R}$ , then it is untrue that the fill-in for the whole major stage of the Givens reduction is necessarily minimised.

Proof.

The proof is again by means of a counter-example. In figure 3.3, $\underline{x}$ again represents the pivot element.

| | | | | | | |
|---|---|---|---|---|---|---|
| $\underline{x}$ | o | o | o | o | o | |
| x | x | x | x | o | o | 1 |
| x | x | x | x | o | o | 2 |
| x | o | o | o | x | x | 3 |

figure 3.3

$\mathfrak{R}$ would order the rows in the order 3, 2, 1 giving a total fill-in of 9 while the row ordering 1, 2, 3 gives a fill-in of only 8. Hence the theorem has been proved.

Although theorem 3.2 indicates that this criterion is not optimal, it is seen, from the results of table 3.1, to produce, in general, a lower total fill-in than using a straight unweighted row count.

Let an active row at any stage of the Givens reduction be one which has a non-zero in the pivotal column and will therefore be operated on during that stage. Then, another method of row ordering might be to select that active row which would cause the least local fill-in. Since non-zeros in the pivotal row inevitably cause fill-in in the corresponding positions of any row being operated on, these positions are considered to be filled-in initially and, furthermore,

the additional fill-ins of this kind which are an inevitable consequence of a fill-in in the pivotal row are assumed to take place at the same time as this fill-in. This strategy works on the example of figure 3.2 and produces a tie on the example of figure 3.3. The following theorem indicates that this ordering again fails to achieve the desired result.

Theorem 3.3.

If the rows are ordered by the local minimum fill-in criterion, then it is untrue that the fill-in for the whole major stage of the Givens reduction is necessarily minimised.

Proof.

Once more a counter-example is used to prove this theorem. Consider the submatrix of figure 3.4 where the (1,1) element is chosen as pivot.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| x | o | o | o | o | o | o | |
| x | x | o | o | o | o | o | 1 |
| x | o | x | x | o | o | o | 2 |
| x | o | x | x | x | o | o | 3 |
| x | o | x | x | o | x | o | 4 |
| x | o | x | x | o | o | x | 5 |

figure 3.4

Then, if the rows are ordered as shown in the diagram the total fill-in to result is 13. However, if they are ordered so that the row giving local minimum fill-in is chosen at each stage, then the fill-in is 14. A possible such ordering is to select the rows in the order 2, 1, 3, 4, 5. This completes the proof of the theorem.

The author has tried experimenting with even more sophisticated strategies involving the notion of nested blocks (for example, rows 2 and 3 of figure 3.4 above) but, in each case, counter-examples exist and the overcomplication of the selection method does not necessarily result in any saving of overall growth in the number of non-zeros. It might, of course, be possible, especially if the original matrix

is very sparse, to consider all permutations of the rows and order them to obtain the minimum fill-in for the complete major stage. This method becomes very time consuming if the matrix is not very sparse (less than, say, 3 or 4 elements per row) or if substantial fill-in occurs to the lower trapezoidal part of the matrix during the orthogonal reduction. In addition, there is no guarantee that local minimisation of fill-in at the major reduction step level will result in a global minimisation for the whole reduction process and obviously the total fill-in will depend on the criterion for pivot selection. This additional factor would indicate that there is no great value in expending too much effort on the row ordering when a sub-optimal ordering is all that can be achieved. Therefore, the remainder of this discussion concentrates on the three ordering methods mentioned previously which are all computationally feasible and which give a good reduction in fill-in over taking the rows in their natural order.

In each case the pivot was selected from the remaining submatrix by choosing the column with minimum column count and selecting the non-zero from that column with minimum row count (this method is henceforth called '$r_i$ within $c_j$'). Justification for using this criterion will be found later in this section.

The results giving the final number of non-zeros in the upper triangular and lower trapezoidal parts of the matrix for these three orderings are given in table 3.1. In addition, runs were performed using a row ordering equivalent to the natural ordering so that some idea of the possible improvement by these orderings can be obtained. Runs were made with the three structured matrices used in previous chapters of this thesis where an additional row has been added with a non-zero in its first column and zeros elsewhere. Three different orders of random matrix were used to see the effect of making the system more rectangular and runs were made at each order with matrices of three different densities, each value in the table being the average obtained over five separate runs. It is seen that although, in every

case, there is little difference in the final number of non-zeros in the upper triangle (of the matrix permuted so that the pivots are on the diagonal in the natural order) the row orderings do, in general, produce a fairly substantial saving in the final number of non-zeros in the lower trapezoidal part of the system sometimes producing reductions of over 25%. That this reduction occurs in the lower trapezium is particularly gratifying because of the weight given to it in table 2.1 as regards storage and the number of operations for the solution of future right-hand sides. The reason why the row orderings produce little effect on the upper triangular part of the system is that, irrespective of the row ordering, the final structure of the pivot row (which is necessarily in the upper triangle) will be the logical sum of its original structure and all the active rows at this stage. Hence, the fill-in in this part of the matrix should not be so dependent on the local row orderings, particularly if some suitable pivot criterion is also being used. This phenomenon is also responsible for the much greater number of non-zeros in the upper triangle than in the lower trapezium in all of the square examples of table 3.1. It is seen that the number of final non-zeros in the upper triangle is sometimes over twice that in the lower trapezium and that it is only when the ratio $m/n$ reaches 2 that the greater size of the lower triangle counterbalances this effect. This phenomenon is seen to occur again later in this section and in section 4 of this chapter. It is also seen, from the results of table 3.1, that $\mathfrak{R}$ does, in general, produce a saving, over the minimum unweighted row ordering, in the number of non-zeros in the lower trapezium of the decomposed form. This should not be too surprising since $\mathfrak{R}$ is essentially a local row ordering criterion while ordering the rows on minimum row count is an a priori method. The other local method, that of local minimum fill-in produces a fill-in comparable to that of $\mathfrak{R}$ and often slightly worse than it. Because the $\mathfrak{R}$ ordering is much easier to implement, this ordering is used for the experiments on

pivot selection and for the comparisons with Householder's method later in this section. The relative merits of the three orderings and that of keeping the natural row ordering is seen to remain the same under change of order and density.

An examination was then conducted to investigate the effect of altering the basis on which the pivot element is selected at each stage of the reduction. The row ordering technique was kept constant during this investigation and was chosen to be the $\mathfrak{R}$ of the preceeding paragraphs because of its relatively successful results and its ease of implementation. Five different strategies were tested and compared against a test case of the natural column ordering. The methods which were compared were as follows.

The method of the first part of this section, namely $r_i$ within $c_j$, was used. This criterion is chosen since it brings in the least number of rows at each stage and should result in a minimisation of fill-in in the pivot row due to a reduction in the number of rows involved in the logical summation mentioned earlier. In addition, the growth in the lower trapezium should be reduced by such a scheme since effectively the pivot chosen at each stage introduces the maximum number of zero elements to the lower trapezium. The reverse of this method, called $c_j$ within $r_i$, is also tried in an attempt to keep the growth of non-zeros low by choosing a very sparse row as the pivot row. Another selection criterion tried was to choose the non-zero element $(i,j)$ such that $r_i \times c_j$ was a minimum over all the non-zeros in the unreduced part of the matrix. Such a strategy is called the Markowitz criterion after the similar scheme for pivot selection in linear equations (chapter 5). Although this does not have the same direct relationship as in the linear equation case since it does not minimise the maximum fill-in, it should produce a beneficial effect by ensuring that neither $r_i$ nor $c_j$ become very large. An adaptation of this method lays somewhat more stress on

reducing the number of non-zeros in the pivotal column (for the same reasons as was given for the method $r_i$ within $c_j$ ) and is implemented by selecting as pivot that non-zero for which the count $r_i \times c_j^2$ is minimised. Finally, the a priori method of ordering the columns in ascending order of column count prior to the elimination is also investigated. At each stage, the non-zero in the column with minimum row count is chosen as pivot.

Results are given in table 3.2 for these five orderings and that of taking the columns in their natural order. The matrices were identical to those used in compiling table 3.1 and again the figures in the table indicate the number of non-zeros in the upper and lower triangular parts of the final matrix. The following notes can now be made.

(1) As was seen in the results of table 3.1 it is also observed here that the final density of the lower trapezium is always less than that of the upper triangle. As mentioned earlier this is very pleasing because of the weighting given to the lower trapezium in the counts of table 2.1.

(2) All the orderings show a marked improvement over using the natural column ordering often reducing the density of the final form by more than 50%.

(3) The a priori ordering does significantly worse than the best local orderings thus discounting it as a viable method. It is particularly bad on square systems but this comparison holds through all the examples tried.

(4) The method $c_j$ within $r_i$ is substantially worse than the other local methods as regards fill-in to the lower trapezium and in many cases shows no improvement over natural column ordering as regards the fill-in to the upper triangle. It is thus felt that this method of ordering is not desirable.

(5) The Markowitz criterion has similar deficiences often being only slightly better than method $c_j$ within $r_i$. The improvement

to this result when the criterion $r_i \times c_j^2$ is used is an indication of the greater importance of reducing $c_j$ rather than keeping $r_i$ small.

(6) When this is taken to its logical extreme in the method $r_i$ within $c_j$, the effect of placing the emphasis on the $c_j$ is clearly seen. Not only does this method yield the lowest fill-in in the lower trapezium as might be expected from the previous discussion, but it often gives the best results in terms of fill-in to the upper triangle. Thus, this method not only achieves a much lower operation count for the solution of successive right-hand sides but it also keeps the total fill-in produced in the reduction process to a minimum.

(7) The reflections of notes (1) - (6) are true for the structured and random matrices tested and hold over a range of orders and densities.

(8) Notes (2) to (7) indicate that both for reduction in operation count for subsequent solution of right-hand sides and for reduction in fill-in, the use of $r_i$ within $c_j$ for pivot selection is clearly the best criterion of all those tested.

It would appear that a very good method of implementing Givens reduction to upper triangular form is to select the pivot at each stage on the basis of $r_i$ within $c_j$ and then to order the rows using $\mathcal{R}$. It is this ordering which is now used for Givens when comparing his procedure with that of Householder for which the same pivot selection criterion is used.

It is generally assumed to be the case that a reduction using Givens will give decomposed forms requiring twice the amount of storage and twice the amount of operations for subsequent solution of right-hand sides than the decomposed forms of Householder. This is based on the factors multiplying $Q$ in the first two columns of table 2.1 and the factors multiplying $mn$ and $mn^2$ in the other columns. If it were true, then, in order that Givens' method was

competitive with Householder's, it would be necessary that the lower trapezium be half as dense in the Givens decomposition as in the Householder one. However, a closer examination of columns 1 and 2 of table 2.1 indicate that factors of $n$ appear to counterbalance the above argument if the number of non-zeros in $Q$ is of order $n$. The results of table 3.3 indicate that this is indeed the case with the number of non-zeros in $Q$ being generally less than $10n$ so, for example, the $5n$ difference in the operation counts in column 2 of table 2.1 becomes significant. In order to see whether this effect coupled with the proven lower fill-in for Givens' method could outweigh the factors multiplying the $Q$s in table 2.1, the two methods of Householder and Givens were tested with the usual selection of random matrices and the results are given in table 3.3. In this table values for the number of operations required for the solution of further right-hand sides and for the storage of the decomposed forms is also given. The calculation for these is done according to table 2.1 where, for example, $Q$ is the final number of non-zeros in the lower trapezium plus $n$ for the elements $(i,i)$. The results indicate that, even although Householder's method invariably gives a larger fill-in than Givens' one (thus supporting previous arguments in this section and theorem 2.1 of the next chapter) and, in particular, gives a significantly denser lower trapezium, it generally gives a lower operation count for the subsequent solution of right-hand sides and requires less storage for its decomposed form. The only cases where this is not true is for the very sparse matrices (less than 3 elements per row) where the factor of order $n$ in table 2.1 plays a significant role. Thus, although this case should be borne in mind, the findings of this chapter indicate that, from sparsity considerations alone, the method of Householder should be preferred to that of Givens as a general purpose method for the orthogonal reduction of a matrix to upper triangular form. The extent to which it provides a viable means of solving the least squares

problem in comparison with other techniques is the subject of the following two sections.

It should be noted that all these calculations have been made on random matrices and a few structured ones. For matrices with a particular structure, there may be other orthogonal reduction procedures which yield better results than the implementation of Householder with the aforementioned pivot selection strategy. An example of this kind for band matrices is given in Reid (1967).

As a final rider to this section, mention can be made of another possible ordering strategy which could be used when implementing the procedures of this section. Since it is desirable to keep the number of elements in the final decomposition and particularly in the lower trapezium low, then an ordering which presorted the matrix to nearly upper triangular form might be useful. An algorithm for such an ordering for square systems is given in Tewarson (1967c) and, might be extended to the rectangular case to provide the basis for such an ordering method. Initial experiments on such orderings have proved unfruitful because it is difficult to obtain a very good a priori sort of the kind mentioned. However, the author believes some algorithm using this sort might be developed at least for systems which are nearly square.

| MATRIX / PIVOT CHOICE | 55 x 54 CURTIS 1. | 58 x 57 WILLOUGHBY 1 | 200 x 199 WILLOUGHBY 2 | RANDOM 50 x 50 DENSITY 0.03 | 0.05 | 0.10 | RANDOM 75 x 50 DENSITY 0.03 | 0.05 | 0.10 | RANDOM 100 x 50 DENSITY 0.03 | 0.05 | 0.10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Final No. of non-zeros in upper triangle NAT. COL. | 866 | 639 | 18335 | 425 | 832 | 1174 | 601 | 970 | 1190 | 701 | 1018 | 1200 |
| A priori column - in order of ascending column count. | 700 | 489 | 8866 | 183 | 542 | 970 | 312 | 692 | 1059 | 447 | 788 | 1118 |
| $\min_{i,j} (r_i.c_j)$ | 600 | 439 | 8128 | 242 | 545 | 957 | 334 | 672 | 1069 | 428 | 782 | 1132 |
| $\min_{i,j} (r_i c_j^2)$ | 557 | 360 | 7855 | 140 | 470 | 946 | 281 | 641 | 1044 | 365 | 729 | 1115 |
| $r_i$ within $c_j$ | 575 | 416 | 7895 | 136 | 500 | 960 | 296 | 629 | 1037 | 398 | 737 | 1103 |
| $c_j$ within $r_i$ | 651 | 454 | 8985 | 528 | 736 | 1043 | 675 | 1006 | 1162 | 804 | 1028 | 1180 |
| Final No. of non-zeros in lower triangle NAT. COL. | 439 | 309 | 9062 | 124 | 333 | 725 | 367 | 931 | 1621 | 572 | 1402 | 2535 |
| A priori - column in order of ascending column count. | 277 | 222 | 4339 | 37 | 147 | 464 | 206 | 570 | 1287 | 365 | 968 | 2161 |
| $\min_{i,j} (r_i.c_j)$ | 210 | 166 | 4298 | 80 | 219 | 505 | 268 | 736 | 1332 | 448 | 1266 | 2215 |
| $\min_{i,j} (r_i c_j^2)$ | 211 | 140 | 4075 | 34 | 154 | 449 | 198 | 609 | 1274 | 315 | 1034 | 2109 |
| $r_i$ within $c_j$ | 173 | 131 | 3287 | 17 | 107 | 360 | 178 | 481 | 1107 | 305 | 839 | 1883 |
| $c_j$ within $r_i$ | 244 | 194 | 4960 | 233 | 344 | 615 | 565 | 1214 | 1578 | 916 | 1779 | 2468 |

Table 3.2. Final numbers of non-zeros in matrix after decomposition using various criteria for selecting the pivots at each stage. In each case the remaining rows at each major step are ordered by $\mathcal{R}$. In the two a priori column arrangements, the pivot is, at each stage, the non-zero in the column with current minimum row count. The $\min_{i,j}$ are taken over these (i,j) for which $a_{ij} \neq 0$. The values for random matrices are each the average of five runs at the same size and density.

| ROW ORDERING \ MATRIX | | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 | | | RANDOM 75 x 50 | | | RANDOM 100 x 50 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | DENSITY | | | DENSITY | | | DENSITY | | |
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| Final No. of non-zeros in Upper Triangle | Natural Row | 581 | 416 | 7740 | 136 | 500 | 951 | 295 | 606 | 1027 | 398 | 741 | 1107 |
| | Ascending Row count 'A Priori' | 582 | 419 | 7667 | 136 | 497 | 961 | 297 | 638 | 1029 | 398 | 722 | 1101 |
| | $\mathcal{R}$ | 575 | 416 | 7895 | 136 | 500 | 960 | 296 | 629 | 1037 | 398 | 737 | 1103 |
| | Local Minimum Fill | 575 | 416 | 7912 | 136 | 500 | 966 | 296 | 627 | 1036 | 399 | 734 | 1105 |
| Final No. of non-zeros in Lower Triangle | Natural Row | 206 | 134 | 3999 | 17 | 129 | 425 | 256 | 662 | 1288 | 558 | 1209 | 2212 |
| | Ascending Row count 'A Priori' | 185 | 158 | 3638 | 17 | 109 | 380 | 182 | 490 | 1116 | 312 | 854 | 1957 |
| | $\mathcal{R}$ | 173 | 131 | 3287 | 17 | 107 | 360 | 178 | 481 | 1107 | 305 | 839 | 1883 |
| | Local Minimum Fill | 174 | 131 | 3463 | 17 | 106 | 366 | 178 | 486 | 1120 | 303 | 839 | 1890 |

Table 3.1. Final number of non-zeros in matrix after decomposition using various orderings of the rows within the Givens' major step. At each stage, the pivot selected is the non-zero with minimum row count among those non-zeros in the column of minimum column count. The values for random matrices are each the average of five runs at the same size and density.

| METHOD \ MATRIX | | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 DENSITY | | | RANDOM 75 x 50 DENSITY | | | RANDOM 100 x 50 DENSITY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| TOTAL FILL-IN | GIVENS | 510 | 322 | 10502 | 80 | 485 | 1076 | 363 | 925 | 1774 | 555 | 1329 | 2491 |
| | HOUSEHOLDER | 517 | 325 | 11301 | 80 | 528 | 1174 | 486 | 1179 | 2053 | 887 | 1850 | 2975 |
| NNZIFD* IN LOWER TRIANGLE | GIVENS | 173 | 131 | 3287 | 17 | 107 | 360 | 178 | 481 | 1107 | 305 | 839 | 1883 |
| | HOUSEHOLDER | 250 | 174 | 4774 | 17 | 174 | 483 | 313 | 766 | 1408 | 679 | 1408 | 2374 |
| STORAGE | GIVENS | 975 | 735 | 14668 | 220 | 764 | 1730 | 602 | 1641 | 3301 | 1058 | 2465 | 4919 |
| | HOUSEHOLDER | 917 | 721 | 12578 | 303 | 800 | 1568 | 747 | 1514 | 2573 | 1185 | 2247 | 3619 |
| NUMBER OF OPERATIONS TO SOLVE SUBSEQUENT RHS. | GIVENS | 1321 | 994 | 21242 | 254 | 978 | 2450 | 1058 | 2603 | 5515 | 1668 | 4143 | 8685 |
| | HOUSEHOLDER | 1221 | 882 | 17551 | 370 | 1024 | 2101 | 1310 | 2520 | 4231 | 2114 | 3905 | 6243 |

Table 3.3. A comparison of Householder's method with that of Givens using row ordering on $\mathcal{R}$. In both cases, the pivot choice is that non-zero with minimum row count within the column of minimum column count. The values for the random matrices are the average of five runs at the same size and density.

* NNZIFD = Number non-zeros in final decomposition.

## Section 4. Some theoretical reflections on a comparison of the various methods.

This section makes some attempt at looking at the fill-in properties of the methods outlined in section 2. It serves mainly to illustrate some of the difficulties in making such theoretical comparisons. The theorems and discussion tend to show that no hard and fast results are available. As is very common in the field of sparse matrix technology, it appears that the best method is dependent on the structure of the problem.

The following two conjectures are stated and justified and there then follows some arguments which refute these conjectures and illustrate one of the difficulties arising in making comparisons involving a transition from the numeric to the Boolean.

Conjecture 4.1.

If $L$ is the lower trapezoidal part of the $LU$ decomposition of a $m \times n$ matrix $A$ $(m > n)$, then the number of non-zeros in the matrix $A^TA$ is never less than the number of non-zeros in the matrix $L^TL$ irrespective of the pivot choices in the decomposition.

Justification of conjecture.

Let $P$ and $Q$ be permutation matrices such that

$$PAQ = LU$$

then

$$Q^TA^TAQ = U^T(L^TL)U \qquad \ldots\ (4.1)$$

Now, clearly the left hand side has the same number of non-zeros as $A^TA$. In the right hand side the matrix $L^TL$ is premultiplied by $U^T$ and postmultiplied by $U$ and so, if these are considered in a Booleanised form, the matrix $U^T(L^TL)U$ does not have less non-zeros than $L^TL$. That is to say $A^TA$ does not have less non-zeros than $L^TL$.

Conjecture 4.2.

The upper triangular matrix left after premultiplication of $A$ by an orthogonal transformation of a Givens or Householder type has the

same structure as the upper triangular part in the $LL^T$ decomposition of the normal matrix $A^TA$.

Justification for conjecture.

Let Q be an orthogonal matrix such that

$$QA = U$$

where U is upper triangular.

Then,

$$A^TQ^TQA = A^TA = U^TU \qquad \ldots\ldots (4.2)$$

but the $LL^T$ decomposition of $A^TA$ gives

$$A^TA = LL^T \qquad \ldots\ldots (4.3)$$

and from equations (4.2) and (4.3) and the unicity of decomposition the result of the conjecture has been established.

The results of section 5 show empirically that the two conjectures are wrong, counter-results to conjecture 4.1 being observed in table 5.3 and to conjecture 4.2 in table 5.1. In fact, theorem 4.3 illustrates just how wrong conjecture 4.1 can be. The fault in the two justifications is the same in either case and can be observed by examining the following two examples. The first illustrates the deficiency in conjecture 4.1, the second the deficiency in conjecture 4.2.

Example 1.

Consider the matrix of figure 4.1

$$A = \begin{pmatrix} a & b & & \\ & c & & \\ & & d & \\ f & & & e \end{pmatrix}$$

figure 4.1

This matrix has the LU decomposition given in figure 4.2.

$$L = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ f/a & -bf/ac & 0 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} a & b & 0 & 0 \\ & c & 0 & 0 \\ & & d & 0 \\ & & & e \end{pmatrix}$$

figure 4.2.

This means that $L^TL$ has the form given in figure 4.3 which when premultiplied by $U^T$ and postmultiplied by $U$ gives the matrix of figure 4.4. In both cases these matrices are symmetric and only the pattern in the upper triangle has been given.

$$L^TL = \begin{pmatrix} 1+f^2/a^2 & -bf^2/a^2c & 0 & f/a \\ & 1+b^2f^2/a^2c^2 & 0 & -bf/ac \\ & & 1 & 0 \\ & & & 1 \end{pmatrix}$$

figure 4.3.

$$U^T(L^TL)U = \begin{pmatrix} a^2+f^2 & ab & 0 & fe \\ & b^2+c^2 & 0 & 0 \\ & & d^2 & 0 \\ & & & e^2 \end{pmatrix}$$

figure 4.4.

The matrix of figure 4.4 is indeed seen (from figure 4.1) to be identical to the matrix $A^TA$ as would be expected from equation (4.1) with matrices P and Q equal to the identity matrix. However, it is seen from figures 4.3 and 4.4 that $L^TL$ is not less dense

than $A^TA$. In fact, the element (2,4) (and, by symmetry, (4,2)) is zero in $A^TA$ but is non-zero in $L^TL$. This is because of numerical cancellation during the multiplications by U and $U^T$. On postmultiplication by U, element (2,4) has the value $-bef/_{ac}$ and on premultiplication by $U^T$ it becomes $-bef/_{ac} \times c + bef/_{a}$ which is clearly numerically zero although it would be non-zero were Boolean arithmetic used throughout. Conjecture 4.1 thus gives a correct numerical result inapplicable to the Booleanising treatment suggested in the conjecture.

Example 2. Consider the matrix of figure 4.5

$$A = \begin{pmatrix} a & & e & f \\ & b & & \\ & & c & \\ & & & d \end{pmatrix}$$

figure 4.5

This matrix will clearly give no fill-in if Givens' method is used. Since it is already upper triangular, the U of Givens' method will be identical to the matrix A of figure 4.5. If, however, the normal equations are formed, then the (3,4) element becomes ef. If the $LL^T$ decomposition is then performed on this normal matrix, the (3,4) element becomes $ef - ae/_{a^2}.af$ which is numerically zero due to exact cancellation, but which would be non-zero if Boolean arithmetic were used. The difference here is highlighted by observing that in Boolean arithmetic the decomposition of a matrix is always denser than the matrix itself, which, as is seen in this example, is not always the case. It should be noted that most sparse matrix solvers do not allow for this in their storage schemes. Thus, once again, conjecture 4.2 holds in a numerical sense but does not hold when Boolean matrices are considered.

Conjectures 4.1 and 4.2 and the two examples just quoted indicate that great care must be taken if numerical results are to be carried over to the Boolean case. The Booleanisation of algorithms is

discussed at some length by Brayton et al. (1970) and a further example of the difference between Boolean and numerical results is to be found in chapter one in the discussion on the Boolean inverse of a matrix. Willoughby (1971c) to some extent avoids this possibility of exact numerical cancellation by considering only M matrices. His results then carry over immediately to the Boolean case.

As is seen from the two examples, conjectures 4.1 and 4.2 do not hold because of numerical cancellation. This is seen empirically in section 5 where the results in table 5.1 show that conjecture 4.2 is incorrect while those in table 5.3 indicate that conjecture 4.1 does not hold. It may be argued that the results of section 5 are invalid because they exclude the possibility of exact numerical cancellation. This argument is countered by noticing that in some cases cancellation is dependent on the actual value of the initial non-zeros and, in general, sparse matrix algorithms themselves operate on a Boolean fill-in pattern without considering the above-mentioned type of cancellation.

It may, of course, be possible to develop an algorithm which takes advantage of the numerical result contained in conjecture 4.2 where, for example, the sparsity pattern of the decomposition of the normal matrix was determined by a QU Boolean decomposition (Q orthogonal) of the original matrix. Even then, however, numerical cancellation could upset this calculation. If such a method were devised, then the maximum amount of fill-in of Givens' method over the normal equations method might be determined by the use of the following theorem.

Theorem 4.1.

If the orthogonal decomposition of an arbitrary square matrix is carried out with only elements which were non-zero in the original matrix chosen as pivots, then the upper triangular matrix in the

decomposition is denser than the elementary factors of the orthogonal matrix.

Proof.

Without loss of generality, a permutation can be performed on the matrix so that the pivots lie on its main diagonal. Assume that, in the final QU decomposition, element (i,j) of Q (i > j) is non-zero. Then, in the stage of the decomposition where (j,j) is the pivot, (i,j) is non-zero as is element (i,i) and hence at this stage element (j,i) will become non-zero if it is not already. Hence, whenever an element (i,j) of Q is non-zero, element (j,i) of U is also non-zero and so the theorem has been proved.

The example in figure 4.6 indicates that the restriction of pivots to initial non-zeros is a necessary one.

$$\begin{pmatrix} x & 0 & 0 \\ 0 & x & x \\ x & x & 0 \end{pmatrix}$$

figure 4.6

In this matrix, if a QU decomposition is performed pivoting down the diagonal in the natural order, element (3,1) of Q is non-zero while element (1,3) of U is zero.

As regards a general comparison of the methods outlined in section 2, the following theorem is stated and proved.

Theorem 4.2.

For the same ordering, it is possible that the matrix $A^TA$ of the normal equations method is arbitrarily denser than the matrix $L^TL$ obtained by Wilkinson and Peters method or the orthogonal and upper triangular matrices obtained by QU decomposition.

Proof.

The proof is constructive and the result follows by considering the m×n matrix of figure 4.7.

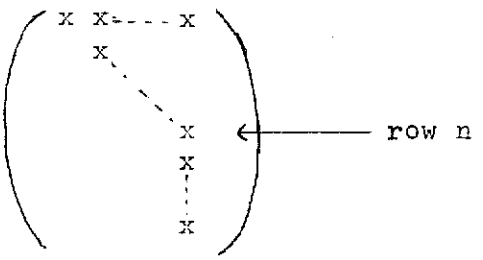



figure 4.7

In this matrix, the $L^TL$ of Wilkinson and Peters method is diagonal while in the orthogonal reduction methods $Q$ is an $m \times n$ matrix of the same structure as the lower trapezium of figure 4.7 while $U$ has the structure of the $n \times n$ block in figure 4.7. However, $A^TA$ is a full $n \times n$ matrix for any ordering of the rows and columns, and so the theorem has been proved.

Unfortunately, although this theorem indicates a possible danger in the use of normal equations, it does not establish a strict hierarchy of the methods since the example of figure 4.1 indicates that $L^TL$ may be denser than $A^TA$. In fact, the following theorem holds.

Theorem 4.3.

It is possible that the matrix $L^TL$ obtained by the method of Wilkinson and Peters is arbitrary denser than the matrix $A^TA$ of the normal equations method, if the same ordering is used in both cases.

Proof.

This proof is again constructive and follows easily if the example in figure 4.8 is considered.

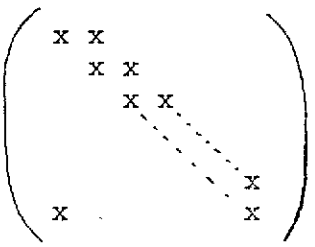


figure 4.8

This matrix will give $L^TL$ as a full matrix whereas $A^TA$ will be tridiagonal with non-zeros at the end of the first row and column. Hence the theorem has been proved.

Theorem 4.1 and conjecture 4.2 do indicate that the fill-in due to orthogonal reduction methods is likely to be less than that due to normal equations. However, the extent of this difference is totally dependent on the example under consideration (see table 5.1) and the additional computations and storage requirements needed in the orthogonalisation methods more than overbalances this reduction in fill-in.

The preceeding discussion in this section indicates the difficulties in a theoretical comparison of the various methods, and so comparisons are not made until after the experimental results in section 5. Some reflections are, however, made concerning the various methods, these reflections being based on the preceeding discussion and the counts given in the first two columns of table 2.1. The following approximate relationships concerning the densities of the various matrices involved in the solution processes are first stated (under the assumption that A is sparse) and then table 4.1 is obtained from table 2.1. As in table 2.1, the capital letters represent the number of non-zeros in the corresponding matrix.

(i) $1.5 \times A \sim L + U$ (LU decomposition of A)

(ii) $L \sim U$ (if A is square)

(iii) $L \sim Q$ ($QU_1$ decomposition of A)

(iv) $Q \sim L_1$ ($L_1DL_1^T$ decomposition of $A^TA$)

(v) $3Q \sim U_1$ (from table 3.2)

(vi) $L_1 \sim L_2$ ($L_2DL_2^T$ decomposition of $L^TL$)

| METHOD / COUNT | NORMAL EQUATIONS | GIVENS | HOUSEHOLDER | WILKINSON AND PETERS. |
|---|---|---|---|---|
| Total Store Required. | 2·3Q | 5Q - 2n | 4Q + n | 3Q - n |
| Number of Operations for Subsequent Right-hand sides. | 3·3Q - n | 7Q - 4n | 5Q + n | 4Q - 2n |

Table 4.1. Approximate counts when A is square and sparse. Q stands for the total number of non-zeros in the elementary factors of Q in the QU decomposition of A.

From table 4.1 and table 2.1, the following suggestions as to the behaviour of the various methods can be made.

(i) For square, fairly sparse systems both the method of Wilkinson and Peters and that of normal equations are better than the orthogonal reduction techniques. The two methods are somewhat similar in performance, with the former method triumphing when very sparse systems are considered (unlike the discussion in section 2 for full matrices). Unless the system is very sparse Householder's method will be better than Givens'.

(ii) As the density of the matrix increases, so does that of $A^TA$ and the matrix $L_1$ of its $L_1DL_1^T$ decomposition but the L and U in the LU decomposition of A are likely to be affected substantially hence causing the method of Wilkinson and Peters to suffer to a greater extent than that of normal equations. However, since Q will almost invariably be denser than L, the orthogonal reduction methods will also suffer and (by table 2.1 and 4.1) will become increasingly worse than the other two methods. In addition, the factors of n in table 2.1 will become less significant and the method of Givens will become considerably worse than that of Householder.

(iii) As the system becomes more rectangular, $A^TA$ will become increasingly dense because of there being more interactions between

columns due to their greater length. The effect on the LU decomposition of A (given that the density remains the same) should not be so marked as that due to (ii) above and so the relative merits of the methods of Wilkinson and Peters and normal equations should remain the same as in (i). Once again, due to increases in the number of non-zeros in Q the orthogonal methods should do worse, with Givens' method again performing rather less well than Householder's.

Hence, it can be remarked that the above discussion indicates that the method of Wilkinson and Peters may be expected to be rather more competitive than in the full case, particularly if the original matrix is fairly sparse. The extent to which this is true is seen in the results of the next section.

## Section 5. Some experimental observations on a comparison of the various methods.

In this section, the various methods mentioned in section 2 were tested. The matrices used were the same as those in section 3, namely the three structured matrices used throughout this thesis to which a row with a non-zero in its first column is added and various random matrices. The random matrices were of orders 50x50, 75x50, and 100x50 and, at each order, runs were made with matrices of densities 0.03, 0.05, and 0.10. In every case, ten runs at each of the nine combinations of order and density were made and the various quantities averaged to give the results given in the tables.

Perhaps the main problem in comparing methods is to see what particular characteristics of the methods should be used in the comparison. The two most important aspects of any sparse matrix algorithm are the amount of work that has to be done and the amount of storage that is used by the algorithm. It is these which are then used as the criteria for judging methods in this section. The operation counts and storage requirements for the various methods are given in table 2.1.

There now follows a description of each of the experiments performed followed by any conclusions which can be deduced from the results. The tables of results for these experiments are to be found at the end of this section.

### Experiment 1.

Runs were made testing Givens' method against the method of normal equations using the same ordering in each case. The row ordering within Givens was merely that of $\mathcal{R}$ of section 3. This experiment was run with a view to testing conjecture 4.2 in its Boolean sense and the results are given in table 5.1. The results in this table indicate that although none of the numerical cancellation described in section 4 occurs in the case of the structured matrices, it appears to happen in several of the random cases, particularly

where the density is low. In each case, the number of non-zeros in the upper triangle of the orthogonal decomposition is less than the number in the upper triangle of the $LL^T$ decomposition of the normal equations thus supporting the discussion following conjecture 4.2 and example 2 of section 4.

Experiment 2.

Clearly, it is helpful if, in the implementation of the method of Wilkinson and Peters, the number of non-zeros in the lower triangle of the LU decomposition of A is kept low because of the product $L^TL$ which is decomposed later in the process. The density of L and U is controlled by the pivot selection criterion used in the LU decomposition of A. Accordingly, runs were made on the usual selection of matrices using two different selection criteria for this decomposition. The first was the normal Markowitz criterion where the pivot selected at each stage was that non-zero which would minimise the maximum possible fill-in (viz. the non-zero whose product of row count and column count is a minimum). A modified form of this procedure which should minimise the growth in L was then tried. This was to select at each stage the column with minimum column count and to choose as pivot the non-zero element in that column with minimum row count. In this way the number of zeros added to L at each stage is maximised. The results are shown in table 5.2. They indicate that it is generally best, when implementing the method of Wilkinson and Peters, to try to keep the overall growth of non-zeros in the LU decomposition of A as low as possible. In this way, L is kept sparser than if a local order to add the maximum number of zeros at each stage is utilised. In each case, the final symmetric decomposition is carried out by choosing the diagonal element in the row with minimum row count as the pivot at each stage. The first method mentioned above is the one used in the comparisons in table 5.4 of this section since it generally does better, not only in preserving sparsity in L, but

also in reducing the number of operations for subsequent solution of right-hand sides, this being true over a range of sizes and densities.

Experiment 3.

Runs were made comparing the number of non-zeros in the $L_1DL_1^T$ decomposition of the normal matrix and the number of non-zeros in a similar symmetric decomposition of the matrix $L^TL$ obtained after LU factorisation of the original matrix. The same column ordering, the ordering chosen so that pivots in the decomposition of the normal equations lay in columns with minimum column count at each stage, was used in both cases and two row orderings for the LU decomposition were tried. The row orderings tried were those which chose the pivot on the basis of the non-zero in the column with

(i) minimum local fill-in

and (ii) minimum weighted local fill-in

where the weights taken were those given by the equation

$$\text{weighted fill-in of row } i = \sum_j f_{ij}(n-j) \qquad \ldots.(5.4)$$

where the sum is taken over all $j$ for which $a_{ij}$ is non-zero and $f_{ij}$ is the fill-in caused in column $j$ if row $i$ is chosen as the pivot row. In the two cases the final symmetric decomposition of $L^TL$ is performed by selecting as pivot at each stage the diagonal element with minimum row count. The results of experiment 3 are given in table 5.3. The following comments on these results can now be made. It is interesting to note that the a priori ordering of the columns does not affect the LU decomposition of $A$ as much as might be expected. This is probably because the column choice for pivoting in $A^TA$ selects a column which has had least interaction in the original $A$ and would thus be a good choice for pivot column in the LU decompositon of $A$ also. It is evident, if the columns are arranged a priori that (given a square system) any fill-in in the

last column cannot add to the density of the final L and fill-ins in earlier columns have proportionately greater chances of being in the final L. Thus, it comes as no surprise that the ordering using the weighted count is generally superior to the one using the unweighted one. As the system becomes more rectangular, so the weighting becomes less accurate and the results are less favourable to the weighted count. The results of this experiment clearly demonstrate that conjecture 4.1 is false. In almost every case, the number of non-zeros in $L^TL$ is significantly greater than the number in $A^TA$. However, it is interesting to note that it is generally true that the number of non-zeros in the decompositions of these two matrices is nearly the same and, in the sparsest cases, there are sometimes less non-zeros in the decomposition of $L^TL$ than in the decomposition of $A^TA$. This would seem to indicate (see table 2.1) that the method of Wilkinson and Peters will be comparable with that of normal equations being particularly well suited when A is sparse. It would also indicate that as A becomes denser it is as much the increase in density of the LU decomposition of A over A as anything which causes the Wilkinson and Peters method to become inferior.

Experiment 4.

All the runs necessary to produce the data for this experiment have already been made during the previous experiments. The results, which are in table 5.4, indicate the relative properties of the three methods under consideration as reflected in the matrices tested. The ordering for Householder's method was that used in section 3, namely to select the pivot on the basis of the non-zero with minimum row count among those lying in the column with minimum column count. The ordering for the normal equations was to choose as pivot the diagonal element with minimum row count. The criterion for pivot selection for the initial LU decomposition of A in the method of Wilkinson and Peters was Markowitz's criterion while the final symmetric decomposition was performed similarly to the strategy used

in the normal equations case.

From table 5.4., the results of this experiment and this section can be summarised in the following notes.

(i) If the matrix is square and sparse, then, on all counts, the method of Wilkinson and Peters is comparable with all the others doing particularly well if the initial LU decomposition of A does not introduce much fill-in. In this square and sparse case, the method of Householder is also a competitive method. This would seem to indicate that, for such cases, the fill-in to the upper triangle of the QU decomposition was overestimated in the discussion at the end of section 4.

(ii) As the matrix becomes denser, the method of Wilkinson and Peters becomes worse than that of the normal equations but, in general, only by a factor of about 20%. The method of Householder becomes very uncompetitive on the decomposition phase.

(iii) If the density is kept low and the matrix is made more rectangular, then the method of Wilkinson and Peters is best. However, the method of Householder suffers very badly from such a change.

(iv) It is only in the cases where the matrix is fairly dense and rectangular that the method of normal equations is significantly superior to the others and even then it is better than the method of Wilkinson and Peters by a factor of only about 25% on the number of operations for the solution of subsequent right-hand sides and about 50% on the storage and number of operations for the decomposition.

(v) The results, in general, support the conjectures made at the end of section 4.

Thus, it can be said that the work of this chapter has established that the method of Wilkinson and Peters performs comparatively well on general matrices which are not both dense and rectangular. It is therefore suggested that such a process should be used in any general

routine for the solution of sparse overdetermined sets of linear equations.

| METHOD \ MATRIX | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 DENSITY | | | RANDOM 75 x 50 DENSITY | | | RANDOM 100 x 50 DENSITY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| GIVENS Number of non-zeros in $U_1$ where $A=QU_1$ | 495 | 404 | 7313 | 207 | 492 | 960 | 316 | 623 | 1054 | 381 | 732 | 1141 |
| NORMAL EQUATIONS Number of non-zeros in $L_1$ where $A^TA=L_1L_1^T$ | 495 | 404 | 7313 | 244 | 501 | 960 | 323 | 627 | 1054 | 385 | 732 | 1141 |

Table 5.1. A comparison of Givens method (pivot choice $r_i$ within $c_j$ and rows ordered on $\mathcal{R}$) with normal equations using the same pivots and ordering. The results for random matrices are the average of five runs at the same size and density.

| METHOD \ MATRIX | | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 DENSITY | | | RANDOM 75 x 50 DENSITY | | | RANDOM 100 x 50 DENSITY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| Number of non-zeros in L where A=LU | Wilk.1 | 213 | 195 | 1069 | 99 | 149 | 338 | 128 | 193 | 692 | 167 | 246 | 954 |
| | Wilk.2 | 217 | 191 | 1225 | 92 | 157 | 357 | 154 | 287 | 736 | 192 | 356 | 1066 |
| Number of non-zeros in $L^TL$. | Wilk.1 | 463 | 318 | 7292 | 140 | 403 | 909 | 193 | 398 | 1053 | 234 | 440 | 1093 |
| | Wilk.2 | 463 | 305 | 7987 | 96 | 523 | 996 | 289 | 637 | 1103 | 338 | 669 | 1139 |
| Number of operations for subsequent right-hand sides | Wilk.1 | 1449 | 960 | 22132 | 395 | 1166 | 2542 | 717 | 1469 | 3156 | 903 | 1740 | 3498 |
| | Wilk.2 | 1420 | 937 | 23504 | 281 | 1315 | 2667 | 837 | 1791 | 3371 | 1046 | 2135 | 3650 |

Table 5.2. Comparison of two different methods for selecting pivots in the LU decomposition of A in Wilkinson and Peters method (Wilk.1 : Markowitz, Wilk.2 : min $r_i$ within min $c_j$). The results for random matrices are the average of five runs at the same size and density.

| METHOD \ MATRIX | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 | | | RANDOM 75 x 50 | | | RANDOM 100 x 50 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | DENSITY | | | DENSITY | | | DENSITY | | |
| | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| Number of non-zeros in $A^TA$ or $L^TL$ Normal | 391 | 361 | 1570 | 174 | 292 | 664 | 198 | 350 | 792 | 223 | 409 | 907 |
| Wilkinson and Peters 1 | 516 | 327 | 9080 | 151 | 484 | 1008 | 245 | 558 | 1121 | 287 | 605 | 1145 |
| Wilkinson and Peters 2 | 422 | 333 | 8291 | 142 | 456 | 981 | 248 | 550 | 1113 | 303 | 630 | 1151 |
| Number of non-zeros in decomposition of $A^TA$ or $L^TL$ Normal | 495 | 404 | 7313 | 244 | 501 | 960 | 323 | 627 | 1054 | 385 | 732 | 1141 |
| Wilkinson and Peters 1 | 533 | 339 | 9936 | 154 | 504 | 1034 | 309 | 676 | 1151 | 388 | 777 | 1195 |
| Wilkinson and Peters 2 | 436 | 333 | 9354 | 146 | 483 | 1010 | 312 | 677 | 1145 | 394 | 780 | 1194 |

Table 5.3. Comparison of the method of Wilkinson and Peters with the method of normal equations. In each case the column ordering is that which makes the pivot at each stage of the decomposition of the normal equations have minimum column count. The method of Wilkinson and Peters has the same column ordering but chooses the pivot within that column as the non-zero which introduces least local fill-in (Wilkinson and Peters 1) or least weighted local fill-in (Wilkinson and Peters 2), the weights being described in the text of experiment 3. The results for random matrices are the average of five runs at each size and density.

| METHOD \ MATRIX | | 55 x 54 | 58 x 57 | 200 x 199 | RANDOM 50 x 50 | | | RANDOM 75 x 50 | | | RANDOM 100 x 50 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | DENSITY | | | DENSITY | | | DENSITY | | |
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| OPERATIONS FOR DECOMPOSITION | Normal Equations | 3732 | 2708 | 217948 | 1077 | 3919 | 13129 | 1794 | 5958 | 16011 | 2454 | 7940 | 19070 |
| | Householder | 6248 | 2899 | 601971 | 558 | 5745 | 24436 | 5189 | 22004 | 60732 | 12053 | 41725 | 102342 |
| | Wilkinson and Peters | 5098 | 2217 | 393823 | 667 | 4410 | 17935 | 1753 | 6238 | 27269 | 2457 | 8210 | 30640 |
| TOTAL STORAGE REQUIRED | Normal Equations | 787 | 686 | 8192 | 367 | 673 | 1255 | 484 | 862 | 1474 | 583 | 1029 | 1686 |
| | Householder | 917 | 721 | 12578 | 303 | 800 | 1568 | 747 | 1514 | 2573 | 1185 | 2247 | 3619 |
| | Wilkinson and Peters | 901 | 632 | 6950 | 298 | 733 | 1607 | 467 | 892 | 2076 | 580 | 1054 | 2370 |
| NUMBER OF OPERATIONS FOR SUBSEQUENT RIGHT-HAND SIDES | Normal Equations | 1228 | 1033 | 15306 | 561 | 1124 | 2165 | 757 | 1439 | 2478 | 918 | 1711 | 2777 |
| | Householder | 1221 | 882 | 17551 | 370 | 1024 | 2101 | 1310 | 2520 | 4231 | 2114 | 3905 | 6243 |
| | Wilkinson and Peters | 1439 | 960 | 16132 | 405 | 1166 | 2542 | 717 | 1469 | 3156 | 903 | 1740 | 3498 |

Table 5.4. A comparison of the three methods, discussed in this chapter, for solving the least squares problem. In addition to the counts given for Householder's method in the decomposition phase there are, of course, n square root operations to be performed. The results for random matrices are the average from 5 runs.

# CHAPTER 8.

## SOME REFLECTIONS ON THE EIGENVALUE PROBLEM.

## Section 1. Introduction.

In this chapter, an investigation is begun into the eigenvalue problem for sparse matrices. The problem is to find, given a general sparse matrix $A$, its eigenvalues $\lambda_i$ (not necessarily distinct) and its eigenvectors $x_i$ such that

$$Ax_i = \lambda_i x_i \qquad \ldots\ (1.1)$$

Techniques for solving this problem when $A$ is full are found throughout the literature and details of the various methods can be found in, for example, Wilkinson (1965), Wilkinson (1966), and Fox and Mayers (1968). No attempt at the more general eigenvalue problem

$$Ax = \lambda Bx \qquad \ldots\ (1.2)$$

is made here, although a description of methods for the solution of (1.2) when $A$ is full can be found in Martin and Wilkinson (1968).

The eigenvalue problem breaks down into the two cases $A$ symmetric and $A$ unsymmetric. For general unsymmetric $A$, the normal technique is to reduce $A$ to Hessenberg form using either the plane rotations of Givens, the unitary transformations of Householder, or the elementary stabilised transformations of Gauss. The reductions are done in a symmetric fashion so that the eigenvalues of the reduced form are identical to that of the original matrix and the eigenvectors can be recovered easily. The eigenproblem for the Hessenberg form is then best solved, in general, by using the QR algorithm of Francis (1961, 1962) or by the Laguerre method (Parlett (1964)) for finding the eigenvalues, and finding the eigenvectors from them by inverse iteration. Section 2 is concerned with the reduction of the matrix to Hessenberg form. Tewarson (1970b) discusses some criteria for carrying out the reduction in a sparsity preserving context but gives no simple pivot selection criteria and no experimental results. This section attempts to remedy this and discusses the three reduction techniques mentioned above. It is

seen that in theory and in practice the only sensible way to perform the reduction in a sparsity context is to use the Gaussian reduction method of elementary stabilised transformations. Even then, it is found that the original system has to be fairly sparse for such a method to present any significant saving over considering the matrix as full.

In the symmetric case, orthogonal transformations are again performed on the original matrix in a symmetric fashion, this time reducing it to tridiagonal form. The eigenvalues of the tridiagonal form can then be found by the bisection method using the property of Sturm sequences followed by inverse iteration for the eigenvectors or can be found by again using the QR algorithm. In section 3, the reduction of a symmetric matrix to tridiagonal form is discussed. Again a paper by Tewarson (1970a) carried a sparsity-oriented discussion of this case but no experimental results. It is seen that, in spite of the results in section 2 which indicate that Givens' method is better than Householder's for preservation of sparsity, both methods produce such a growth in the number of non-zeros that there is no advantage in treating the matrix as other than full. In this section a symmetric storage scheme, mentioned in section 2 of chapter five, is described in detail.

It is often the case that the complete eigensystem of the matrix in equation (1.1) is not required, and only a few isolated or grouped eigenvectors or eigenvalues are needed. Some discussion of such problems is conducted in section 4 where, in addition, possible ways of overcoming the fill-in problems of the reductions in the symmetric case are considered.

In conclusion, it can be said that while it may generally be possible to utilise current sparsity techniques in the unsymmetric eigenvalue problem, the case of determining the complete eigensystem for a sparse symmetric matrix is still a very open one and the

suggestions in section 4 can only be considered a very tentative step in this direction.

## Section 2. The unsymmetric eigenvalue problem.

As was mentioned in the introduction to this chapter, the kernel to most methods for calculating the complete eigensystem of an unsymmetric matrix lies in reducing the matrix under consideration to Hessenberg form. A matrix A is said to be in upper Hessenberg form if $a_{ij} = 0$ whenever $i > j+1$. The lower Hessenberg form is similarly defined. This section deals with methods of reducing the given matrix to such a form by means of similarity transformations. The three methods discussed in this section are the two orthogonal similarity transformations of Givens and Householder and the transformation, using elementary matrices, of Gauss (Wilkinson (1965) has some details of these methods). While it is in general true that if Gaussian reduction is performed more care has to be taken against induced instability, as is the case in the solution of linear equations, it is certainly true that there is less fill-in using the Gaussian method than if either of the orthogonal methods is used. This is formalised in theorems 2.1 and 2.2, but, before these theorems are stated and proved, the methods are first discussed from a sparsity point of view. (Chapter 6 of Wilkinson (1965) describes the methods when applied to full matrices).

### Givens Method.

A description of the first step of this method is first given and then a general account of the fill-in caused by it is presented.

The rows (and cols) of the matrix are first permuted so that the element (2,1) is non-zero. The logical sum of this pivot row and every row (apart from the first two rows) with a non-zero in the first column is taken in turn and is used to replace these two rows in the matrix, this process continues for some ordering of rows where at each stage the pivot row is that which is left after the previous logical sum operation. An identical sequence of logical operations is then carried out with column 2 and similarly indexed columns of

the matrix. Consider the example in figure 2.1

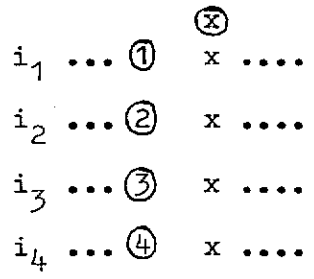



figure 2.1

Let $\oplus$ denote the logical sum ($1+1 = 1$) operating on rows of a matrix. Then, the stages of the reduction on the above matrix, given the ordering $i_2 < i_4 < i_3$, are as follows :

$$\textcircled{1} \leftarrow \textcircled{1} \oplus \textcircled{2}$$
$$\textcircled{2} \leftarrow \textcircled{1}$$
$$\textcircled{1} \leftarrow \textcircled{1} \oplus \textcircled{4}$$
$$\textcircled{4} \leftarrow \textcircled{1}$$
$$\textcircled{1} \leftarrow \textcircled{1} \oplus \textcircled{3}$$
$$\textcircled{3} \leftarrow \textcircled{1}$$

and a similar sequence of operations is then performed on columns $i_1, \ldots . i_4$ with the same ordering as above. This happens irrespective of the zero, non-zero structure of the matrix bar that utilised in determining rows 1 to 4 and their ordering.

**Premultiplication**:

From this description, there is a fill-in in position $(i,j)$ due to premultiplication at stage $k$ if and only if $a_{ik} \neq 0$ and there exists an $l$, $k+1 \leq l < i$ such that $a_{lk} \neq 0$ and $a_{lj} \neq 0$. This is shown in figure 2.2. where the element $a_{ij}$ fills-in during Givens premultiplication.

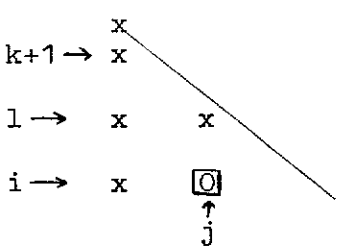



figure 2.2

Postmultiplication:

By the discussion of the previous page, fill-in at the postmultiplication stage of the element $a_{ij}$ will require that column $j$ be summed with column $k+1$, which means that row $j$ had been summed with row $k+1$ at the premultiplication stage, which in turn means that element $a_{jk}$ has to be non-zero. Further, fill-in due to postmultiplication will then occur if there exists a column $l$ $(k+1 \leq l < j)$ such that $a_{il} \neq 0$ and it is summed with column $k+1$ which will happen if $a_{lk} \neq 0$. This is necessary for fill-in to occur and it is clearly seen to be sufficient. Hence, zero element $a_{ij}$ will be filled in during postmultiplication if and only if $a_{jk} \neq 0$ and there exists an $l$ $(k+1 \leq l < j)$ such that $a_{il} \neq 0$ and $a_{lk} \neq 0$.

Since, for the whole process, one has to consider both pre and postmultiplication, it is possible (considering premultiplication as being performed first) that element $a_{il} = 0$ in the original matrix but is made non-zero by the premultiplication. This phenomenon is called interaction between pre and postmultiplication. An example of this and of fill-in due to postmultiplication now follow in figures 2.3 and 2.4.

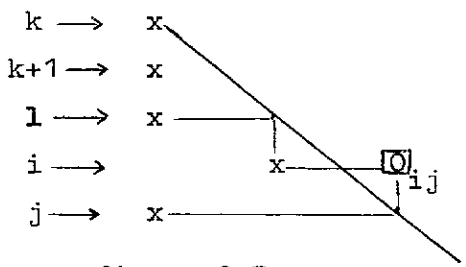



figure 2.3.

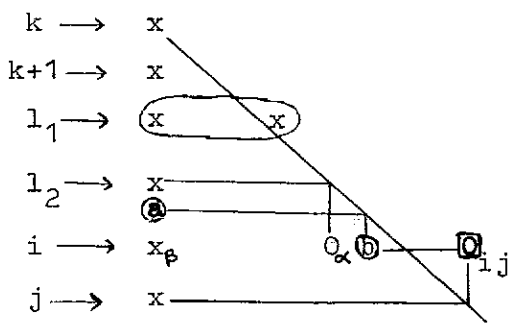



figure 2.4.

In figure 2.3, the element (i,j) will be filled-in by Givens' postmultiplication. Figure 2.4 gives an example of 'interaction'. Element $\alpha$ is filled in by premultiplication causing the element (i,j) to fill-in on subsequent postmultiplication. If all pairs ⓐ & ⓑ are such that both are not non-zero, then no such fill-in (of (i,j)) would occur if postmultiplication were considered in isolation.

If the diagonal elements are non-zero, then the above 'interaction' will not occur. This is because, if $\beta$ is non-zero, (necessary for the $\alpha$ fill-in on premultiplication), then: (i) if $i < j$, on postmultiplication, column j becomes the logical sum of columns k+1, i, and j and so $a_{ij}$ is filled-in by postmultiplication alone; (ii) if $i > j$, premultiplication fills in element $a_{ij}$. Hence, element (i,j) will be filled-in even without considering 'interactions' which are thus redundant concepts in this case of matrices with non-zero diagonals.

To sum up this reduction of Givens, at stage k element $a_{ij}$ will be filled-in if

$$a_{ik} \neq 0 \quad \text{and} \quad a_{lk} \cdot a_{lj} \neq 0 \qquad (k+1 \leq l < i),\ (i,j \geq k+1) \qquad \ldots\ (2.1)$$

or

$$a_{jk} \neq 0, \qquad a_{lk} \neq 0 \qquad (k+1 \leq l < j)$$

and either

$$a_{il} \neq 0 \quad \text{or} \quad a_{ik} \neq 0 \quad \text{and} \quad a_{rk} \cdot a_{rl} \neq 0 \ (k+1 \leq r < i) \qquad \ldots\ (2.2$$

$$(j \geq k+1)$$

Householder's Method

The following discussion establishes that premultiplication by the Householder orthogonalisation matrix affects the same rows as the method of Givens and that each of these rows is replaced by the logical sum of all of these rows. The postmultiplication causes a similar operation to be performed on the corresponding columns of the matrix left after the premultiplication phase.

The Boolean structure of the Householder matrix at stage k of the reduction process is given by the equation

$$H = I + ww^T \qquad \ldots\ (2.3)$$

where Boolean addition and multiplication is used and the n x 1 (0,1) vector w is given by the equation

$$\begin{aligned} w_i &= 0 \qquad (i \leqslant k) \\ w_{k+1} &= 1 \qquad i=k+1 \end{aligned} \qquad \ldots\ (2.4)$$

$w_i = 1$ if and only if $a_{ik}$ is non-zero. $(i > k+1)$

where $a_{ik}$ is the (i,k)th element of the matrix before stage k of the reduction. The matrix, at this stage, is A.

In the following detailed discussion of pre and postmultiplication all the matrices and operations are assumed to be Boolean.

Premultiplication.

Consider the element (i,j) of the semi-reduced form HA. Then,

$$\begin{aligned} (HA)_{ij} &= a_{ij} + w_i(w^T A)_{\cdot j} \\ &= a_{ij} + w_i(w^T A_{\cdot j}) \end{aligned} \qquad \ldots\ (2.5)$$

and so it will be equal to 1 if either $a_{ij} \neq 0$ or $w_i = 1$ and there exists an l $(l \geqslant k+1)$ such that $w_l = 1$ and $a_{lj} \neq 0$. That is, fill-in will occur in the premultiplication phase if $a_{ij} = 0$ and

$$a_{ik} \neq 0 \text{ and } a_{lk} \cdot a_{lj} \neq 0 \text{ for some } l \geqslant k+1 \qquad \ldots\ (2.6)$$

An example of this is shown in figure 2.5.

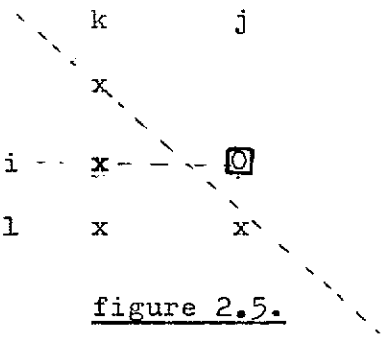



figure 2.5.

This clearly supports the above-mentioned statement that the sparsity pattern of the affected rows is the logical sum of all the rows (from k+1 to n) with a non-zero in column k.

Postmultiplication:

First postmultiplication is considered in the absence of the type of interactions mentioned in the discussion of Givens' method. Then,

$$(AH)_{ij} = a_{ij} + (Aw)_i w_j^T$$
$$= a_{ij} + (A_{i\bullet}w)w_j^T \quad \ldots\ldots (2.7)$$

and so, for fill-in to occur due to postmultiplication alone, $a_{ij}$ must equal zero, $w_j \neq 0$, and there exists an $l (l \geq k+1)$ such that $a_{il} \neq 0$ and $w_l \neq 0$. That is, $a_{ij} = 0$ and

$$a_{jk} \neq 0, \quad a_{lk} \cdot a_{il} \neq 0 \quad \text{for some } l \geq k+1 \quad \ldots\ldots (2.8)$$

where $i, j \geq k+1$.

An example of this is shown in figure 2.6.

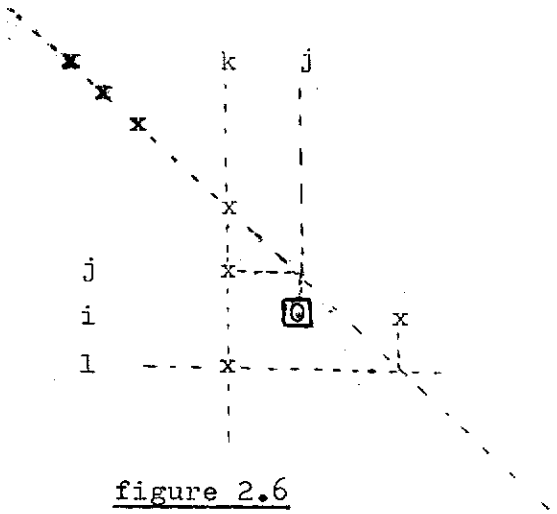



figure 2.6

Again, this supports the previous discussion since it is evident that fill-in due to postmultiplication occurs in columns corresponding to rows affected during premultiplication. Each of the columns thus affected is replaced by the logical sum of such columns.

Clearly, the only second order effect will be due to the element $a_{il}$ being filled in in the premultiplication phase of the reduction process. This could happen in the following manner as indicated

in figure 2.7 (here, without loss of generality, the case $k = 1$ is considered)

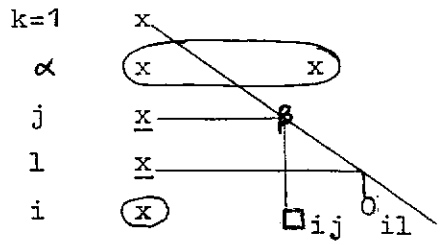



figure 2.7

If the ovaled elements are non-zero ($\propto$ any row other than k), then the element $a_{il}$ will be non-zero after the premultiplication causing fill-in at $a_{ij}$ in the postmultiplication phase due to the underlined non-zeros in the figure.

It is worth noting that, in order for this interaction effect to occur at $(i,j)$ the element $a_{ik}$ must be non-zero. However, if, in addition, the diagonal of the matrix is non-zero, then the element $\beta$ is non-zero in the above diagram indicating that fill-in will occur at $(i,j)$ in the premultiplication phase due to the elements $a_{jk}$, $a_{ik}$, and $\beta$ and so no interaction can occur. This possibility of interaction is included in

$$\begin{aligned} &a_{jk} \neq 0, \quad a_{lk} \neq 0 \quad \text{and} \\ \text{either} \quad &a_{il} \neq 0 \quad \text{for some} \quad l \geqslant k+1 \\ \text{or} \quad &a_{il} = 0 \quad \text{and} \quad a_{ik} \neq 0, \quad a_{rk} \cdot a_{rl} \neq 0 \quad \text{for some} \quad r > k+1 \\ \text{where} \quad &j > k+1 \end{aligned} \qquad \ldots\ (2.9)$$

This condition for fill-in due to postmultiplication completes the description of fill-in in Householder's method.

Gauss's Method.

Gauss reduction is essentially different from that of Givens and Householder by virtue of the fact that it is an unsymmetric reduction. Again, it can be considered in two stages. A premultiplication stage and a postmultiplication one.

Premultiplication

Premultiplication involves subtracting multiples of the k+1 th row from other unreduced rows in the matrix which have a non-zero in column k. Fill-in at this stage is exactly as in Gaussian elimination for the solution of linear algebraic equations. It is illustrated in figure 2.8 below.

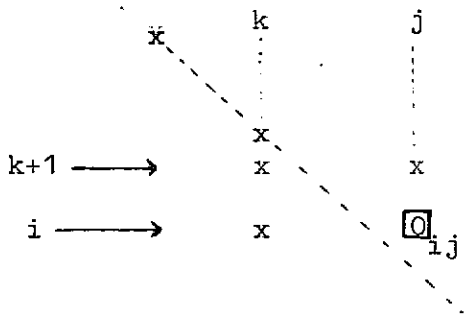



figure 2.8

This is formalised by saying that a fill-in will occur in position (i,j) if $a_{ij} = 0$ and

$$a_{ik} \neq 0 \quad \text{and} \quad a_{k+1,j} \neq 0 \qquad \ldots\ldots (2.10)$$

where $i,j > k+1$

Postmultiplication.

Postmultiplication by the inverse of the Gaussian reduction matrix G (to preserve eigenvalues ... a similarity transformation has to be used) can only cause fill-in in the k+1 th column of A.

Since the sparsity structure of the k+1 th column of G is given by w where

$$w = \begin{cases} w_i = 0 & (i < k+1) \\ w_{k+1} = 1 & \\ w_i = 1 \text{ if and only if } a_{ik} \neq 0 & (i > k+1) \end{cases} \qquad \ldots\ldots (2.11)$$

it holds that

$$(AG)_{ik+1} = A_{i\bullet} w \qquad \ldots\ldots (2.12)$$

and so there will be fill-in due to postmultiplication (again interactions because of premultiplication are neglected at this stage) in position (i,k+1) if $a_{i\ k+1} = 0$ and there exists an l

such that $w_1 \neq 0$ and $a_{il} \neq 0$, that is if

$$a_{lk} \neq 0 \quad \text{and} \quad a_{il} \neq 0 \qquad (l \geq k+1) \qquad \ldots\ldots (2.13)$$

An example of this fill-in is given in figure 2.9.

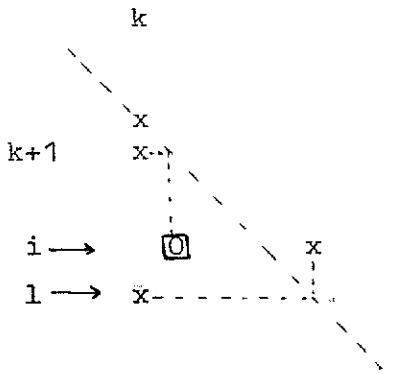



figure 2.9.

Again it is possible, as with Givens' and Householder's methods of reduction, to have a double effect. That is, if premultiplication is performed first, it is possible for an element filled-in at this stage to give rise to a fill-in on postmultiplication which would not have occurred were postmultiplication carried out first. The following example in figure 2.10 illustrates this point.

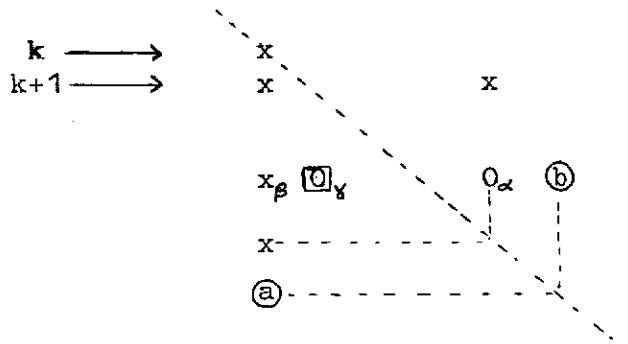



figure 2.10.

The fill-in at $\gamma$ is caused on postmultiplication since element $\alpha$ is made non-zero at the premultiplication stage. If no pair (a,b) exists such that (a) and (b) are both non-zero, then fill-in would not have occurred if postmultiplication were carried out alone. This is an example of interaction. Note that $\beta$ has to be non-zero

to cause fill-in at $\alpha$ in the premultiplication phase.

However, if the diagonal elements of A are non-zero, this interaction effectively will not take place, since the element $\gamma$ would be filled-in at the premultiplication stage. Thus, for matrices with non-zero diagonals, the post and premultiplication phases can be considered independently, as in the case of Givens and Householder. Of course, the same zero of A may be filled in by both the premultiplication and the postmultiplication.
This possibility of interaction is included in

$$\begin{aligned} & a_{lk} \neq 0 \quad (\text{where } l \geq k+1) \quad \text{and} \\ \text{either} \quad & a_{il} \neq 0 \\ \text{or} \quad & a_{il} = 0 \quad \text{and} \quad a_{ik} \neq 0, \ a_{k+1\,l} \neq 0 \end{aligned} \qquad \ldots\ (2.14)$$

This conditon for fill-in due to postmultiplication completes the sparsity-oriented description of Gauss's Method.

As a first step towards establishing a hierarchy for these three reduction methods with regard to preservation of sparsity the following lemma is stated and proved.

Lemma 2.1.

If, at a single stage of the reduction on any sparsity pattern, method A produces more fill-in than method B, then, in the complete reduction process, method A cannot produce less fill-in than method B if the same pivotal sequence is used in each case.

Proof.

Consider the two methods working on the same matrix M. Then, if AM is the result of conducting the first major reduction step of method A on the matrix M,

$$\text{Structure } (AM) \supseteq \text{Structure } (BM) \qquad \ldots\ (2.15)$$

where the '$A \supseteq B$' means that the non-zero pattern of B is contained in that of A.

Clearly, if method B is performed a second time on structure BM, then any new non-zeros introduced by this reduction are either in structure AM or would be introduced were method B implemented on structure AM. That is,

$$\text{structure (BAM)} \supseteq \text{structure (BBM)}$$

but, by equation (2.15) with AM replacing M,

$$\text{structure (AAM)} \supseteq \text{structure (BAM)}$$

thus giving the inequality

$$\text{structure (AAM)} \supseteq \text{structure (BBM)} \qquad \ldots. \quad (2.16)$$

Repeated applications of the logic leading to equation (2.16) indicate the proof of this lemma, since the matrix M can be first ordered so that pivots are chosen along the diagonal for both methods.

It is, of course, possible that, even if the inequality (2.15) is strict, the final one will not be. This is borne out by the results of a few runs made with the Givens and Householders methods on structured matrices. That a strict inequality does sometimes hold is shown in the examples which follow the next two theorems.

The following two theorems establish the required hierarchy of methods.

Theorem 2.1.

Given the same pivotal strategy, Householder's method of reduction to Hessenberg form will never produce less total fill-in than the method of Givens.

Proof.

This follows easily from the prior discussion of the methods where, from equations (2.1), (2.6) and (2.2), (2.9), it is seen that Givens' and Householder's methods satisfy the conditions of methods B and A respectively in lemma 2.1. Use of this lemma then completes the proof of this theorem.

An example which indicates that strict inequality may occur is given in figures 2.11(a) and (b).

$$\begin{pmatrix} X & X & X & 0 & X \\ X & 0 & X & 0 & 0 \\ 0 & 0 & X & X & 0 \\ X & 0 & X & 0 & 0 \\ X & X & 0 & X & 0 \end{pmatrix}$$

figure 2.11(a)

$$\begin{pmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ X & 0 & X & 0 & 0 \\ X & X & X & X & X \end{pmatrix}$$

figure 2.11(b)

Figure 2.11(b) indicates the sparsity structure after one major step of Givens on the matrix of figure 2.11(a). It is seen from this that the method of Givens on this matrix will produce a total fill-in of 10. However, at the first step alone, Householder's method produces a fill-in of 11 on the matrix of figure 2.11(a).

The following theorem establishes Gauss's method as the best for an unsymmetric matrix in a sparsity preservation sense.

Theorem 2.2.

Given the same pivotal strategy, Givens' method of reduction to Hessenberg form will never produce less total fill-in than the method of Gauss.

Proof.

Again nearly all of the work for this proof has been done in the discussion of the three methods which preceeded these theorems.

Since, in both methods, it is assumed that $a_{k+1\,k}$ is non-zero, equations (2.1) and (2.10) indicate that the local fill-in due to Givens is never less than the local fill-in due to Gauss in the premultiplication phase. The same is true of the postmultiplication phase when equations (2.2) and (2.14) are considered with $j = k+1$ in equation (2.2). Thus, the conditions of lemma 2.1 hold and the proof of this theorem follows similarly to the proof of theorem 2.1.

Again it is possible to give an illustration, shown in figure 2.12, of a matrix for which a final strict inequality holds.

$$\begin{pmatrix} X & X & 0 & 0 \\ X & X & 0 & X \\ X & X & X & 0 \\ X & 0 & 0 & X \end{pmatrix}$$

figure 2.12

In Givens method, total fill-in occurs when it is implemented on the matrix of figure 2.12. However, if the method due to Gauss is employed, then elements (1,3) and (1,4) will not fill-in during the reduction process.

Some runs were made using these three reduction methods on a number of different matrices. The examples tried were the unsymmetric structured matrices of orders 54, 57 and 199 used elsewhere in this thesis and random matrices of orders 50, 75 and 100 and various densities. The results are shown in table 2.1 and bear out the remarks of this chapter, although the pivot selection criteria, since they were local ones, could have caused different pivots to be selected in each case. In addition, the results of chapter seven support theorem 2.1 which, in turn, supplies a justification for the remarks made in that chapter. Tewarson (1970b) has a discussion of pivoting strategies for these reduction methods, but these are based on an exact determination of the fill-in at each stage and would take too long for a feasible implementation. It seems unlikely that they would produce a very significant improvement over the method used here. For Householder's and Gauss's methods, the pivot selection criterion was to select, at each stage, the row with minimum row count as the pivot row. This was also done in the case of Givens where, in addition, the remaining rows with non-zeros in the pivot column were ordered in order of ascending row count.

The following notes on the results of table 2.1 can now be made.

(i) The important counts are those shown in the table. Since it is seldom desired to retain all of the transformations (inverse iteration

is used to find the eigenvectors), the principal storage problem lies in accommodating the maximum number of non-zeros which have to be held at any one stage rather than the total number created during the reduction process. The number of non-zeros in the final Hessenberg form is also of interest because of subsequent operations done on that form to complete the solution to the eigenproblem (see sections 1 and 4).

(ii) In all methods it was found to be important to do some form of pivoting. For example, if pivots were chosen in the natural order using Gauss' reduction on the 57 x 57 case, then the total fill-in was 2047, the maximum number of non-zeros held was 949, and the final number of non-zeros in the Hessenberg form was 928. The methods of Householder and Givens also give greater values for all these three quantities if pivoting is not used. Thus, particularly in the case of Gauss, pivoting of the kind mentioned before should be used if full advantage is to be taken of theorem 2.2 and Gauss's method is to become feasible.

(iii) If the results of Givens' and Householder's methods are compared it is seen that the total fill-in in both cases is almost always the same. Part of the reason for this is that, in addition to the pivot row after each stage in both cases being the logical sum of rows with non-zeros in the pivot column, the corresponding column is also a similar logical sum in each case, thus making the identical fill-in quantities more significant than in the cases considered in chapter seven. The reason why occasionally Householder does better than Givens (apparently contrary to theorem 2.1) is because of a different pivotal sequence in each case. It is seen that Householder's method has, as in the full case, about half the number of operations of Givens's method and so, since other things are about equal, would be the method preferred between these two.

(iv) For both Givens and Householder, it is observed that, apart

from the very sparse random cases, the maximum number of non-zeros held in these reduction processes is very nearly the number in a full matrix of the original size. This fact alone makes either of these methods unsuitable for use in a sparsity context.

(v) Fortunately, this particular count for the Gaussian method is down by a factor of about 3 and is consistently much the same as the final number of non-zeros in the Hessenberg form. Although this is always much less than the total number of non-zeros in a full matrix of the same order, the last row in the table indicates that not much advantage can be gained by considering the final Hessenberg form as other than full unless the matrix under consideration is fairly small and sparse.

(vi) The results of this section and the notes (i) to (v) indicate that a suitable method for reducing a sparse matrix (held in packed form) to Hessenberg form is that of Gauss, the methods of Givens and Householder being totally unsuitable.

## Section 3. The symmetric eigenvalue problem.

In this section, the symmetric reduction of a general symmetric matrix to tridiagonal form (this is a symmetric Hessenberg form) is considered. Of the methods considered in the last section, only two are symmetric reductions namely those due to Givens and Householder. It is, of course, possible to consider the matrix as an unsymmetric one and to reduce it to upper Hessenberg form using Gauss reduction as mentioned in the previous section. Although this takes advantage of the comparatively low fill-in of Gauss's method (see the results in table 2.1) it does, of course, lose the considerable advantage obtained by only storing one half of the matrix and only effectively performing one half the number of operations. A mention of this possibility of treating the symmetric matrix as unsymmetric is made in the notes discussing the results in table 3.2 and the possibility of using Gauss in this fashion is considered there.

It is evident that the equivalent of theorem 2.2 holds in the symmetric case. The description of the two methods are similar, and it still holds that, after a major stage in Givens premultiplication, the rows with non-zeros in the pivot column have a sparsity structure corresponding to the logical sum of the sparsity structure of the preceeding rows with non-zeros in the pivot column. In Householder's method, each row is the logical sum of all the other rows and not just the ones preceeding it. A similar thing happens to the corresponding columns of the matrix in each case. The statements and the discussion of the previous section lead to a statement of the following theorem for which an exact proof is unnecessary.

Theorem 3.1.

In the reduction of a symmetric matrix to tridiagonal form using symmetric orthogonal reductions, the total fill-in using Householder's method is never less than that using Givens' method if similar pivots are chosen in each case.

In table 3.1 results are given of runs of Givens' and Householder's methods on several structured and random matrices. The results of this table give an experimental verification of theorem 3.1. It is, in addition, observed that, in every case, the final total fill-in is the same. This is not really surprising since, at stage $k$, the final status of row and column $k+1$ must be identical if the structure was the same at the beginning of the stage, and it is required for a difference in total fill-in that the structure of this column and row should be different. This is different from the unsymmetric case discussed earlier. However, a small example indicates that the invariance of Householder to row interchanges within its major step can provide a counter-example. This is given in figure 3.1 where it is observed that, because of the property of Givens being used it is impossible to preorder the matrix so that natural ordering is employed.

$$\begin{pmatrix} X & X & X & X & 0 \\ X & 0 & 0 & 0 & 0 \\ X & 0 & 0 & 0 & X \\ X & 0 & 0 & X & 0 \\ 0 & 0 & X & 0 & 0 \end{pmatrix}$$

figure 3.1

After one stage of reduction the matrices resulting from Givens and Householder reduction are given in figures 3.2(a) and 3.2(b) respectively.

$$\begin{pmatrix} X & X & 0 & 0 & 0 \\ X & X & 0 & X & X \\ 0 & 0 & 0 & 0 & X \\ 0 & X & 0 & X & X \\ 0 & X & X & X & 0 \end{pmatrix} \begin{matrix} \\ \\ \longleftarrow \\ \longleftarrow \\ \\ \end{matrix}$$

figure 3.2(a)

$$\begin{pmatrix} X & X & 0 & 0 & 0 \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & X \\ 0 & X & X & X & 0 \end{pmatrix}$$

figure 3.2(b)

At the second stage the rows (and columns) marked '$\longleftarrow$' in figure 3.2(a) are interchanged in both cases and the reduction is performed again. This gives the total fill-in for Givens as 11, 3 less than

for the method of Householder.

In practice, the advantage to be gained from using Givens over Householder is dependent on being able to choose orderings so that cases like those in figure 3.1 and 3.2 occur. The gains made are seen empirically in table 3.2 where Givens method has been compared against Householder's for symmetric matrices. Some comments can now be made concerning the results in this table.

Notes.

(i) It can first be observed as a programming aid that, if the diagonal of the matrix is non-zero, all the fill-in can be observed by witnessing the effect of premultiplications only since if $a_{ij}$ fills-in on subsequent postmultiplication then either $a_{ij}$ or $a_{ji}$ must have filled-in in the prior premultiplication.

(ii) Again it is advisable to use some sparsity criterion to select pivots; for example, to choose as pivot row that row which has minimum row count. As opposed to chapter seven, the effect of row ordering within a Givens major step has virtually no effect on the total fill-in. In addition, the criterion for pivot selection did not effect this count greatly. This was observed in several runs on similar matrices which are not recorded in table 3.2. This would imply that the use of more complicated ordering techniques as in Tewarson (1970a) would do little to improve the situation. The reason for this is simply the double fill-in effect due to pre and postmultiplication which causes an immediate and rapid growth in the number of non-zeros whatever pivoting and row ordering criterion is employed.

(iii) Although the fill-in is much less than in the unsymmetric case (for example, at stage $k$, there can be no fill-in in the rows 1 to k), the results of table 3.2 indicate that no advantage can really be taken of any sparsity in the original matrix (unless it is very sparse and small) when reducing a symmetric matrix to tridiagonal form. The matrix should be considered full and Householder reduction should

generally be preferred since its operation count is about half that of Givens.

(iv) It may be debated whether any advantage can be gained by considering the matrix as unsymmetric and using Gauss reduction thus utilising theorem 2.2 and the results of the last section at the cost of having to store all of the matrix. A comparison of tables 2.1 and 3.2 indicates that there would be, in general, a slight saving in storage and a substantial saving in operation count were such a method implemented. However, the saving in storage is only slight and is more than counterbalanced by the greater instability of Gaussian reduction and the fact that it requires more work to complete the eigensolution of a Hessenberg matrix than a tridiagonal one.

(v) The results of this section and the notes given above indicate that, if it is desired to reduce a general symmetric matrix to tridiagonal form, it is generally best to regard it as full and reduce it by means of Householder transformations. Attention should, of course, be paid to the particular structure of the matrix and to the information concerning the eigensystem which is required (see section 4).

As was mentioned in section 2 of chapter five, a storage scheme has been developed to handle symmetric matrices and is based on that used in Method Y of Curtis and Reid (1971d) for unsymmetric ones. A fuller description of it than was given then is now presented and a program implementing this storage scheme is given in the appendices. The description of this scheme (reference should be made to the figure and notes on page 119) takes the form of being a detailed description of the program in the appendices.

The following arrays are used in the program, most of them being integral to the storage scheme. The matrix is assumed to be of order n and to have, initially, KA non-zeros.

R(i) i=1,N — Set to 1 initially and R(i) put to zero when element (i+1,i) is used as pivot.

B(i) i=1,N — Points to address in array S of beginning of column (or row) i of the matrix.

DIAG(i) i=1,N — Points to address in array S of diagonal element of column (or row) i of the matrix. Not necessary but used in this implementation.

O(i) i=1,N — Contains column (and row) index of the i th element in the pivotal ordering. At stage k of the elimination, elements O(k+2).... O(n) are undefined.

S(i) i=1,4×KA,4 — This array gives information concerning the i th non-zero of the matrix where the elements are counted down the columns in the natural order starting with the first non-zero in each column and finishing with the diagonal element. There is, of course, extra space defined in S after $4 \times KA$ to allow for an increase in the number of non-zeros during the elimination process.

Each unit of 4 is as follows:

| row index | column index | pointer to next * element in the row | pointer to next + element in the column |
|---|---|---|---|

* Is zero for the last element in a row

\+ Is -1 for the diagonal element.

Columns (rows) are accessed in a dog-leg fashion as is shown in figure 3.3

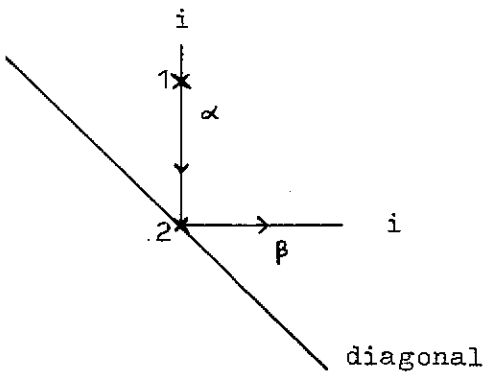



figure 3.3.

For row (or column) i, B(i) points to element 1 while DIAG(i) points to element 2.

For part $\alpha$ : K = 1 and for part $\beta$ : K = 0

and,

for element in position W in S, the row index is given by W+1-K and the pointer to the next element in the column is given by W+2+K, both of these numbers being positions in the array S.

The program, given in the appendices, which utilises this scheme determines the fill-in due to the method of Householder. It uses the scheme described below which is generally unsatisfactory because of the search for zero-headed columns. Because of the high fill-in in Householder's method (see table 3.2) it is, however, not so inefficient as might be expected.

Process for determining fill-in due to Householder reduction.

An initial row and corresponding column is chosen as pivot. In this case, let it be row and column 1. (The pivot element in the Gauss or Givens sense is the element (2,1)).

(i) The row (column) selected as pivot row (column) is searched until a non-zero is found, this row is selected as the next pivot row for the reduction, signified by placing its index in the next position in array O. If no such non-zero is found then the pivot row is a singleton and it is possible to proceed with the next stage in the elimination.

(ii) The search along the pivot row continues for further non-zeros in uneliminated columns. If none exist then a doubleton exists and again it is possible to proceed to the next step in the elimination process. For any columns Y with such a non-zero in the pivot row a search is made down that column in conjunction with the pivot column introducing a fill-in into the column when a row is encountered with a non-zero in the pivot column and a zero in column Y . This is shown in figures 3.4(a) and 3.4(b). This search need only extend

from the physical beginning of column Y, B(Y), to its diagonal element (DIAG(Y) or W such that S(W) = S(W+1), if DIAG is not used) where, of course, previously eliminated rows are ignored. The algorithm goes back to (ii) and continues until the pivot row is exhausted.

(iii) The pivot row is now retraced from the beginning looking for columns with a zero in that row, the columns again being uneliminated ones. This column, Z , say, is traced down from its physical beginning round its dogleg to its physical end (S(W+2)= 0) in conjunction with the pivot column. This complete trace is a necessary one, as will be shown later. The occurrence of non-zeros in the two columns in the same uneliminated row is looked for. If not found, the tracing (iii) for more columns with zeros in the pivot row is continued. If found, the pivot column and column Z are retraced this time looking for an uneliminated and non-pivotal row with a zero in column Z and a non-zero in the pivot column. Such a zero is filled in and the scan continues until there are no more non-zeros in the pivotal column. This is shown in figures 3.5(a), 3.5(b), 3.6(a) and 3.6(b). The algorithm then goes back to (iii) and continues thus until all the columns with zero in the pivot row have been dealt with.

The algorithm then goes back to (i) and continues the whole process for n-2 steps by which time the reduction is complete.

In the following figures, (a) refers to the scheme as if the matrix were held completely in core, while (b) refers to the storage scheme whereby only the upper triangular part of the matrix is held.

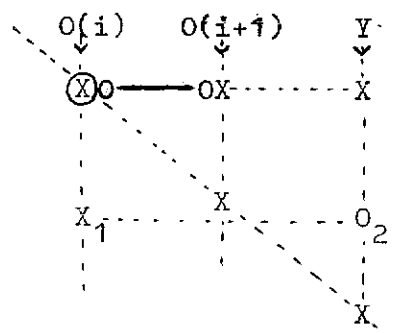



figure 3.4(a)

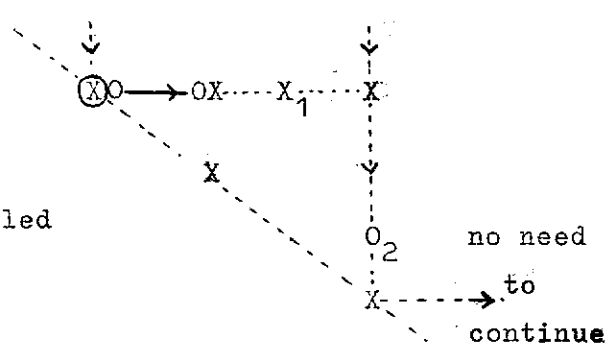
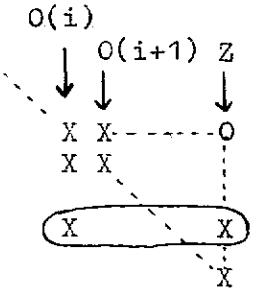
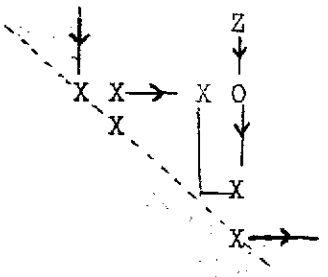



figure 3.4(b)

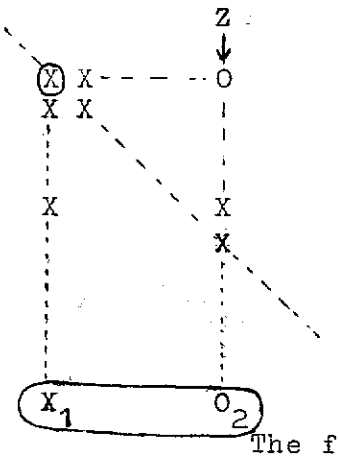


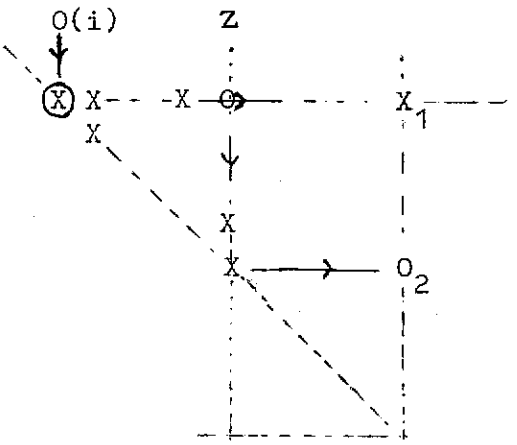




The search for zero headed columns with two non-zeros in the same row.

figure 3.5(a)

figure 3.5(b)




The fill-in of zero headed columns.

figure 3.6(a)

figure 3.6(b)

The full trace of the zero-headed column is necessary in calculating the fill-in caused by the Householder reduction of a symmetric matrix to tridiagonal form. The argument to justify this statement is as follows. Suppose that in figure 3.7 below it was unnecessary to continue along the second half of the dogleg.

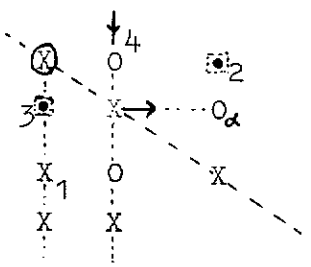


figure 3.7

Then, the fill-in of $\alpha$ would have to be taken account of in some other part of the calculation; either in tracing down the column of $\alpha$ or because it lies in a column and row with non-zeros in the pivot row and column. Certainly, if $\alpha$ is filled-in at all

during this stage of the reduction process either the element 2 or the element 3 has to be non-zero. Because of element 4 being zero, element 3 must be also and so element 2 must be non-zero. Hence, the fill-in of $\alpha$ is not included in the first pass of the algorithm or in any other zero-headed column. Thus $\alpha$ can only be filled-in on the second half of the dogleg of the column headed by the zero element 4, and so such scans are necessary ones in the above-mentioned reduction process.

It is stressed that this method of evaluating the fill-in caused by Householder reduction is not optimal. It has merely been described in order to help the reader follow the workings of the sparse matrix storage scheme as given in the program in the appendices.

Section 4. Suggestions for further investigation.

The work of the last two sections has indicated that virtually no advantage can be taken of the sparsity of the matrix if symmetric orthogonal reductions are used to reduce it to upper Hessenberg or tridiagonal form. It is only when the matrix is of a particularly suitable structure for the reduction process that it is worth considering such a procedure. An example of this would be if the matrix were reducible when symmetric permutations might be used to reduce it to the form of figure 4.1.

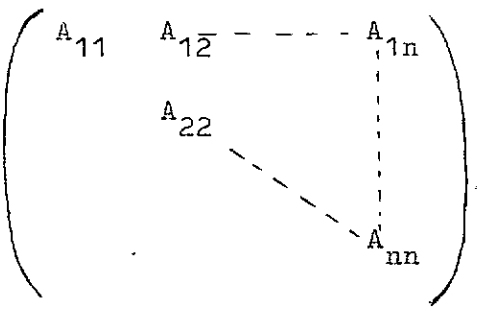



figure 4.1.

Such permutations and methods for finding them have been described in chapter three. The eigenvalues of the matrix of figure 4.1 are just those of the diagonal blocks, while the eigenvectors are partitioned as in the matrix of the figure and are zero in all partitions except one, in which partition the values are identical to the eigenvalues of the corresponding diagonal block.

If the matrix is of symmetric band form, then it may be reduced by means of a sequence of Givens rotations to triple diagonal form without the sparsity of the matrix being increased substantially in the intermediate stages (as was the case in tables 3.1 and 3.2). This method, an adaptation of an idea due to Rutishauser (1963), is described by Schwarz (1968) and involves using supplementary Givens rotations to chase non-zeros out of the matrix immediately after they have been created by previous rotations. It is particularly suitable for matrices with a small bandwidth although it is believed that some variant of this kind or a variant of Jacobi's (1846) method may be

possible for a more general type of sparse matrix. It may, of course, be possible to employ bandwidth minimisation techniques to first reduce the bandwidth of an arbitrary symmetric matrix before employing Schwarz's method to reduce it to tridiagonal form. Some techniques for such a minimisation are given by Alway and Martin (1965), Spillers (1965), Rosen (1968), Akyuz and Utku (1968), Cuthill and McKee (1969), Segethova (1970), and Arany, Smyth and Szoda (1971) and a review of several current techniques is given in Cuthill (1972). In order to see how effective bandwidth minimisation routines might be, symmetrised versions of the 54 x 54 matrix of Curtis and the 199 x 199 matrix of Willoughby were permuted using a routine at the National Physical Laboratory. A permutation was found to reduce the bandwidth of the 54 x 54 case to eleven while it was shown empirically that the minimum bandwidth was greater than nine. This would just about be a reasonable case on which to apply techniques akin to those described above. However, in the 199 x 199 case it has been shown that no permutation exists with a bandwidth of less than 52, while one has been found producing a bandwidth of 80 in place of the original bandwidth of 169. The author is indebted to Dr. D.W. Martin of NPL for these results. They indicate that it is dubious whether a general matrix will give a bandwidth reduction large enough to make the above method suitable although there could, of course, be a certain category of matrix for which this were feasible. An additional factor against using such bandwidth minimisation techniques (now out of favour for the solution of linear algebraic systems (Martin (personal communication))) is in the time spent in finding such a permutation. However, further investigation in this direction might be fruitful.

Because of the results of the preceeding sections, it is interesting to examine possibilities for obtaining the eigenvalues and eigenvectors of a general matrix without any prior reductions by orthogonal matrices. An obvious method which commends itself is that

of inverse iteration (see, for example, Wilkinson (1965)) where the basic operation is that of solving a set of linear equations thus allowing the sparsity techniques of chapters 2 to 5 to be utilised in the eigenvalue problem. This technique requires some prior knowledge of the eigenvalues or at least an estimate of their size and is particularly good when only a few eigenvectors are required. The method can also be used to obtain improved estimates of the eigenvalues. It would be useful to be able to obtain the decomposition of a matrix $(a - pI)$ from that of $A$ or $(A - p_1 I)$ (for some $p_1 \neq p$) in order to save having to perform a new decomposition for the inverse iteration for several eigenvectors of different eigenvalues, should that be required. In addition, it is sometimes advantageous when performing inverse iteration to alter the matrix by replacing $A - pI$ by $A - qI$ where $q$ is obtained from the iterations using $A - pI$ (Wilkinson (1966)) and is a better estimate of the eigenvalue nearest to $p$. Such an updating is termed a 'modification of the triangular factors' (Bennett (1965), Green (1968)). In these two papers, the modification of $A$ is restricted to the addition of a matrix of rank 1 and so is no good for the present purpose. The following suggestion is an attempt to remedy this. There is no limit in the following theory to what the rank of the change may be although it is assumed in what follows that the pivotal sequence is unchanged by the modification. It is assumed without loss of generality, that the matrix $A$ has been permuted so that the pivots are taken from the diagonal and

$$A = LU \qquad \ldots\ (4.1)$$

where

$U$ is unit triangular with elements $u_{ij}$, $i<j$.

$L$ is lower triangular with elements $l_{ij}$, $i \geq j$.

The essential part of this method lies in calculating the function

$$f_k(i,j) = \hat{a}_{ij}^{(k)} - a_{ij}^{(k)} \qquad \ldots\ (4.2)$$

where $\hat{a}_{ij}^{(k)}$ is the (i,j)th element of the modified matrix $\hat{A}$ at the k th stage of the LU decomposition process and $a_{ij}^{(k)}$ the corresponding element from A. A recurrence relation for the $f_k$ s of equation (4.2) is

$$f_{k+1}(i,j) - f_k(i,j) = \hat{u}_{kj}\,\hat{l}_{ik} - u_{kj}\,l_{ik} \qquad \ldots\ (4.3)$$

where $\hat{u}_{ij}$ and $\hat{l}_{ij}$ are elements in the upper and lower triangle of the LU decomposition of $\hat{A}$ respectively. Now,

$$\hat{u}_{kj} = g(u_{kj} + f_k(i,j)/a_{kk}^{(k)}) \qquad \ldots\ (4.4)$$

and

$$\hat{l}_{ik} = l_{ik} + f_k(i,k) \qquad \ldots\ (4.5)$$

where

$$g = (1 + f_k(k,k)/a_{kk}^{(k)})^{-1} \qquad \ldots\ (4.6)$$

and so, from equations (4.2), (4.3), (4.4) and (4.5),

$$f_{k+1}(i,j) - f_k(i,j) = 1/(l_{kk}-f_k(k,k))\{(\hat{l}_{ik}\hat{u}_{kj}f_k(i,k)+f_k(k,j).\hat{l}_{ik} - f_k(k,j)f_k(i,k) - f_k(k,k).\hat{u}_{kj}\hat{l}_{ik})\} \qquad \ldots\ (4.7)$$

It is seen that equation (4.7) gives a formula for updating the $f_k$ in terms of the newly created row and column in the LU decomposition of $\hat{A}$. Of course, if $\hat{a}_{kj}^{(k)}$ or $\hat{a}_{ik}^{(k)}$ equals zero, then $f_k(i,j)$ will not be updated. This is intuitively obvious and can be seen from equation (4.7) when it is observed that $\hat{a}_{ij}^{(k)} = 0 \Longrightarrow f_k(i,j) = 0$.

Thus, a possible method of obtaining the LU decomposition of $\hat{A}$, a modification of A, would be as follows. Associate a variable f(i,j) with every non-zero element of the LU decomposition of $\hat{A}$. The $f_k(i,j)$,(k=1,...n), and the elements of the LU decomposition of $\hat{A}$ are then calculated in the following manner.

Set $f_1(i,j) = \hat{a}_{ij}^{(1)} - a_{ij}^{(1)}$ where many of the $f_1(i,j)$ may be zero. For each k, k=1,2,..n-1, $\hat{u}_{kj}$ and $\hat{l}_{ik}$ are calculated from equations (4.5) and (4.4) and are used to calculate $f_{k+1}$ from equation (4.7). Naturally, the evaluation of $f_{k+1}$ from equation (4.7)

only involves looking down columns of $A^{(k)}$ with non-zeros of $\hat{A}^{(k)}$ in row k of the same column, and $f_{k+1}$ overwrites $f_k$ when it is calculated as do $\hat{l}$ and $\hat{u}$ overwrite the corresponding l and u. After these n-1 stages, setting $\hat{l}_{nn} = l_{nn} + f_n(n,n)$ completes the LU decomposition of A.

The storage requirements are not too severe at one more storage location per non-zero, but the number of arithmetic operations is very large since equation (4.7) requires at least 4 multiplications and 5 additions.

By slightly extending the storage requirements, however, this exceptionally bad defect can be overcome. For this improved method equation (4.3) is used instead of (4.7) and the old column of L and row of U for each stage are stored in temporary auxiliary storage. This means that only 2 multiplications and 2 additions are required for each update of $f_k(i,j)$. The rest of the algorithm is essentially identical to the previous one and a skeleton algol program for this update is given at the end of this section. Pivoting is assumed to be down the main diagonal and in the loops over rows or columns only non-zeros are accessed.

The use of this algorithm for the update in the eigenvalue problem is evident. The great saving by using such a procedure is that it is unnecessary to keep a copy of A once its LU decomposition is known. It is also comparable in operation count with having to recalculate A by remultiplying L and U, forming the modified A, and decomposing it from scratch.

It is often of interest in the eigenvalue problem to obtain a group of dominant eigenvalues and eigenvectors rather than a few isolated ones. A method for doing this, based on the power method, is developed by Clint and Jennings (1970, 1971) and is particularly suited to the solution of the eigenproblem for sparse systems.

Very often, some physical insight into the problem, or the known solution to a neighbouring problem, may yield some a priori estimate of the eigenvalues. However, if the above methods are to be used to solve a general problem, there must be first some method of obtaining approximate eigenvalues. In the unsymmetric case this is best done by first reducing the matrix to Hessenberg form by means of the more sparsity preserving Gaussian reductions and then using established methods (for example, Hyman (1957)) to calculate its eigenvalues. In the symmetric case, the comments in section 3 and the beginning of this section dissuade the use of orthogonal transformations to reduce the matrix to tridiagonal form and so the following procedure is recommended. Perform the LU decomposition of $A$ by means of row Gauss elimination with pivoting and obtain the principal minors of $A-pI$ from this reduction process (Martin and Wilkinson (1967) do this for symmetric band matrices). Since these minors have the Sturm sequence property, it is possible to quickly obtain an estimate of the eigenvalues by this method. It is not recommended to use this method for an accurate determination of the eigenvalues by means of the bisection method because of the time taken in the row Gauss eliminations for $A-pI$ with varying values of $p$. Similarly, such a method could be tedious for a determination of the complete eigensystem of $A$. However, a combination of a few row Gauss eliminations combined with inverse iteration may well provide a feasible method of tackling the symmetric eigenvalue problem in a way in which advantage can be taken of the original sparsity of the matrix.

The algorithm for modifying the triangular factors is given on the next page.

C Set f(i,j) equal to $\hat{a}_{ij}-a_{ij}$ for all i,j, where $\hat{A}$ is the modification to A. All variables and arrays are assumed to have been already defined.

```
for  k:=1  step 1  until  N-1  do
begin
      G:=   (1+f(k,k)/L(k,k))**-1 ;
     for i:=k+1 step 1 until  N  do
     begin
           AUXE(i):=L(i,k) ;
           L(i,k):=L(i,k) + f(i,k) ;
     end;
     for j:=k+1 step 1 until  N  do
     begin
           AUXXI(j):= U(k,j) ;
           U(k,j):= G * (U(k,j) - f(k,i)/AUXE(k)) ;
           for i:=k+1 step 1 until  N  do
           if AUXE(i) ≠ 0  then
           f(i,j):= f(i,j) + L(i,k)*U(k,j) - AUXE(i)* AUXXI(j)  ;
     end;
end;
     L(n,n):= L(n,n) + f(n,n) ;
```

C The vectors L and U now hold the lower and upper triangular portions of the triangular decomposition of $\hat{A}$ respectively.

| MATRIX / METHOD | 54 x 54 | 57 x 57 | 199 x 199 | RANDOM 50 x 50 | | | RANDOM 75 x 75 | | | RANDOM 100 x 100 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| **Total fill-in** GAUSS | 1825 | 1755 | 35792 | 315 | 1306 | 1862 | 2975 | 4218 | 4664 | 6805 | 8156 | 8486 |
| HOUSEHOLDER | 2455 | 2560 | 38077 | 703 | 1893 | 2099 | 4244 | 4949 | 4898 | 8642 | 9172 | 8786 |
| GIVENS | 2455 | 2560 | 38238 | 694 | 1893 | 2099 | 4218 | 4949 | 4898 | 8642 | 9172 | 8786 |
| **Max. No. non-zeros Held** GAUSS | 910 | 796 | 17812 | 282 | 655 | 1069 | 1408 | 2115 | 2690 | 3113 | 4096 | 5052 |
| HOUSEHOLDER | 2599 | 2635 | 38444 | 717 | 1935 | 2310 | 4168 | 5127 | 5404 | 8750 | 9549 | 9708 |
| GIVENS | 2488 | 2549 | 37024 | 664 | 1853 | 2278 | 4047 | 5077 | 5354 | 8611 | 9480 | 9682 |
| **Number of non-zeros in Hessenberg Form** GAUSS | 908 | 795 | 17811 | 281 | 653 | 1066 | 1404 | 2114 | 2689 | 3111 | 4092 | 4848 |
| HOUSEHOLDER | 1538 | 1600 | 20096 | 665 | 1239 | 1303 | 2700 | 2845 | 2923 | 4948 | 5107 | 5148 |
| GIVENS | 1538 | 1600 | 20268 | 655 | 1239 | 1303 | 2675 | 2845 | 2923 | 4948 | 5107 | 5148 |
| **Number of operations for Decomposition** GAUSS | 26611 | 18804 | 1410133 | 1624 | 13417 | 33566 | 38381 | 80905 | 154408 | 135149 | 270155 | 434653 |
| Squares | 1260 | 1296 | 19056 | 192 | 870 | 1138 | 1851 | 2528 | 2677 | 4186 | 4757 | 4826 |
| HOUSEHOLDER Multn's | 215281 | 222528 | 12495223 | 16494 | 130228 | 183793 | 407205 | 608800 | 659963 | 1320477 | 1558240 | 1592300 |
| GIVENS Sine-cos | 1208 | 1241 | 18848 | 162 | 825 | 1090 | 1783 | 2456 | 2604 | 4091 | 4659 | 4728 |
| GIVENS Multn's | 405654 | 421478 | 24615616 | 25302 | 241714 | 349598 | 774162 | 1177156 | 1281067 | 2563573 | 3042322 | 3115986 |
| % Density Hessenberg After Gauss. | 59.0 | 46.5 | 88.6 | 21.3 | 49.3 | 80.5 | 48.0 | 72.3 | 92.0 | 60.4 | 79.5 | 94.1 |

Table 2.1. A comparison of fill-in properties for the three methods of reduction to Hessenberg form. Results for random values are the average of five runs at each value.

| METHOD \ MATRIX | | 54 x 54 | 57 x 57 | 199 x 199 | RANDOM 50 x 50 DENSITY | | | RANDOM 75 x 75 DENSITY | | | RANDOM 100 x 100 DENSITY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| Total fill-in | GIVENS | 1125 | 1145 | 18378 | 239 | 729 | 964 | 1638 | 2289 | 2351 | 3993 | 4386 | 4290 |
| | HOUSEHOLDER | 1125 | 1145 | 18378 | 239 | 729 | 964 | 1638 | 2289 | 2351 | 3993 | 4386 | 4290 |
| Max. no. of non-zeros held. | GIVENS | 1100 | 1103 | 18163 | 261 | 696 | 1031 | 1547 | 2302 | 2546 | 3906 | 4487 | 4707 |
| | HOUSEHOLDER | 1141 | 1138 | 18346 | 264 | 722 | 1053 | 1574 | 2349 | 2586 | 3955 | 4544 | 4742 |

Table 3.1. A comparison of Givens and Householder reductions with regards to total fill-in and maximum number of non-zeros held. The pivots in each case are the same and are chosen to reduce the fill-in for Givens reduction. All results for random matrices are the average from five runs at each order and density.

| METHOD \ MATRIX | | 54 x 54 | 57 x 57 | 199 x 199 | RANDOM 50 x 50 DENSITY | | | RANDOM 75 x 75 DENSITY | | | RANDOM 100 x 100 DENSITY | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CURTIS 1 | WILLOUGHBY 1 | WILLOUGHBY 2 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 | 0.03 | 0.05 | 0.10 |
| Total fill-in | GIVENS | 1125 | 1145 | 18378 | 239 | 729 | 964 | 1638 | 2289 | 2351 | 3993 | 4386 | 4290 |
| | HOUSEHOLDER | 1125 | 1145 | 18378 | 240 | 729 | 964 | 1638 | 2289 | 2351 | 3993 | 4386 | 4290 |
| Max. No. of non-zeros held | GIVENS | 1100 | 1103 | 18163 | 261 | 696 | 1031 | 1547 | 2302 | 2546 | 3906 | 4487 | 4700 |
| | HOUSEHOLDER | 1141 | 1138 | 18346 | 264 | 722 | 1053 | 1574 | 2349 | 2586 | 3955 | 4544 | 4742 |
| Number of Operations | | 1188 | 1191 | 18522 | 241 | 743 | 1034 | 1655 | 2352 | 2546 | 4046 | 4530 | 4676 |
| | GIVENS | 152778 | 151726 | 9488830 | 15171 | 76284 | 127660 | 252857 | 433762 | 491829 | 971240 | 1157624 | 1220004 |
| | | 1240 | 1246 | 18719 | 275 | 786 | 1082 | 1720 | 2424 | 2619 | 4139 | 4628 | 4774 |
| | HOUSEHOLDER | 79963 | 79703 | 5797923 | 8379 | 40411 | 66738 | 131523 | 223691 | 252502 | 497535 | 591619 | 622088 |

Table 3.2. A comparison of Givens' and Householder's methods for symmetric matrices. The pivots are selected independently in each case. The results for random matrices are the average from five runs at each order and density.

## REFERENCES


Akyuz F.A. and Utku S. (1968) An automatic relabelling scheme for bandwidth minimisation of stiffness matrices. Am. Inst. Aeronaut. Astronaut. J. 6, pp.728-730.

Alway G.G. and Martin D.W. (1965) An algorithm for reducing the bandwidth of a matrix of symmetric configuration. Computer J. 8, pp.264-272.

Arany I., Smyth W.F., and Szoda L. (1971) An improved method for reducing the bandwidth of sparse symmetric matrices. Proc. IFIP Conf., Ljubljana. Numerical Mathematics, Booklet TA-1, pp.6-10. North-Holland, Amsterdam.

Baker J.J. (1962) A note on multiplying Boolean matrices. (Pracniques). Comm. ACM. 5, p.102.

Barkley R.W., and Motard R.R. (1972) Decomposition of nets. To appear in proc. of NATO conf. on decomposition as a tool for solving large scale problems. Cambridge, England.

Bellman R. (1957) Dynamic Programming. Princeton Univ. Press., N.J.

Bennett J.M. (1965) Triangular factors of modified matrices. Num. Math. 7, pp.217-221.

Berge C. (1962) The Theory of graphs. Methuen, London.

Bertelè U. and Brioschi F. (1971) A note on a paper by Spillers and Hickerson. Quart. Appl. Math. 24, pp.311-313.

Branin F.H. (1959) The relation between Kron's method and the classical methods of network analysis. IRE WESCON Convention Record, Part 2. pp.1-29

Brayton R.K., Gustavson F.G., and Willoughby R.A. (1970) Some results on sparse matrices. Math. Comp. 24, pp.937-954.

Bree D. (1964) Some remarks on the application of graph theory to the solution of sparse systems of linear equations. Publication of the Bonneville Power Administration.

Busacker R.G. and Saaty T.L. (1965) Finite graphs and and networks. McGraw-Hill, New York.

Businger P., and Golub G.H. (1965) Handbook Series Linear Algebra. Least squares solutions by Householder transformations. Num. Math. 7, pp.269-276.

Carré B.A. (1966) The partitioning of network equations for block iteration. Computer J. 9, pp.84-96.

Cheyney E.W. (1966) Introduction to approximation theory. McGraw-Hill, New York.

Clint M., and Jennings A. (1970) The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration. Computer J. 13, pp.76-80.

Clint M., and Jennings A. (1971) A simultaneous iteration method for the unsymmetric eigenvalue problem. J. IMA. 8, pp.111-121.

Comstock D.R. (1964) A note on multiplying Boolean matrices 2. (Pracniques). Comm. ACM. 7, p.13

Curtis A.R., and Reid J.K. (1971a) On the automatic scaling of matrices for Gaussian elimination. AERE Harwell Report. TP 444. To appear in J. IMA.

Curtis A.R., and Reid J.K. (1971b) Fortram subroutines for the solution of sparse sets of linear equations. AERE report. R.6844. HMSO.

Curtis A.R., and Reid J.K. (1971c) The solution of large sparse unsymmetric systems of linear equations. Proc. IFIP. Conf., Ljubljana. Numerical mathematics, Booklet TA-1. pp.1-5. North-Holland, Amsterdam.

Curtis A.R., and Reid J.K. (1971d) The solution of large sparse unsymmetric systems of linear equations. J. IMA. 8, pp.344-353.

Cuthill E. (1972) Several strategies for reducing the bandwidth of matrices. In Rose and Willoughby (1972). pp.157-166.

Cuthill E., and McKee J. (1969) Reducing the bandwidth of sparse symmetric matrices. Proc. 24th National Conf. ACM. pp.157-172. Brandon Systems Press, N.J.

de Buchet J. (1971) How to take into account the low density of matrices to design a mathematical programming package. Relevant effects on optimisation and inversion algorithms. In Reid (1971a). pp.211-217.

de Villiers E.V.D.S. (1971) Scatter storage techniques for sparse matrices. M.Sc. thesis, Univ. of Newcastle.

Duff I.S. (1970) Network analysis and graph theory. Dissert. for Dipl. Adv. Math., Oxford.

Dulmage A.L., and Mendelsohn N.S. (1959) A structure theory of bipartite graphs of finite exterior dimension. Trans. Roy. Soc. Canada. 53, Sect.3, pp.1-13.

Dulmage A.L., and Mendelsohn N.S. (1962) On the inversion of sparse matrices. Math. Comp. 16, pp.494-496.

Dulmage A.L., and Mendelsohn N.S. (1963a) Two algorithms for bipartite graphs. J. SIAM. 11, pp.183-194.

Dulmage A.L., and Mendelsohn N.S. (1963b). Remarks on solutions of the optimal assignment problem. J. SIAM. 11, pp.1103-1109.

Dulmage A.L., and Mendelsohn N.S. (1967) Graphs and Matrices. Ch. 6 of Harary (1967b) pp.167-227.

Erdelsky P.J. (1968) A general theorem on dominant-diagonal matrices. Linear algebra and its applications. 1, pp.203-209.

Erdös P. (1967) Applications of probabilistic methods to graph theory. In Harary (1967a). pp.60-64.

Erisman A.M. (1972) Sparse matrix approach to the frequency domain analysis of linear passive electrical networks. In Rose and Willoughby (1972). pp.31-40.

Eufinger J. (1970) Eine Untersuchung zur Auflösung magerer Gleichungssysteme. J. für reine u. angew. Mathematik. 245, pp.208-220.

Eufinger J. (1971) Operations on directed graphs. J. für reine u. angew. Mathematik. 247, pp.146-154.

Evans D.J. (1972) A new iterative procedure for the solution of sparse systems of linear difference equations. In Rose and Willoughby (1972). pp.89-100.

Feller W. (1968) An introduction to probability theory and its applications.(3rd edition). Wiley, New York.

Fiedler M., and Pták V. (1962) On matrices with non-positive off-diagonal elements and positive principal minors. Czech. Math. J. 12, pp.382-400.

Ford L.R., and Fulkerson D.R. (1957) A simple algorithm for finding maximal network flows and an applicaton to the Hitchcock problem. Canad. J. Math. 9, pp.210-218.

Ford L.R., and Fulkerson D.R. (1962) Flows in networks. Princeton Univ. Press, N.J.

Fox L., and Mayers D.J. (1968) Computing Methods for Scientists and Engineers. Oxford Univ. Press.

Francis J.G.F. (1961, 1962) The QR transformation - a unitary analogue to the LR transformation. Parts 1 and 2. Computer J. 4, pp.265-271 and 332-345.

Fulkerson D.R., and Wolfe P. (1962) An algorithm for scaling matrices. SIAM Rev. 4, pp.142-146.

George J.A. (1972) Block elimination on finite element systems of equations. In Rose and Willoughby (1972). pp.101-114.

Gill P.E., and Murray W. (1970) A numerically stable form of the simplex algorithm. Tech. Report No. Math. 87. NPL, Teddington. (To appear in J. Lin. Algebra Appl.)

Givens W. (1958) Computation of plane unitary rotations transforming a general matrix to triangular form. J. SIAM. 6, pp.26-50.

Golub G. (1965) Numerical methods for solving linear least squares problems. Num. Math. 7, pp.206-216.

Green D.R. (1968) Triangular factors of modified matrices. Algorithm 319. Comm. ACM. 11, p.12.

Griesmer J.H., and Jenks R.D. (1971) A set of SCATCHPAD examples. Publication of Thomas J. Watson Research Center, Yorktown Heights, N.Y.

Gustavson F.G. (1972) Some basic techniques for solving sparse systems of linear equations. In Rose and Willoughby (1972). pp.41-52.

Gustavson F.G., Liniger W.M., and Willoughby R.A. (1970) Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations. J. ACM. 17, pp.87-109.

Hall M. (1956) An algorithm for distinct representatives. Am. Math. Monthly. 63, pp.716-717.

Hall P. (1935) On representatives of subsets.
J. London Math. Soc. 10, pp.26-30.

Harary F. (1959a) Graph theoretic methods in the management sciences. Management Sci. 5, pp.387-403.

Harary F. (1959b) A graph theoretic method for the complete reduction of a matrix with a view toward finding its eigenvalues. J. Math. Phys. 38, pp.104-111.

Harary F. (1960) On the consistency of precedence matrices.
J. ACM. 7, pp.255-259.

Harary F. (1962a) A graph theoretic approach to matrix inversion by partitioning. Num. Math. 4, pp.128-135.

Harary F. (1962b) The determinant of the adjacency matrix of a graph. SIAM Rev. 4, pp.202-210.

Harary F. (Ed) (1967a) A seminar on graph theory.
Holt, Rinehart and Winston, London.

Harary F. (Ed) (1967b) Graph theory and theoretical physics.
Academic Press, New York.

Harary F. (1969) Graph Theory. Addison-Wesley, Mass.

Harary F. (1971) Sparse matrices and graph theory.
In Reid (1971a). pp.139-150.

Harary F. (1972) Sparse digraphs. Classification and algorithms.
Ann-Arbor publication.

Harary, Norman, and Cartwright (1965) Structure of models.
An introduction to the theory of directed graphs.
Wiley, New York.

Harrison B.K. (1963) A discussion of some mathematical techniques used in Kron's method of tearing.
J. SIAM. 11, pp.258-280.

Hatchel G.D. (1972) Vector and matrix variability type in sparse matrix algorithms. In Rose and Willoughby (1972). pp.53-64.

Hatchel G.D., Brayton R.K., and Gustavson F.G. (1971) The sparse tableau approach to network analysis and design.
IEEE Trans. on circuit theory. 18, pp.101-113.

Householder A.S. (1958) Unitary triangularisation of a nonsymmetric matrix. J. ACM. 5, pp.339-342.

Hamming R.W. (1971) Introduction to applied numerical analysis.
McGraw-Hill, New York.

Heap B.R. (1966) Random Matrices and Graphs.
Num Math. 8, pp.114-122.

Hellerman E., and Rarick D.C. (1972a) The partitioned preassigned pivot procedure ($P^4$).
In Rose and Willoughby (1972). pp.67-76.

Hellerman E., and Rarick D.C. (1972b) Reinversion with the preassigned pivot procedure.
To appear in Math. Prog. J.

Hershkowitz M., and Noble S.B. (1965) Finding the inverse and connections of a type of large sparse matrix.
Nav. Res. Logist. Quart. 12, pp.119-132.

Hsieh H.Y., and Ghausi M.S. (1971a) On sparse matrices and optimal pivoting algorithms.
IBM Tech. Report TR 22.1249.

Hsieh H.Y., and Ghausi M.S. (1971b) Probabilistic approach to optimal pivoting and prediction of fill-in for random sparse matrices.
IBM Tech. Report TR 22.1248.

Hyman M.A. (1957) Eigenvalues and eigenvectors of general matrices.
Proc. 12th National Conf. ACM, Houston, Texas.
Brandon Systems Press, N.J.

Jacobi C.G.J. (1846) Über ein leichtes Verfahren die in der Theorie der Sächulärstörungen vorkommenden Gleichungen numerisch aufzulösen. Crelle's J. 30, pp.51-94

Jaeger A., and Wenke K. (1969) Lineare Wirtschaftsalgebra.
Teubner, Stuttgart.

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

Jimenez A.J. (1969) Computer handling of sparse matrices.
IBM Report TR 00.1373.

Khabasa I.M. (1963) Iterative least squares method suitable for solving large sparse matrices.
Computer J. 6, pp.202-206.

Knuth D.E. (1968) The art of computer programming. Vol.1
Fundamental algorithms. Addison-Wesley, Mass.

König D. (1950) Theorie der endlichen und unendlichen graphen. Chelsea, New York.

Kron G. (1953) A set of principles to interconnect the solutions of physical systems.
Appl. Phys. 24, pp.965-980.

Kron G. (1954) A method of solving very large physical systems in easy stages.
Proc. IRE. 42, pp.680-686.

Kron G. (1963) Diakoptics. Macdonald, London.

Kuhn H.W. (1955) The Hungarian method for solving the assignment problem.
Nav. Res. Logis. Quart. 2, pp.83-97.

Larsen L.J. (1962) A modified inversion procedure for product form of the inverse linear programming codes.
Comm. ACM. 5, pp.382-383.

Lo Dato V.A. (1970) The permutation of a certain class of matrices.
Computer J. 13, pp.405-410.

Marcus M., and Minc H. (1965) Permanents.
Am. Math. Monthly. 72, pp.577-591.

Martin R.S., and Wilkinson J.H. (1967) Handbook Series Linear Algebra. Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices.
Num. Math. 9, pp.279-301.

Martin R.S., and Wilkinson J.H. (1968) Handbook Series Linear Algebra. Reduction of the symmetric eigenproblem $Ax=\lambda Bx$ and related problems to standard form.
Num. Math. 11, pp.99-110.

Markowitz H.M. (1957) The elimination form of the inverse and its application to linear programming.
Management Sci. 3, pp.255-269.

Mason S.J. (1953) Feedback Theory - some properties of signed flow graphs.
Proc. IRE. 41, pp.1144-1156.

Maurer W.D. (1968) An improved hash code for scatter storage.
Comm. ACM. 11, pp.35-38.

Morris R. (1968) Scatter storage technique.
Comm. ACM. 11, pp.38-43.

Munkres J. (1957) Algorithms for the assignment and transportation problems.
J. SIAM. 5, pp.32-38.

McNamee J.M. (1971) A sparse matrix package (Part 1). Algorithm 408.
Comm. ACM. 14, pp.265-273.

Nathan A., and Even R.K. (1967) The inversion of sparse matrices by a strategy derived from their graphs.
Computer J. 10, pp.190-194.

Nuding E., and Kahlert-Warmbold I. (1971) A computer-orientated representation of matrices.
Computing 6, pp.1-8.

Ogbuobiri E.C. (1970) Dynamic storage and retrieval in sparsity programming.
IEEE Trans. on PAS. 89, pp.150-155.

Ogilvie J.C. (1968) The distribution of number and size of connected components in random graphs of medium size.
Proc. IFIP Conf., Edinburgh. Applications 3, Booklet H. pp.89-92.

Orchard-Hays W. (1968) Advanced linear programming computing techniques. McGraw-Hill, New York.

Oystein O. (1963) Graphs and their uses. Random House Inc., New York.

Palásti I. (1966) On the strong connectedness of directed random graphs.
Stud. Sci. Math. Hungarica. 1, pp.205-214.

Palásti I. (1970) On some structured properties of given types of random graphs.
Magyar Tud. Akad. Mat. Oszt. Közl. 19, pp.33-72.

Parlett B. (1964) Languerre's method applied to the matrix eigenvalue problem.
Math. Comp. 18, pp.464-485.

Parter S. (1960) On the eigenvalues and eigenvectors of a class of matrices.
J. SIAM. 8, pp.376-388.

Parter S. (1961) The use of linear graphs in Gauss elimination. SIAM Rev. 3, pp.119-130.

Paton K. (1971) An algorithm for the blocks and cutnodes of a graph. Comm. ACM. 14, pp.468-475.

Phillips W. (1970) The storage and inversion of large sparse matrices. Publication of ICI, Wilmslow.

Ponstein J. (1966) Self avoiding paths and the adjacency graph of a matrix. J. SIAM. 14, pp.600-609.

Powell M.J.D., and Reid J.K. (1968) On applying Householder transformations to linear least squares problems. Harwell report TP 322.

Purdom P. (1970) A transitive closure algorithm. BIT. 10, pp.76-94.

Read R. (Ed) (1971) Graph theory and computing. Academic Press, New York.

Reid J.K. (1967) A note on the least squares solution of a band system of linear equations by Householder reductions. Computer J. 10, pp.188-189.

Reid J.K. (Ed) (1971a) Large sparse sets of linear equations. Proc. of Conf. at Oxford. Academic Press, London.

Reid J.K. (1971b) On the method of conjugate gradients for the solution of large sparse systems of equations. In Reid (1971a). pp.231-254.

Reid J.K. (1971c) A note on the stability of Gaussian elimination. J. IMA. 8, pp.374-375.

Reid J.K. (1972a) Sparse matrices and decomposition with applications to the solution of algebraic and differential equations. To appear in proc. NATO Conf. on decomposition as a tool for solving large scale problems, Cambridge, England.

Reid J.K. (1972b) Least squares solution of sparse systems of non-linear equations by a modified Marquadt algorithm. To appear in proc. NATO Conf. on decomposition as a tool for solving large scale problems, Cambridge, England.

Rheinboldt W.C., and Meztenyi C.K. (1972) GRAAL - A graph algorithmic language. In Rose and Willoughby (1972). pp.167-176.

Rose D.J. (1970) Triangulated graphs and the elimination process. J. Math. Anal. Appl. 3, pp.597-609.

Rose D.J. (1971) A graph-theoretic study of the numerical solution of sparse positive definite systems of equations. In Graph theory and computing, **Read** (1971).

Rose D.J., and Bunch J.R. (1972) The role of partitioning in the numerical solution of sparse systems. In Rose and Willoughby (1972). pp.177-187.

Rose D.J., and Willoughby R.A. (Ed) (1972) Sparse matrices and their applications. Proc. of Conf. at IBM research center, N.Y. Plenum Press, New York.

Rosen R. (1968) Matrix bandwidth minimisation. Proc. 23rd National Conf. ACM. pp.585-595. Brandon Systems Press, N.J.

Rosenblatt D. (1957) On the graphs and asymptotic forms of finite Boolean relation matrices and stochastic matrices. Nav. Res. Logis. Quart. 4, pp.151-167.

Rutishauser H. (1963) On Jacobi rotation patterns. Proc. of symposia in applied mathematics. Vol.15. Experimental arithmetic, high speed computing and mathematics. pp.219-239.

Sargent R.W.H., and Westerberg A.W. (1964) "Speed-up" in chemical engineering design. Trans. Inst. Chem. Engnrs. 42, pp.190-197.

Sato N., and Tinney W.F. (1963) Techniques for exploiting the sparsity of the network admittance matrix. IEEE Trans. on PAS. 82, pp.944-949.

Saunders M.A. (1972) Large-scale linear programming using the Cholesky factorisation. Report no. STAN-CS-72-252, Comput. Sci. Dept. Stanford Univ.

Schwarz H.R. (1968) Tridiagonalisation of a symmetric band matrix. Num. Math. 12, pp.231-241.

Segethova J. (1970) Elimination procedures for sparse symmetric linear systems of a special structure. Tech. report 70-121. NGL-21-002-008. Univ. of Maryland, Computer Sci. Center.

Simonnard M.(1962) Programmation Linéaire. (Transl. by Jewell W.S. Prentice-Hall, N.J. 1966). Dunod, Paris.

Smith D.M. (1969) Data logistics for matrix inversion. In Willoughby (1969). pp.127-137.

Spillers W.R. (1965) On diakoptics : Tearing an arbitrary system. Quart. Appl. Math. 23, pp.188-190.

Spillers W.R., and Hickerson N. (1968) Optimal elimination for sparse symmetric systems as a graph problem. Quart. Appl. Math. 26, pp.425-432.

Steward D.V. (1962) On an approach to techniques for the analysis of the structure of large systems of equations. SIAM Rev. 4, pp.321-342.

Steward D.V. (1965) Partitioning and tearing systems of equations. J. SIAM. Num. Anal. 2, pp.345-365.

Steward D.V. (1972) Partitioning and tearing of systems : methods and applications. Ph.D. Thesis, Univ. of Wisconsin. To appear.

Tewarson R.P. (1966) On the product form of inverses of sparse matrices. SIAM Rev. 8, pp.336-342.

Tewarson R.P. (1967a) On the product form of inverses of sparse matrices and graph theory. SIAM Rev. 9, pp.91-99.

Tewarson R.P. (1967b) Solution of a system of simultaneous linear equations with a sparse coefficient matrix by elimination methods. BIT. 7, pp.226-239.

Tewarson R.P. (1967c) Row column permutation of sparse matrices. Computer J. 10, pp.300-305.

Tewarson R.P. (1968) On the orthonomalisation of sparse vectors. Computing. 3, pp.268-279.

Tewarson R.P. (1970a) On the transformation of symmetric sparse matrices to the triple diagonal form. Int. J. Comp. Math. 2, pp.247-258.

Tewarson R.P. (1970b) On the reduction of a sparse matrix to Hessenberg form. Int. J. Comp. Math. 2, pp.283-295.

Tewarson R.P. (1970c) Computations with sparse matrices.
SIAM Rev. 12, pp.527-543.

Tewarson R.P. (1971) Sorting and ordering sparse linear systems.
In Reid (1971a). pp.151-168.

Thorelli L.E. (1966) An algorithm for computing all paths in a graph.
BIT. 6, pp.347-349.

Tiernan J.C. (1970) An efficient search algorithm to find the elementary circuits of a graph.
Comm. ACM. 13, pp.722-726.

Tinney W.F. (1969a) Comments on using sparsity techniques for power system problems.
In Willoughby (1969). pp.25-34.

Tinney W.F. (1969b) Some examples of sparse matrix methods for power network problems.
Publication of the Bonneville Power Administration.

Tinney W.F., and Ogbuobiri E.C. (1970) Sparsity techniques : Theory and Practice.
Publication of the Bonneville Power Administration.

Varga R.S. (1962) Matrix Iterative Analysis. Prentice-Hall, N.J.

Walsh J. (Ed) (1966) Numerical Analysis - An introduction.
Academic Press, London.

Warshall S. (1962) A theorem on Boolean matrices.
J. ACM. 9, pp.11-12.

Weil R., and Kettler P. (1969) An algorithm to provide structure for decomposition.
In Willoughby (1969). pp.11-24.

Wenke K. (1964) Praktische Anwendung linearer Wirtschaftsmodelle.
Unternehmensforschung. 8, pp.34-46.

Willoughby R.A. (Ed) (1969) Proceedings of the symposium on sparse matrices and their applications, Yorktown Heights, N.Y.
IBM Report RA 1 (No.11707).

Willoughby R.A. (1970) A survey of sparse matrix technology.
Lecture at Conf. on Computer Oriented Analysis of Shell Structures, Lockheed, Palo Alto Res. Lab.

Willoughby R.A. (1971a) Some pivot considerations in direct methods for sparse matrices.
IEEE Int. Conf. on Systems, Networks, and Computers.
Oaxtepec, Mor., Mexico. pp.387-390.

Willoughby R.A. (1971b) Sparse matrix algorithms and their relation to problem classes and computer architecture.
In Reid (1971a). pp.255-277.

Willoughby R.A. (1971c) Talk at Computer Sci. Dept., Univ. of Waterloo.

Willoughby R.A. (1972) A matrix reducibility algorithm.
To appear in Math. Comp.

Wilkinson J.H. (1965) The Algebraic Eigenvalue Problem.
Oxford Univ. Press.

Wilkinson J.H. (1966) Calculation of eigensystems of matrices.
Chapter 3 in Walsh (1966). pp.27-61.

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

Winograd S. (1968) A new algorithm for inner product.
IEEE Trans. on Computers. 17, pp.693-694.

Yaspan A. (1966) On finding a maximal assignment.
Operations Research. 14, pp.646-651.

Zollenkopf K. (1971) Bi-factorisation - Basic computational algorithm and programming techniques.
In Reid (1971a). pp.75-96.