SPARSE MATRIX TECHNOLOGY

electronic edition

Sergio Pissanetzky

Sparse Matrix Technology

electronic edition

Sergio Pissanetzky

Professionally typeset by LaTeX. This work is in compliance with the mathematical typesetting conventions established by the International Organization for Standardization (ISO).

Dr. Pissanetzky retired after a rewarding career as an Entrepreneur, Professor, Research Scientist and Consultant. He was the founder of Magnus Software Corporation, where he focused on development of specialized applications for the Magnetic Resonance Imaging (MRI) and the High Energy Particle Accelerator industries. He has served as Member of the International Editorial Board of the "International Journal for Computation in Electrical and Electronic Engineering", as a Member of the International Advisory Committee of the International Journal "Métodos Numéricos para Cálculo y Diseño en Ingeniería", and as a member of the International Committee for Nuclear Resonance Spectroscopy, Tokyo, Japan. Dr. Pissanetzky has held professorships in Physics at Texas A&M University and the Universities of Buenos Aires, Córdoba and Cuyo, Argentina. He has also held positions as a Research Scientist with the Houston Advanced Research Center, as Chairman of the Computer Center of the Atomic Energy Commission, San Carlos de Bariloche, Argentina, and as a Scientific Consultant at Brookhaven National Laboratory. Dr. Pissanetzky holds several US and European patents and is the author of two books and numerous peer reviewed technical papers. Dr. Pissanetzky earned his Ph.D. in Physics at the Balseiro Institute, University of Cuyo, in 1965. Dr. Pissanetzky has 35 years of teaching experience and 30 years of programming experience in languages such as Fortran, Basic, C and C++. Dr. Pissanetzky now lives in a quite suburban neighborhood in Texas.

Website: http://www.SciControls.com

ISBN 978-0-9762775-3-8

Contents

Preface to the Electronic Edition

This is an electronic edition of the classic book Sparse Matrix Technology by Sergio Pissanetzky, originally published in English by Academic Press, London, in 1984, and later translated into Russian and published by MIR, Moscow, in 1988. The electronic edition has been typed from the original, with only minor changes of format where dictated by electronics.

Preface

As computers grow in power and speed, matrices grow in size. In 1968, practical production calculations with linear algebraic systems of order 5,000 were commonplace, while a "large" system was one of order 10,000 or more.[a]

In 1978, an over determined problem with 2.5 million equations in 400,000 unknowns was reported;[b] in 1981, the magnitude of the same problem had grown: it had 6,000,000 equations, still in 400,000 unknowns.[c] The matrix of coefficients had 2.4×10^{12} entries, most of which were zero: it was a *sparse* matrix. A similar trend toward increasing size is observed in eigenvalue calculations, where a "large" matrix is one of order 4,900 or 12,000.[d] Will matrix problems continue to grow even further? Will our ability to solve them increase at a sufficiently high rate?

But this is only one side of the question. The other side concerns the microcomputer explosion. Microcomputers now have about the same power as large computers had two decades ago. Are users constrained to solving matrix problems of the same size as those of twenty years ago?

The owner of a microcomputer may not care too much about the cost of computation; the main difficulty is storage. On a large machine, the cost of solving a matrix problem increases rapidly if the size of the problem does, because both storage and labor grow. The overall cost becomes a primary consideration. How can such cost be minimized for a given problem and installation?

Answers to these and other related questions are given in this book for the following classes of matrix problems: direct solution of sparse linear algebraic equations, solution of sparse standard and generalized eigenvalue problems, and sparse matrix algebra. Methods are described which range from very simple yet surprisingly effective ideas to highly sophisticated algorithms. Sparse matrix technology is now a well established discipline, which was defined as "the art of handling sparse matrices".[e] It is composed of a beautiful blend of theoretical developments, numerical experience and practical considerations. It is not only an important computational tool in a broad spectrum

[a]In the Preface, the pertinent references are given as footnotes, because this enhances clarity. The full list of references is given at the end of the book. Tinney, 1969,[237] p.28; Willoughby, 1971,[250] p.271.

[b]Kolata, 1978.[144]

[c]Golub and Plemmons, 1981, [106] p.3.

[d] Cullum and Willoughby, 1981,[42] p. 329; Parlett, 1980,[175] p. XIII.

[e]Harary, 1971.[124]

of computational areas,[f] but also is in itself a valuable contribution to the general development of computer software. The new ideas developed during the last fifteen years were used to devise nearly optimum algorithms for a variety of matrix problems. Research in the field is currently very active and the spectrum of applications broadens continuously. Sparse matrix technology is here and will stay.

The concept expressing the nature of our concern is contained in the title of the book. Technology is applied science, the science or study of the practical or industrial arts.[g] The phrase "sparse matrix technology" was an everyday saying in the early nineteen seventies at the IBM T. J. Watson Research Center.[h] Nowadays it seems to be in desuetude. The material for the book was selected from the several Symposia and Congresses on large matrix problems regularly held since 1968.[i] Major sources of inspiration were: an advanced course with four review articles,[j] excellent survey articles[k] and books,[l] a collection of papers,[m] and many publications which are cited where pertinent. Several basic ideas can be found in the literature published before 1973.[n] No attempt is made, however, to cover such an important amount of material. Rather, the fundamental methods and procedures are introduced and described in detail, the discussion reaching the point where the reader can understand the specialized literature on each subject. A unified treatment is provided whenever possible, although, like any field of human knowledge which grows fast, sparse matrix technology has grown unevenly. Some areas are well developed, while other areas lack further research. We have not included proofs of all the theorems, except when they are closely related to practical techniques which are used subsequently. The concepts and methods are introduced at an elementary level, in many cases with the help of simple examples. Many fundamental algorithms are described and carefully discussed. Ready-to-use very efficient and professional algorithms are given in Fortran. The reader is assumed to be familiar with this popular language. The algorithms, however, are explained so clearly that even a person with a limited knowledge of Fortran can understand them and eventually translate them into other languages. Linear algebra and graph theory are used extensively in the book. No particular acquaintance with these subjects is necessary because all definitions and properties are introduced from the beginning, although some preparation

[f]Rose and Willoughby, 1972.[198]

[g]Webster's Dictionary, second edition, 1957.

[h]Willoughby, 1971;[250] Rose and Willoughby, 1972,[198] Preface; Willoughby, 1972;[251] Hachtel, 1976,[117] p. 349.

[i]Willoughby, 1969;[249] Reid, 1971a;[187] Rose and Willoughby, 1972;[198] Bunch and Rose, 1976;[28] Duff and Stewart, 1979;[68] Duff, 1981b.[61] The Proceedings of the Symposium held at Fairfield Glade, Tennessee, in 1982, will be published as a special issue of the SIAM Journal on Scientific and Statistical Computing, and possibly other SIAM journals, to appear in 1983. The Software Catalog prepared in conjunction with the Symposium is available (Heath, 1982.[126])

[j]Barker, 1977.[10]

[k]Duff, 1977,[55] 1982.[62]

[l]Wilkinson, 1965;[247] Parlett, 1980;[175] George and Liu, 1981.[97]

[m]Björck *et al.* 1981 [16]

[n]Brayton *et al.* 1970; [19] Willoughby, 1972;[251] Tewarson, 1973.[235]

may be helpful. An extensive bibliography and a survey of the relevant literature are included in many sections. The book fills the gap between books on the design of computer algorithms and specialized literature on sparse matrix techniques, on the one side, and user needs and application oriented requirements on the other.

The purpose of the book is to bring sparse matrix technology within reach of engineers, programmers, analysts, teachers and students. This book will be found helpful by everyone who wishes to develop his own sparse matrix software, or who is using it and wishes to understand better how it operates, or who is planning to acquire a sparse matrix package and wishes to improve his understanding of the subject. Teachers who need an elementary presentation of sparse matrix methods and ideas and many examples of application at a professional level, will find such material in this book.

Chapter 1 covers all fundamental material such as storage schemes, basic definitions and computational techniques needed for sparse matrix technology. It is very convenient to read at least Sections 1 to 9 and Section 12 of Chapter 1 first. The first reading may, however, be superficial. The reader will feel motivated to examine this material in more detail while reading other chapters of the book, where numerous references to sections of Chapter 1 are found.

Chapters 2 to 5 deal with the solution of linear algebraic equations. They are not independent. The material in Chapter 2 is rather elementary, but its form of presentation serves as an introduction for Chapters 4 and 5, which contain the important material. Chapter 3 deals with numerical errors in the case where the linear system is sparse, and also serves as an introduction to Chapters 4 and 5. This material is not standard in the literature. Sparse matrix methods and algorithms for the direct solution of linear equations are presented in Chapters 4 and 5. Chapter 4 deals with symmetric matrices, and Chapter 5 with general matrices.

The calculation of eigenvalues and eigenvectors of a sparse matrix, or of a pair of sparse matrices in the case of a generalized eigenvalue problem, is discussed in Chapter 6. Chapter 6 can be read independently, except that some references are made to material in Chapters 1 and 7.

Chapters 7, 8 and 9 deal with sparse matrices stored in row-wise format. Algorithms for algebraic operations, triangular factorization and back substitution are explicitly given in Fortran and carefully discussed in Chapter 7. The material in Chapter 1 is a prerequisite, particularly Sections 8, 9 and 10 and 12 to 17. In addition, Chapter 2 is a prerequisite for Sections 23 to 28 of Chapter 7. Chapter 8 covers the sparse matrix techniques associated with mesh problems, in particular with the finite element method, and in Chapter 9 we present some general purpose Fortran algorithms.

Sparse matrix technology has been applied to almost every area where matrices are employed. Anyone interested in a particular application may find it helpful to read the literature where the application is described in detail, in addition to the relevant chapters of this book. A list of bibliographical references sorted by application was published[o] and many papers describing a

[o]Duff, 1977,[55] p. 501.

variety of applications can be found in the Proceedings of the 1980 IMA Conference[p] and in other publications[q]

Good, robust sparse matrix software is now commercially available. The Sparse Matrix Software Catalog[r] lists more than 120 programs. Many subroutines are described in the Harwell Catalogue[s] and two surveys have also been published.[t] Producing a good piece of sparse matrix software is not an easy task. It requires expert programming skills. As in any field of engineering, the software designer must build a prototype, test it carefully[u] and improve it before the final product is obtained and mass production starts. In software engineering, mass production is equivalent to obtaining multiple copies of a program and implementing them in many different installations. This requires transportability. From the point of view of the user, the software engineer must assume responsibility for choosing the right program and file structures and installing them into the computer. For the user, the product is not the program but the result. The desirable attributes of a good program are not easily achieved.[v] In this book, the characteristics and availability of software for each particular application are discussed in the corresponding sections.

I would like to acknowledge the collaboration of Neil Callwood. He has read the manuscript several times, correcting many of my grammatical infelicities, and is responsible for the "British flavour" that the reader may find in some passages. I would also like to acknowledge the patience and dedication Mrs. Carlota R. Glücklich while typing the manuscript and coping with our revisions.

Sergio Pissanetzky *January 1984*
Sergio@SciControls.com

[p] Duff, 1981b.[61]
[q] Bunch and Rose, 1976;[28] Duff and Stewart, 1979. [68]
[r] Heath, 1982.[126]
[s] Hopper, 1980.[130]
[t] Duff, 1982;[62] Parlett, 1983. [176]
[u] Duff, 1979;[57] Eisenstat *et al.* 1979;[75] Duff *et al.* 1982.[69]
[v] Gentleman and George, 1976;[87] Silvester, 1980.[215]

$$
\mathsf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\left|\begin{array}{ccccc}
A_{11} & & A_{13} & A_{14} & \\
 & A_{22} & & & A_{25} \\
 & & A_{33} & & A_{35} \\
 & & & A_{44} & \\
 & \text{symmetric} & & & A_{55}
\end{array}\right|
\end{array}
$$

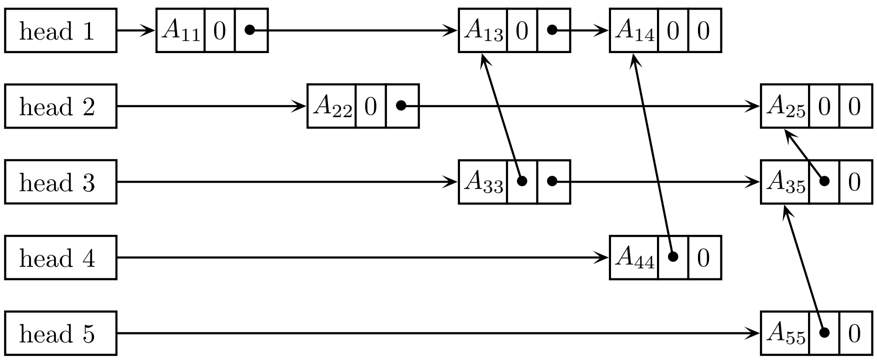


Figure 1.4: Larcombe's version of Knuth's storage scheme for symmetric matrices with no zero elements on the diagonal.

1.8 The sparse row-wise format

The *sparse row-wise format* (Chang, 1969;[29] Curtis and Reid, 1971b;[46] Gustavson, 1972[112]) to be described here is one of the most commonly used storage schemes for sparse matrices. The scheme has minimal storage requirements and at the same time it has proved to be very convenient for several important operations such as addition, multiplication, permutation and transposition of sparse matrices, the solution of linear equations with a sparse matrix of coefficients by either direct or iterative methods, etc. In this scheme, the values of the nonzero elements of the matrix are stored by rows, along with their corresponding column indices, in two arrays, say AN and JA, respectively. An array of pointers, say IA, is also provided to indicate the locations in AN and JA where the description of each row begins. An extra entry in IA contains a pointer to the first empty position in JA and AN. An example is convenient at this point. Consider the matrix:

$$
\mathsf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\
\left|\begin{array}{cccccccccc}
0 & 0 & 1. & 3. & 0 & 0 & 0 & 5. & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 7. & 0 & 1. & 0 & 0
\end{array}\right|
\end{array}
$$

A is represented as follows:

$$\begin{array}{rcllllll} & & 1 & 2 & 3 & 4 & 5 & 6 \\ \mathrm{IA} & = & 1 & 4 & 4 & 6 & & \\ \mathrm{JA} & = & 3 & 4 & 8 & 6 & 8 & \\ \mathrm{AN} & = & 1. & 3. & 5. & 7. & 1. & \end{array} \qquad \mathrm{RR(C)O}$$

The description of row 1 of A begins at the position $\mathrm{IA}(1) = 1$ of AN and JA. Since the description of row 2 begins at $\mathrm{IA}(2) = 4$, this means that row 1 of A is described in positions 1, 2 and 3 of AN and JA. In this example:

IA(1)	=	1	first row begins at JA(1) and AN(1).
IA(2)	=	4	second row begins at JA(4) and AN(4)
IA(3)	=	4	third row begins at JA(4) and AN(4). Since this is the same position at which row 2 begins, this means that row 2 is empty.
IA(4)	=	6	this is the first empty location in JA and AN. The description of row 3 thus ends at position $6 - 1 = 5$ of JA and AN.

In general, row r of A is described in positions $\mathrm{IA}(r)$ to $\mathrm{IA}(r+1) - 1$ of JA and AN, except when $\mathrm{IA}(r+1) = \mathrm{IA}(r)$ in which case row r is empty. If matrix A has m rows, then IA has $m + 1$ positions.

This representation is said to be *complete* because the entire matrix A is represented, and *ordered* because the elements of each row are stored in the ascending order of their column indices. It is thus a **R**ow-wise **R**epresentation **C**omplete and **O**rdered, or RR(C)O.

The arrays IA and JA represent the structure of A, given as the set of the adjacency lists of the graph associated with A. If an algorithm is divided into a symbolic section and a numerical section (Section 1.12), the arrays IA and JA are computed by the symbolic section, and the array AN by the numerical section.

Gustavson (1972)[112] also proposed a variant of row-wise storage, suitable for applications requiring both row and column operations. A is stored row-wise as described, and in addition the structure of A^T is computed and also stored row-wise. A row-wise representation of the structure of A^T is identical to a column-wise representation of the structure of A. It can be obtained by transposition of the row-wise structure of A (Chapter 7). This scheme has been used, for example, for linear programming applications (Reid, 1976).[189]

A much simpler row-oriented scheme was proposed by Key (1973)[141] for unsymmetric matrices. The nonzeros are held in a two-dimensional array of size n by m, where n is the order of the matrix and m the maximum number of nonzeros in a row. This scheme is easy to manipulate but has the disadvantage that m may not be predictable and may turn out to be large.

1.9 Ordered and unordered representations

Sparse matrix representations do not necessarily have to be ordered, in the sense that the elements of each row can be stored in any order while still preserving the order of the rows. The matrix A

The kth step consists of the elimination of the nonzeros on column k of $\mathsf{A}^{(k)}$ *both above and below the diagonal.* Row k is first normalized by dividing all its elements by the diagonal element. Then, convenient multiples of the normalized row k are subtracted from all those rows which have a nonzero in column k either above or below the diagonal. The matrix $\mathsf{A}^{(k+1)}$ is thus obtained with zeros in its k initial columns. This process is continued until, at the end of step n, the identity matrix $\mathsf{A}^{(n+1)} \equiv \mathsf{I}$ is obtained. The kth step of Gauss-Jordan elimination by columns is equivalent to pre-multiplication of $\mathsf{A}^{(k)}$ by D_k^{-1} and by the complete column elementary matrix $(\mathsf{T}_k^C)^{-1}$:

$$\mathsf{A}^{(k+1)} = (\mathsf{T}_k^C)^{-1}\mathsf{D}_k^{-1}\mathsf{A}^{(k)} \tag{2.37}$$

where $\mathsf{A}^{(1)} \equiv \mathsf{A}$ and:

$$\begin{aligned} (D_k)_{kk} &= A_{kk}^{(k)} \\ (T_k^C)_{ik} &= A_{ik}^{(k)} \qquad \text{for all } i \neq k \end{aligned} \tag{2.38}$$

Thus, we have:

$$(\mathsf{T}_n^C)^{-1}\mathsf{D}_n^{-1} \ldots (\mathsf{T}_2^C)^{-1}\mathsf{D}_2^{-1}(\mathsf{T}_1^C)^{-1}\mathsf{D}_1^{-1}\mathsf{A} = \mathsf{I}. \tag{2.39}$$

The factorized form of A is:

$$\mathsf{A} = \mathsf{D}_1\mathsf{T}_1^C\mathsf{D}_2\mathsf{T}_2^C \ldots \mathsf{D}_n\mathsf{T}_n^C, \tag{2.40}$$

and the *product form of the inverse* in terms of column matrices is:

$$\mathsf{A}^{-1} = (\mathsf{T}_n^C)^{-1}\mathsf{D}_n^{-1} \ldots (\mathsf{T}_2^C)^{-1}\mathsf{D}_2^{-1}(\mathsf{T}_1^C)^{-1}\mathsf{D}_1^{-1}. \tag{2.41}$$

The close relationship between this expression and the elimination form of the inverse, Expression (2.24), will be discussed in Section 2.10. The results of the elimination are usually recorded as a table of factors:

$$\begin{matrix} (D_1)_{11}^{-1} & (T_2^C)_{12} & (T_3^C)_{13} \ldots \\ (T_1^C)_{21} & (D_2)_{22}^{-1} & (T_3^C)_{23} \\ (T_1^C)_{31} & (T_2^C)_{32} & (D_3)_{33}^{-1} \\ \vdots & \vdots & \vdots \end{matrix} \tag{2.42}$$

By Equation (2.38), this table is formed simply by leaving each off-diagonal $A_{ik}^{(k)}$ where it is obtained. The diagonal is obtained, as in Gauss elimination, by storing the reciprocals of the diagonal elements used to normalize each row. The lower triangle and diagonal of this table are thus identical to those of the Gauss table. Expressions (2.40) and (2.41) indicate how to use the table (2.42). When solving linear equations by means of $\mathbf{x} = \mathsf{A}^{-1}\mathbf{b}$, Equation (2.41) is used, with the matrices $(\mathsf{T}_k^C)^{-1}$ obtained from the table by reversing the signs of the off-diagonal elements of column k (Property 2.4(d)). The matrices D_k^{-1} are directly available from the table. The product of A with any matrix or vector can also be computed using the table, as indicated by Equation (2.40).

Gauss-Jordan elimination can also be performed by rows. The version by columns requires the addition of multiples of row k to all other rows in order to cancel the off-diagonal elements of column k. This process can be understood conceptually as the construction of new equations which are linear combinations of the original ones. On the other hand, in Gauss-Jordan elimination by rows, we add multiples of *column* k to all other columns, in such a way that the off-diagonal elements of *row* k become zero. This process can be viewed as the construction of new *unknowns* which are linear combinations of the original ones and which satisfy linear equations with some zero coefficients. Alternatively, we can forget about the system of linear equations and view the row algorithm as the triangularization of A^T, the transpose of A, by columns. Doing this, we obtain the equivalent of Expression (2.41):

$$(\mathsf{A}^T)^{-1} = (\mathsf{T}_n'^C)^{-1}(\mathsf{D}_n')^{-1} \ldots (\mathsf{T}_2'^C)^{-1}(\mathsf{D}_2')^{-1}(\mathsf{T}_1'^C)^{-1}(\mathsf{D}_1')^{-1}, \tag{2.43}$$

which by transposition and using $(\mathsf{A}^T)^{-1} = (\mathsf{A}^{-1})^T$ yields:

$$\mathsf{A}^{-1} = (\mathsf{D}_1')^{-1}(\mathsf{T}_1'^R)^{-1}(\mathsf{D}_2')^{-1}(\mathsf{T}_2'^R)^{-1} \ldots (\mathsf{D}_n')^{-1}(\mathsf{T}_n'^R)^{-1} \tag{2.44}$$

Equation (2.44) is the product form of the inverse in terms of row matrices. The elimination by rows is equivalent to multiplying A *from the right* by Expression (2.44). The nontrivial elements of the matrices of Expression (2.44) are recorded as a table of factors in the usual way, and the table can be used to solve linear equations or to multiply either A or A^{-1} by any matrix or vector.

2.10 Relation between the elimination form of the inverse and the product form of the inverse

From the preceding section it should be clear that Gauss-Jordan elimination by columns can be performed equally well if we first eliminate all nonzeros from the lower triangle of A, and then all nonzeros from the upper triangle of A. In fact, when we start at the upper left-hand corner of A, we can eliminate lower and upper portions of columns in any order, provided only that upper portions are eliminated in order, lower portions are also eliminated in order, and the upper portion of any column k is eliminated *after* the lower portion of the preceding column. This statement holds true due to the fact that a row $k+1$ is obtained in final form immediately after the lower portions of columns 1 to k have been eliminated and row $k+1$ has been normalized; row $k+1$ can then be used either immediately or at any later stage to eliminate the upper portion of column $k+1$, provided that the upper portions of columns 1 to k have been previously eliminated. These facts can be stated formally using the properties of the elementary matrices (Section 2.4). We use Property 2.4(c) to express T_k^C as follows:

$$\mathsf{T}_k^C = \mathsf{L}_k^C \mathsf{U}_k^C, \tag{2.45}$$

Table 3.1. Bounds for the norms of L, expression for n_{ij} (see Equation 2.16), and bounds for the norm of the error matrix E for the factorization $\mathsf{LU} = \mathsf{A} + \mathsf{E}$, where all matrices are of order n. The bandwidth of band matrices is assumed not to exceed n.

A	Bounds for L	n_{ij}	Error bounds for factorization
Sparse	$\lVert\mathsf{L}\rVert_1 \le a_M \left(\max_j c_j^L + 1\right)$	$\sum_{k=1}^{m} n_{ij}^{(k)}$	$\lVert\mathsf{E}\rVert_1 \le 3.01\varepsilon_M a_M \max_j \sum_{i=1}^{n} n_{ij}$
	$\lVert\mathsf{L}\rVert_\infty \le a_M \left(\max_i r_i^L + 1\right)$	$m = \min(i, j)$	$\lVert\mathsf{E}\rVert_\infty \le 3.01\varepsilon_M a_M \max_i \sum_{j=1}^{n} n_{ij}$
Full	$\lVert\mathsf{L}\rVert_1 \le a_M n$	$\min(i, j)$	$\lVert\mathsf{E}\rVert_1, \lVert\mathsf{E}\rVert_\infty \le \frac{3.01}{2}\varepsilon_M a_M n(n+1)$
	$\lVert\mathsf{L}\rVert_\infty \le a_M n$		
Band	$\lVert\mathsf{L}\rVert_1 \le a_M(\beta+1)$	$\max[0, \min(i, j, i-j+\beta+1,$	$\lVert\mathsf{E}\rVert_1, \lVert\mathsf{E}\rVert_\infty \le 3.01\varepsilon_M a_M(\beta+1)^2$
$\lvert\backslash\beta\backslash\beta\backslash\rvert$	$\lVert\mathsf{L}\rVert_\infty \le a_M(\beta+1)$	$j-i+\beta+1)]$	
Band	$\lVert\mathsf{L}\rVert_1 \le a_M(\beta+1)$	$\max[0, \min(i, j, i-j+2\beta+1,$	$\lVert\mathsf{E}\rVert_1, \lVert\mathsf{E}\rVert_\infty \le 3.01\varepsilon_M a_M(\beta+1)$
$\lvert\backslash\beta\backslash 2\beta\backslash\rvert$	$\lVert\mathsf{L}\rVert_\infty \le a_M(\beta+1)$	$j-i+\beta+1, \beta+1)]$	$\times(2\beta+1)$

Then, the computed result $\mathbf{w}$ satisfies the exact relation:

$$\mathsf{L}\mathbf{w} = \mathbf{b} + \delta\mathbf{b} \tag{3.52}$$

where, from Equations 3.47 and 3.50, the following bounds hold for the components of $\delta\mathbf{b}$:

$$|\delta b_i| \le 3.01\varepsilon_M b_{Mi}(r_i^L + 1). \tag{3.53}$$

A less tight but simpler bound is obtained if b_M is the absolute value of the largest element of all the vectors $\mathbf{b}^{(k)}$, so that $b_{Mi} \le b_M$ and:

$$|b_i^{(k)}| \le b_M; \qquad i = 1, 2, \ldots, n; \qquad k \le i. \tag{3.54}$$

Then:

$$|\delta b_i| \le 3.01\varepsilon_M b_M(r_i^L + 1). \tag{3.55}$$

Backward substitution is the solution of $\mathsf{U}\mathbf{x} = \mathbf{w}$. It can be viewed as an algorithm with n steps, where the sequence of vectors $\mathbf{w}^{(n)} \equiv \mathbf{w}, \mathbf{w}^{(n-1)}, \ldots, \mathbf{w}^{(2)}, \mathbf{w}^{(1)}$ is computed, with $\mathbf{w}^{(k)}$ and $\mathbf{w}^{(k-1)}$ having their components k to n identical. Step k, $k = n, n-1, \ldots, 1$, is:

$$\begin{aligned} x_k &= w_k^{(k)} \\ w_i^{(k-1)} &= w_i^{(k)} - U_{ik}x_k + g_i^{(k)}; \qquad i = 1, \ldots, k-1, \end{aligned} \tag{3.56}$$

where $g_i^{(k)}$ is the error introduced by the floating point computation. The operations performed on an element $w_i, i < n$, are:

$$w_i - U_{in}x_n + g_i^{(n)} + g_i^{(n)} - U_{i,n-1}x_{n-1} + g_i^{(n-1)} - \ldots - U_{i,i+1}x_{i+1} + g_i^{(i+1)} = x_i \tag{3.57}$$

or:

$$w_i + \sum_{k=i+1}^{n} g_i^{(k)} = \sum_{k=i}^{n} U_{ik}x_k; \qquad i < n, \tag{3.58}$$

Thus, if we define the error vector $\delta\mathbf{w}$:

$$\begin{aligned} &\delta w_i = \sum_{k=i+1}^{n} g_i^{(k)}; \qquad i < n \\ &\delta w_n = 0, \end{aligned} \tag{3.59}$$

we have the following exact relation between the computed numbers:

$$\mathsf{U}\mathbf{x} = \mathbf{w} + \delta\mathbf{w}. \tag{3.60}$$

In order to obtain bounds for $\delta\mathbf{w}$, we let $w_{Mi} = \max_k |w_i^{(k)}|$, so that:

$$|w_i^{(k)}| \leq w_{Mi}; \qquad 1 \leq i \leq n; \qquad i \leq k \leq n. \tag{3.61}$$

In particular, for $k = i, w_i^{(i)} = x_i$, so that $|x_i| \leq w_{Mi}$. We also let w_M be the largest w_{Mi}; therefore:

$$|w_i^{(k)}| \leq w_M; \qquad i = 1, 2, \ldots, n; \qquad k \geq i. \tag{3.62}$$

Then, using Equation 3.22:

$$|g_i^{(k)}| \leq 3.01\varepsilon_M w_{Mi}; \qquad k > i \tag{3.63}$$

and

$$|\delta w_i| \leq 3.01\varepsilon_M w_{Mi} r_i^U, \tag{3.64}$$

where r_i^U is the number of off-diagonal nonzeros in row i of U. Alternatively, using Equation 3.62:

$$|\delta w_i| \leq 3.01\varepsilon_M w_M r_i^U. \tag{3.65}$$

Finally, we consider the residual

$$\mathbf{r} = \mathsf{A}\mathbf{x} - \mathbf{b} \tag{3.66}$$

obtained when the solution $\mathbf{x}$ of System 3.1 is computed using floating point arithmetic. Using Equations 3.41, 3.52 and 3.60, we obtain:

$$\mathbf{r} = -\mathsf{E}\mathbf{x} + \mathsf{L}\delta\mathbf{w} + \delta\mathbf{b}. \tag{3.67}$$

Taking the 1-norm or the ∞-norm, we have:

$$\|\mathbf{r}\| \leq \|\mathsf{E}\|\|\mathbf{x}\| + \|\mathsf{L}\|\|\delta\mathbf{w}\| + \|\delta\mathbf{b}\|. \tag{3.68}$$

From Equation 3.62 we obtain bounds for the norms of $\mathbf{x}$:

$$\begin{aligned} \|\mathbf{x}\|_1 &\leq n w_M \\ \|\mathbf{x}\|_\infty &\leq w_M. \end{aligned} \tag{3.69}$$

Bounds for the norms of E and L are given in table 3.1. Bounds for the norms of $\delta\mathbf{w}$ and $\delta\mathbf{b}$ were obtained from Equations 3.65 and 3.55, respectively, and are listed in Table 3.2. Thus, a bound for $\|\mathbf{r}\|$ can be computed using Equation 3.68.

The residual $\mathbf{r}$ has another interpretation. Let $\tilde{\mathbf{x}}$ be the exact solution of Equation 3.1; then $\mathsf{A}\tilde{\mathbf{x}} = \mathbf{b}$ and

$$\mathbf{r} = \mathsf{A}(\mathbf{x} - \tilde{\mathbf{x}}), \tag{3.70}$$

Table 3.2. Values of some parameters and bounds for the norms $\delta\mathbf{b}$ and $\delta\mathbf{w}$ for forward and backward substitution.

A	Parameters of Section 3.4	Forward substitution	Backward substitution
Sparse	See Section 3.4	$\|\delta\mathbf{b}\|_1 \leq 3.01\varepsilon_M b_M n_L$ $\|\delta\mathbf{b}\|_\infty \leq 3.01\varepsilon_M b_M \left(\max_i r_i^L + 1\right)$	$\|\delta\mathbf{w}\|_1 \leq 3.01\varepsilon_M w_M n'_U$ $\|\delta\mathbf{w}\|_\infty \leq 3.01\varepsilon_M w_M \max_i r_i^U$
Full	$r_i^L = i - 1$ $r_i^U = n - i$ $n_L = n(n+1)/2$ $n'_U = n(n-1)/2$	$\|\delta\mathbf{b}\|_1 \leq (3.01/2)\varepsilon_M b_M n(n+1)$ $\|\delta\mathbf{b}\|_\infty \leq 3.01\varepsilon_M b_M n$	$\|\delta\mathbf{w}\|_1 \leq (3.01/2)\varepsilon_M w_M n(n-1)$ $\|\delta\mathbf{w}\|_\infty \leq 3.01\varepsilon_M w_M (n-1)$
Band $\|\backslash\beta\backslash\beta\backslash\|$	$r_i^L = \min(i-1, \beta)$ $r_i^U = \min(n-i, \beta)$ $n_L = (n-\beta/2)(\beta+1)$ $n'_U = (n-\beta/2-1/2)\beta$	$\|\delta\mathbf{b}\|_1 \leq 3.01\varepsilon_M b_M (n-\beta/2)(\beta+1)$ $\|\delta\mathbf{b}\|_\infty \leq 3.01\varepsilon_M b_M (\beta+1)$	$\|\delta\mathbf{w}\|_1 \leq 3.01\varepsilon_M w_M (n-\beta/2-1/2)\beta$ $\|\delta\mathbf{w}\|_\infty \leq 3.01\varepsilon_M w_M \beta$
Band $\|\backslash\beta\backslash 2\beta\backslash\|$	$r_i^L = \min(i-1, \beta)$ $r_i^U = \min(n-i, 2\beta)$ $n_L = (n-\beta/2)(\beta+1)$ $n'_U = (2n-2\beta-1)\beta$	$\|\delta\mathbf{b}\|_1 \leq 3.01\varepsilon_M b_M (n-\beta/2)(\beta+1)$ $\|\delta\mathbf{b}\|_\infty \leq 3.01\varepsilon_M b_M (\beta+1)$	$\|\delta\mathbf{w}\|_1 \leq 3.01\varepsilon_M w_M (2n-2\beta-1)\beta$ $\|\delta\mathbf{w}\|_\infty \leq 6.02\varepsilon_M w_M \beta$

spondence between fill-ins and new edges added to the graph is evident. The reader can finish the exercise.

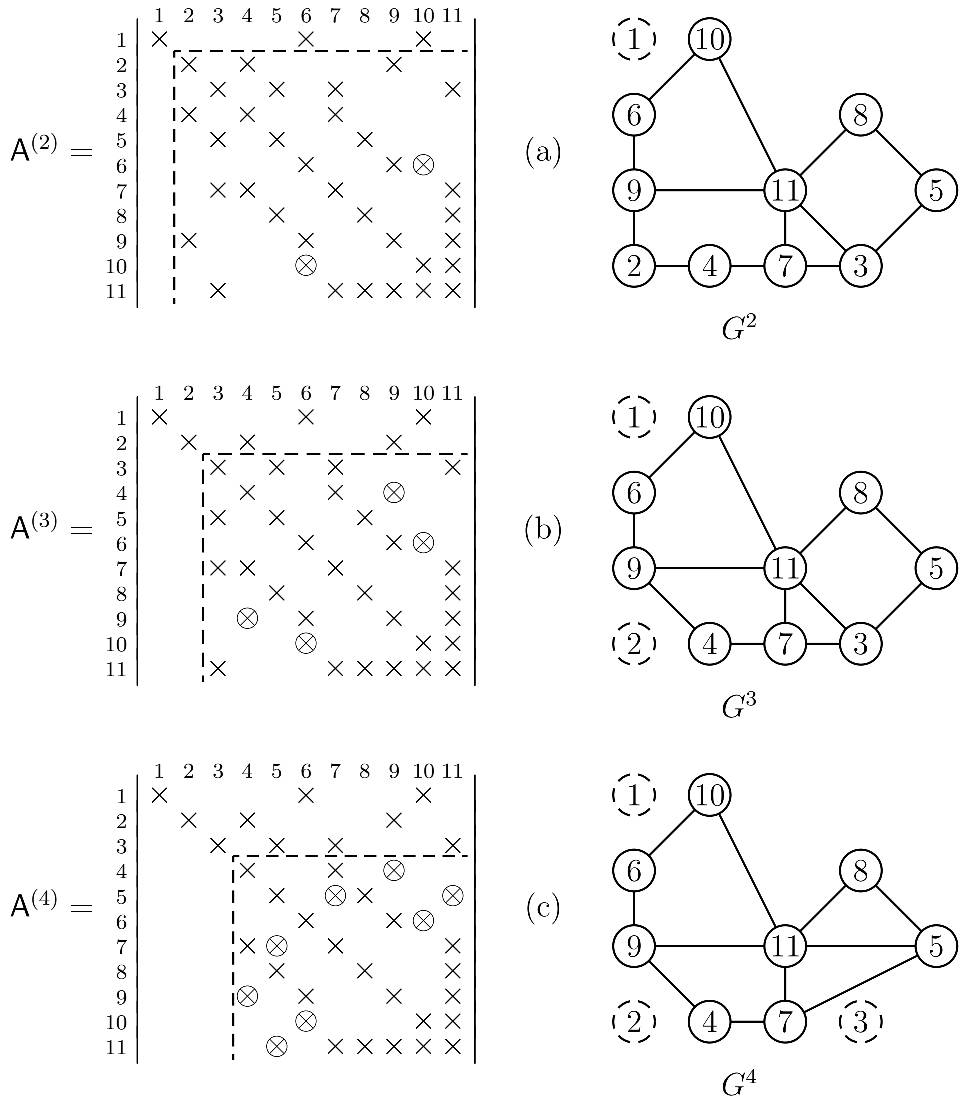


Figure 4.8: The three initial elimination steps and the corresponding elimination graphs for the matrix of Fig. 4.1(a). Fill-ins are encircled.

In terms of graph theory, Parter's rule says that the adjacent set of vertex k becomes a clique when vertex k is eliminated. Thus, Gauss elimination generates cliques systematically. Later, as elimination progresses, cliques grow or sets of cliques join to form larger cliques, a process known

$Y = \{14, 16, 1, 7\}$ and finds that Y has adjacent vertices in L_5 which have not yet been placed in any partition. Thus $S = \{7\}$ is pushed onto the stack and the algorithm branches to Step 5, where, picking $v_5 = 13$, it is found that the path can not be prolonged any longer, so $t = 1$. Letting $S = \{13\}$, the algorithm continues with Step 1, where S is not modified, and with Step 2, where Y is determined to be $\{13, 15\}$, which becomes the third partition member.

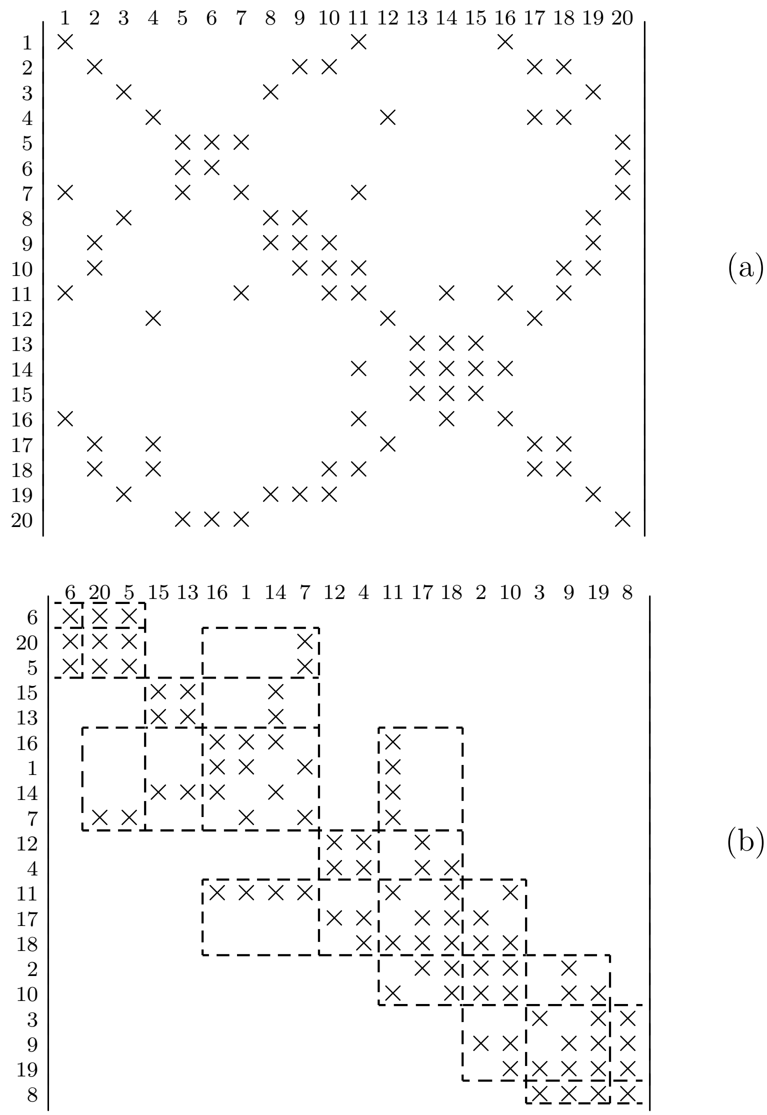
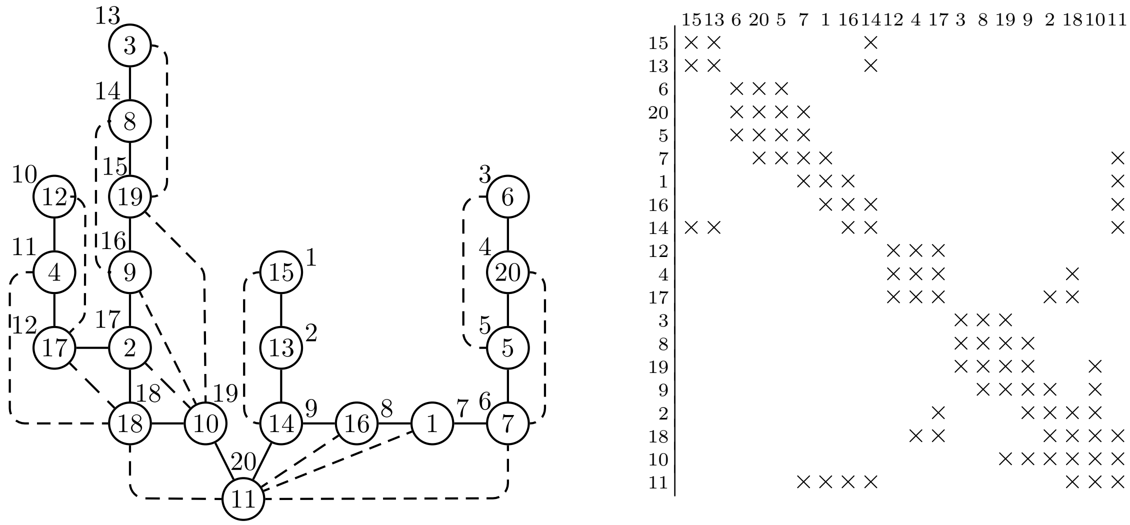


Figure 4.12: The Refined Quotient Tree algorithm. (a) Structure of the matrix corresponding to the graph of Fig. 4.2(a). (b) The permuted block matrix corresponding to the quotient tree of Fig. 4.2(c).

Table 4.1

Class of graphs	Bound for fill-in	Bound for multiplication count	Observations and references
Any, such that $\sigma = 1/2$	$c_3 n \log_2 n + O(n)$	$c_7 n^{3/2} + O[n(\log n)^2]$	Ordering time is $O[(m+n)\log n]$ if separators can be found in $O(m+n)$ time. c_3 and c_7 given by Eq. 4.23 (Lipton *et al.*, 1977[152]
Planar graphs (in this case $\sigma = 1/2, \alpha = 2/3, \beta = 2\sqrt{2}$)	$c_3 n \log n + O(n)$	$c_7 n^{3/2} + O[n(\log n)^2]$	$c_3 \leq 129, c_7 \leq 4002$. Ordering time is $O(n \log n)$ (Lipton and Tarjan, 1979[151]; Lipton *et al.*, 1979[153])
Two-dimensional finite element graphs (in this case $\sigma = 1/2, \alpha = 2/3, \beta = 4\lfloor k/2 \rfloor$)	$O(k^2 n \log n)$	$O(k^3 n^{3/2})$	k is the maximum number of boundary nodes of the elements. Ordering time is $O(n \log n)$ (Lipton *et al.*, 1979[153])
Regular planar grid	$\frac{31}{8} n \log_2 n + O(n)$	$\frac{829}{84} n^{3/2} + O(n \log_2 n)$	(George and Liu, 1981[97])
Any such that $\sigma > 1/2$	$O(n^{2\sigma})$	$O(n^{3\sigma})$	(Lipton *et al.*, 1979[153])
Three-dimensional grid graphs (in this case $\sigma = 2/3$)	$O(n^{4/3})$	$O(n^2)$	(Lipton *et al.*, 1979[153])
Any, such that $1/3 < \sigma < 1/2$	$O(n)$	$O(n^{3\sigma})$	(Lipton *et al.*, 1979[153])
Any, such that $\sigma = 1/3$	$O(n)$	$O(n \log_2 n)$	(Lipton *et al.*, 1979[153])
Any, such that $\sigma < 1/3$	$O(n)$	$O(n)$	(Lipton *et al.*, 1979[153])

The idea is illustrated in Fig. 4.19(a), where the rectangle represents the set of nodes of a two-dimensional finite element grid. Choose σ small separators ($\sigma = 3$ in the figure) which consist of grid lines and dissect the grid into $\sigma + 1$ blocks $R_1, R_2, \ldots$ of comparable size. If all separators are considered to form another single block, a tree partitioning is obtained as shown by the quotient tree of Fig. 4.19(b). The advantages of tree partitioning regarding the reduction of fill-in and operation count were discussed in Section 4.9. Now, let us number the nodes of each R-set sequentially, following lines from left to right as closely as possible, and starting at the bottom left as indicated by the arrows. When all R-sets have been numbered, the separators are also numbered sequentially, as the arrows show. The numbering corresponds to a monotone ordering of the tree. The matrix associated with the finite element grid is partitioned into blocks as shown in Fig. 4.19(c), where all nonzeros are confined to the cross-hatched areas. If Gauss elimination is performed on this matrix, fill-in will result only inside the cross-hatched areas and in the dotted areas. Besides, the hatched blocks are not completely full. For example, the four leading diagonal blocks are banded.



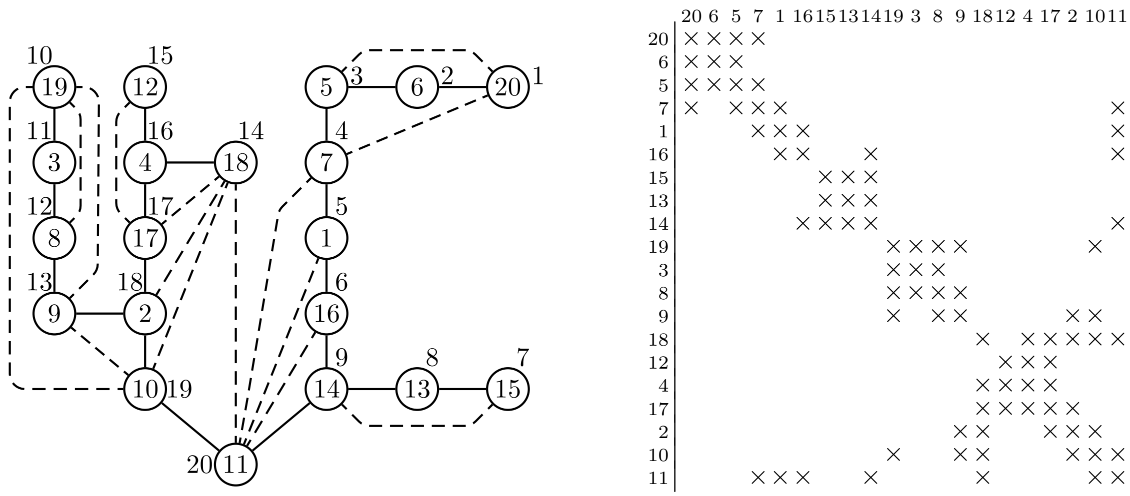
(a) (b)

Figure 4.25: Reverse depth-first ordering, short frond strategy, for the graph of Fig. 4.2(a).

in favor of vertex 19, which is adjacent to two visited vertices: 9 and 10. The reader may continue the search and verify that the spanning tree and reverse depth-first ordering shown in Fig. 4.25(a) may be obtained. The separators (11), (10, 18, 2) and (14) can be immediately identified. The corresponding permuted matrix is shown in Fig. 4.25(b). No fill-in at all is produced by elimination on this matrix, a result obtained at a very low computational cost. The reason why an ordering with no fill-in exists for the graph of Fig. 4.2(a) is that this graph is triangulated (Rose, 1970[194]), see Section 4.16.

Now consider the application of the long frond strategy to the same graph. Again 11 is the starting vertex. Vertices 10 and 18 are the next candidates, both of degree 5. We arbitrarily select vertex 10. At this point $V_v = \{11, 10\}$, and vertices 18, 2, 9 and 19 all have three edges leading to vertices not in V_v. Vertex 18 is discarded because it is adjacent to both visited vertices, while 2, 9 and 19 are adjacent to only one of the visited vertices. Let us choose vertex 2 to be the next vertex to visit.

At this point $V_v = \{11, 10, 2\}$ and $|\mathrm{Adj}(w) - V_v|$ is equal to 3, 2 and 2 for vertices 17, 18 and 9, respectively. Thus, we select vertex 17. Next is vertex 4, which introduces two new edges (while 12 or 18 would have introduced only one), and finally vertex 12, which is adjacent to only two visited vertices (while 18 is adjacent to five). On backtracking to vertex 4 we find the tree arc (4, 18). Figure 4.26(a) shows one possible ordering obtained in this way. The four separators (11), (10, 2), (17, 4) and (14) can be identified. As expected, this strategy has produced more separators than the short frond strategy. The corresponding permuted matrix is shown in Fig. 4.26(b). Elimination would produce 10 fill-ins in this matrix.



(a) (b)

Figure 4.26: Reverse depth-first ordering, long frond strategy, for the graph of Fig. 4.2(a).

When the user is dealing with a large problem, a sophisticated ordering algorithm may be convenient, and may even determine whether the problem is tractable or not. For a medium-size problem, a simple ordering technique may often produce a large improvement as compared with no ordering at all, at a low programming cost.

4.16 Lexicographic search

In this section we continue the analysis of low fill orderings for symmetric matrices, but now from a different point of view. We consider a special class of matrices which can be ordered in such a way that Gauss elimination would cause no fill-in. Then we take advantage of the properties of such matrices to give a procedure which finds a low fill ordering for any symmetric matrix. As usual, we discuss the ideas in terms of graph theory. Let $G^A = (V, E)$ be the undirected graph associated with a symmetric matrix A, and let $G^F = (V, E \cup F)$ be the corresponding filled graph associated with $\mathsf{U} + \mathsf{U}^T$, where $\mathsf{A} = \mathsf{U}^T\mathsf{D}\mathsf{U}$ is the factorization of A and F is the set of new edges (or nonzeros of U) introduced during factorization. If the graph G^A has an elimination ordering for which $F = \emptyset$, i.e., no fill-in is produced if elimination is carried out in that order, we say that G^A is a *perfect elimination graph*. The ordering itself is called a *perfect elimination ordering*. Note that fill-in may result if we eliminate in a different order, even when G^A is a perfect elimination graph. Note also that every elimination graph G^F is a perfect elimination graph since no fill-in would result if elimination were performed again in the same order.

Chapter 6

Sparse Eigenanalysis

6.1 Introduction

The standard eigenvalue problem is defined by

$$\mathsf{A}\,\mathbf{x} = \lambda\,\mathbf{x} \tag{6.1}$$

where A is the given n by n matrix. It is desired to find the *eigenpairs* $(\lambda, \mathbf{x})$ of A, where λ is an *eigenvalue* and $\mathbf{x}$ is the corresponding *eigenvector*. The generalized eigenvalue problem is

$$\mathsf{A}\,\mathbf{x} = \lambda\,\mathsf{B}\,\mathbf{x} \tag{6.2}$$

where A and B are given n by n matrices and again we wish to determine λ and $\mathbf{x}$. For historical reasons the pair A, B is called a *pencil* (Gantmacher, 1959[83]). When $\mathsf{B} = \mathsf{I}$ the generalized problem reduces to the standard one.

Both for simplicity and to follow the general trend imposed by most of the literature and existing software, we restrict the analysis to the case where A is real symmetric and B is real symmetric and positive definite, except when stated otherwise. Almost all the results become valid for hermitian matrices when the conjugate transpose superscript H is written in place of the transpose superscript T. On the other hand, an eigenvalue problem where A or A and B, are hermitian, can be solved using software for real matrices (Section 6.15).

Equation 6.1 has a nonzero solution $\mathbf{x}$ when

$$\mathrm{Det}(\mathsf{A} - \lambda\,\mathsf{I}) = 0. \tag{6.3}$$

This is a polynomial equation of the nth degree in λ, which has n roots $\lambda_1, \lambda_2, \ldots, \lambda_n$. The roots are the eigenvalues of A, and they may be either all different or there may be multiple roots with any *multiplicity*. When A is real symmetric, the eigenvalues are all real. The simplest example is the identity matrix I, which has an eigenvalue equal to 1 with multiplicity n. To each eigenvalue array of pointers `IC` at lines 5 and 24. The multiple switch array `IX` is initialized to 0 at lines 2 and 3. `I`, defined at line 4, identifies each row. The `DO 20` loop scans row `I` of the first given matrix: the column indices, if any, are stored in `JC` at line 11 and the row index is stored in `IX` at line 13, thus turning "on" the corresponding switch. The `DO 40` loop runs over row `I` of the second matrix. For each column index `J`, defined at line 18, the multiple switch is tested at line 19: if the value of `IX(J)` is `I`, then the switch is on, which means that `J` has already been added to the list `JC` and should not be added again. Otherwise, `J` is added to `JC` at line 20. The reader may expect that the sentence `IX(J)=I` should appear between lines 21 and 22 in order to record the fact that the column index `J` has been added to the list `JC`. However, such a record is now not necessary because, during the processing of row `I`, the same value of `J` will never be found again: there are no repeated column indices in the representation of row `I` in the array `JB`.

7.11 Algorithm for the numerical addition of two sparse matrices with N rows

Input:	`IA, JA, AN`	first given matrix in RR(C)U.
	`IB, JB, BN`	second given matrix in RR(C)U.
	`IC, JC`	structure of the resulting matrix in RR(C)U.
	`N`	number of rows of the matrices.
Output:	`CN`	numerical values of the nonzeros of the resulting matrix.
Workspace:	`X`	expanded array used to accumulate the nonzeros; the dimension of `X` is `M`, the number of columns of the matrices.

```
      DO 70 I=1,N
      IH=I+1
      ICA=IC(I)
      ICB=IC(IH)-1
      IF(ICB.LT.ICA)GO TO 70
      DO 10 IP=ICA,ICB
10    X(JC(IP))=0.
      IAA=IA(I)
      IAB=IA(IH)-1
      IF(IAB.LT.IAA)GO TO 30
      DO 20 IP=IAA,IAB
20    X(JA(IP))=AN(IP)
```

213 Sherman, A. H. (1975). On the efficient solution of sparse systems of linear and nonlinear equations. Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, CT. Report No. 46.

214 Shirey, R. W. (1969). “Implementation and analysis of efficient graph planarity testing algorithms.” Ph.D. Dissertation, University of Wisconsin, Madison, WI.

215 Silvester, P. P. (1980). Software engineering aspects of finite elements. In Chari abd Silvester (1980) [30], pp. 69-85.

216 Skeel, R. D. (1981). Effect of equilibration on residual size for partial pivoting. *SIAM J. Numer. Anal.* **18**, 449-454.

217 Smith, B. T., Boyle, J. M., Dongarra, J. J., Garbow, B. S., Ikebe, Y., Klema, V. C. and Moler, C. B. (1976). *Matrix Eigensystem Routines – EISPACK Guide.* Lecture Notes in Computer Science, Vol. 6, 2nd. edn. Springer-Verlag: Berlin.

218 Speelpenning, B. (1978). “The generalized element method.” Department of Computer Science, University of Illinois, Urbana, Champaign, Il. Report UIUCDCS-R-78.

219 Stewart, G. W. (1973). *Introduction to Matrix Computations.* Academic Press; New York.

220 Stewart, G. W. (1974). Modifying pivot elements in Gaussian elimination. *Math. Comput.* **28,** 537-542.

221 Stewart, G. W. (1976a). A bibliographical tour of the large sparse generalized eigenvalue problem. In Bunch and Rose (1976) [28], pp. 113-130.

222 Stewart, G. W. (1976b). Simultaneous iteration for computing invariant subspaces of non-hermitian matrices. *Numer. Math.* **25,** 123-136.

223 Stewart, W. J. and Jennings, A. (1981). Algorithm 570.LOPSI: A simultaneous iteration algorithm for real matrices. *ACM Trans. Math. Software* **7,** 230-232.

224 Strassen, V. (1969). Gaussian elimination is not optimal. *Numer. Math.* **13,** 354-356.

225 Swift, G. (1960). A comment on matrix inversion by partitioning. *SIAM Rev.* **2,** 132-133.

226 Szyld, D. B. and Vishnepolsky, O. (1982). “Some operations on sparse matrices.” Institute for Economic Analysis, New York University. Private communication.

227 Takahashi, H. and Natori, M. (1972). Eigenvalue problem of large sparse matrices. *Rep. Comput. Cent. Univ. Tokyo* **4,** 129-148.

228 Tarjan, R. E. (1971). “An efficient planarity algorithm.” Computer Science Department, Stanford University, Stanford, CA. Technical Report 244.

229 Tarjan, R. E. (1972). Depth first search and linear graph algorithms. *SIAM J. Comput.* **1,** 146-160.

230 Tarjan, R. E. (1975). Efficiency of a good but not linear set union algorithm. *J. Assoc. Comput. Mach.* **22,** 215-225.

231 Tarjan, R. E. (1976). Graph theory and Gaussian elimination. In Bunch and Rose (1976) [28], pp. 3-22.

232 Tewarson, R. P. (1967). On the product form of inverses of sparse matrices and graph theory. *SIAM Rev.* **9,** 91-99.

233 Tewarson, R. P. (1967). Row-column permutation of sparse matrices. *Comput. J.* **10,** 300-305.

Index