Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

157

Ole Østerby
Zahari Zlatev

Direct Methods for Sparse Matrices

Springer-Verlag
Berlin Heidelberg New York Tokyo

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

157

Ole Østerby
Zahari Zlatev

Direct Methods for Sparse Matrices

Springer-Verlag
Berlin Heidelberg New York Tokyo 1983

Authors

Ole Østerby
Computer Science Department, Aa
DK 8000 Aarhus, Denmark

Zahari Zlatev
Air Pollution Laboratory
Danish Agency of Environmental Pr
Risø National Laboratory
DK 4000 Roskilde, Denmark

CR Subject Classifications (1982): G.1.3

ISBN 3-540-12676-7 Springer-Verlag Be
ISBN 0-387-12676-7 Springer-Verlag New York Heidelberg Berlin Tokyo

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

Preface

The mathematical models of many practical problems lead to systems of linear algebraic equations where the coefficient matrix is large and sparse. Typical examples are the solutions of partial differential equations by finite difference or finite element methods but many other applications could be mentioned.

When there is a large proportion of zeros in the coefficient matrix then it is fairly obvious that we do not want to store all those zeros in the computer, but it might not be quite so obvious how to get around it. We shall first describe storage techniques which are convenient to use with direct solution methods, and we shall then show how a very efficient computational scheme can be based on Gaussian elimination and iterative refinement.

A serious problem in the storage and handling of sparse matrices is the appearance of fill-ins, i.e. new elements which are created in the process of generating zeros below the diagonal. Many of these new elements tend to be smaller than the original matrix elements, and if they are smaller than a certain quantity which we shall call the drop tolerance we simply ignore them. In this way we may preserve the sparsity quite well but we probably introduce rather large errors in the LU decomposition to the effect that the solution becomes unacceptable. In order to retrieve the accuracy we use iterative refinement and we show theoretically and with practical experiments that it is ideal for the purpose.

Altogether, the combination of Gaussian elimination, a large drop tolerance, and iterative refinement gives a very efficient and competitive computational scheme for sparse problems. For dense matrices iterative refinement will always require more storage and computation time, and the extra accuracy it yields may not be enough to justify it. For sparse problems, however, iterative refinement combined with a large drop tolerance will in most cases give very accurate results and reliable error estimates with less storage and computation time.

A short description of the Gaussian elimination process is given in chapter 1. Different storage algorithms for general sparse matrices are discussed in chapter 2. Chapter 3 is devoted to the use of pivotal strategies as a tool for keeping the balance between sparsity and accuracy. The possibility of using an iterative refinement process in connection with the Gaussian elimination is the topic of chapter 4.

In chapter 5 we introduce a general computational scheme which includes many well-known direct methods for linear equations and for overdetermined linear systems as special cases. We also demonstrate how the above techniques can be generalized to linear least squares problems. Thus, we show that the theory of most of the direct methods can be studied from a common point of view and that the algorithms described in the previous chapters are applicable not only in connection with Gaussian elimination but also for many other methods. A particular algorithm (the Gentleman - Givens orthogonalization) is discussed in detail in the second part of chapter 5 as an illustration of the above statements.

The algorithms described in chapters 2 - 4 have been implemented in a package for the solution of large and sparse systems of linear algebraic equations. This package, Y12M, is included in the standard library at RECKU (the Regional Computing Centre at the University of Copenhagen). The subroutines of package Y12M with full documentation and with many test-programs are available at the usual cost (for the magnetic tape, machine time, shipment, etc.). Requests should be addressed to J. Wasniewski, RECKU, Vermundsgade 5, DK - 2100 Copenhagen. It should be mentioned that the subroutines are written in FORTRAN. Both double and single precision versions are available. No special features of the computer at the disposal at RECKU (UNIVAC 1100/82) have been exploited and no machine-dependent constants are used. Thus the package is portable and will work without any changes on many large computers. This has been verified by running the subroutines of the package on three different computers: a UNIVAC 1100/82 computer at RECKU, an IBM 3033 computer at the Northern Europe University Computing Centre (NEUCC) and a CDC Cyber 173 computer at the Regional Computing Centre at Aarhus University (RECAU).

The package Y12M also includes subroutines for estimation of the condition number of a sparse matrix. The subroutines can be called when the LU decomposition is calculated and provide a relatively inexpensive but still reliable measure of the sensitivity of the results to round-off errors.

A full documentation of the subroutines from package Y12M with a brief description of the basic ideas applied in the implementation is given in a previous volume of this series (see Z. Zlatev, J. Wasniewski and K. Schaumburg: "Y12M - Solution of Large and Sparse Systems of Linear Algebraic Equations", Lecture Notes in Computer Science, Vol. 121, Springer, Berlin-Heidelberg-New York, 1981).

Decimal notation is used for the numbering of sections and chapters. Thus the third section of chapter 5 is numbered 5.3. The 15th numbered equation in section 3 of chapter 5 is numbered (3.15) and is referenced in another chapter by (5.3.15). Tables and figures are numbered in each chapter. Thus the 7th table or figure in chapter 1 is numbered 1.7. A similar numbering system is used for theorems, corollaries, remarks, etc.

We would like to express our thanks to Angelika Paysen who with great patience and expert skill typed the manuscript.

Contents

Chapter 1: Introduction

1.1 Gaussian elimination

Many practical problems lead to large systems of linear algebraic equations

$$Ax = b, \tag{1.1}$$

where $n \in \tilde{N}$, $A \in \tilde{R}^{nxn}$ and $b \in \tilde{R}^{nx1}$ are given, with $\text{rank}(A) = n$, and $x \in \tilde{R}^{nx1}$ is to be computed.

In this book we shall discuss the solution of (1.1) by means of so-called direct methods and begin with the well known Gaussian elimination. The elimination process will be carried out in $n - 1$ stages

$$A^{(k+1)} = L^{(k)} \cdot A^{(k)}, \qquad (k = 1(1)n-1) \tag{1.2}$$

starting with $A^{(1)} = A$. The lower right $(n-k+1) \times (n-k+1)$ submatrix of $A^{(k)}$ is denoted A_k and its elements are denoted $a_{ij}^{(k)}$, $(i,j = k(1)n)$. For the elements of A_{k+1} we have the formula

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} \cdot a_{kj}^{(k)} / a_{kk}^{(k)}, \qquad i,j = k+1(1)n. \tag{1.3}$$

$L^{(k)}$ is an elementary unit lower triangular matrix with elements

$$\begin{aligned} l_{ii}^{(k)} &= 1, && (i = 1(1)n); \\ l_{ik}^{(k)} &= -a_{ik}^{(k)} / a_{kk}^{(k)}, && (i = k+1(1)n); \\ &\text{otherwise } 0. \end{aligned} \tag{1.4}$$

The end result of the elimination is the upper triangular matrix $U = A^{(n)}$ and the process is equivalent to a triangular factorization

$$A = L \cdot U, \tag{1.5}$$

where

$$L = (L^{(n-1)} \cdot L^{(n-2)} \cdot \ldots \cdot L^{(1)})^{-1}. \tag{1.6}$$

The elements of L and U are thus given by

$$(1.7) \qquad U = \begin{Bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & & a_{1n}^{(1)} \\ & a_{22}^{(2)} & a_{23}^{(2)} & & a_{2n}^{(2)} \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ & & & & \cdot \\ & & & & a_{nn}^{(n)} \end{Bmatrix},$$

and

$$(1.8) \qquad L = \begin{Bmatrix} 1 & & & & \\ -l_{21}^{(1)} & 1 & & & \\ -l_{31}^{(1)} & -l_{32}^{(2)} & 1 & 0 & \\ \cdot & & & \cdot & \\ \cdot & & & \cdot & \\ \cdot & & & & \cdot \\ -l_{n1}^{(1)} & -l_{n2}^{(2)} & \cdots & -l_{n,n-1}^{(n-1)} & 1 \end{Bmatrix}.$$

In order for this factorization to be successful it is necessary that all the denominators in (1.3), $a_{kk}^{(k)}$, be different from 0. Moreover, to ensure reasonably stable computations it is to be desired that the correction terms in (1.3), $a_{ik}^{(k)} \cdot a_{kj}^{(k)} / a_{kk}^{(k)}$ be reasonably small. This is usually accomplished by interchanging rows and/or columns and thus requiring that $|l_{ik}^{(k)}| \leq 1$ or $|a_{kj}^{(k)} / a_{kk}^{(k)}| \leq 1$. We shall return to this topic in section 3.1 and for the moment just prepare ourselves for the row and column interchanges which transform (1.1) into

$$(1.9) \qquad P A Q(Q^T x) = Pb,$$

where P and Q are permutation matrices.

The elimination or factorization (1.5) now becomes

$$LU = PAQ + E, \tag{1.10}$$

where L and U now denote the computed triangular matrices and E is a perturbation matrix which takes care of the computational errors, among other things.

An approximation $\tilde{x}$ to the solution x is now computed by substitution:

$$x_1 = Q U^{-1} L^{-1} P b, \tag{1.11}$$

and we set

$$\tilde{x} = x_1 \tag{1.12}$$

Definition 1.1 $\tilde{x}$ as given by (1.12) is called the direct solution (DS). ∎

Remark 1.2 Even if the computations in (1.11) are performed without errors we may still have $\tilde{x} \neq x$ if $E \neq 0$ in (1.10). ∎

We would expect that the process of elimination and substitution would lead to a 'good' solution if the elements of E are small. This is often the case but we have no a priori guarantee of this, and we don't even have any a priori guarantee that the elements of E will be small if we use only row-interchanges. Therefore the following 'refining' process can be useful.

Compute for $i = 1, 2, \ldots, q-1$

$$r_i = b - A x_i, \tag{1.13}$$

$$d_i = Q U^{-1} L^{-1} P r_i, \tag{1.14}$$

$$x_{i+1} = x_i + d_i, \tag{1.15}$$

and set

$$\tilde{x} = x_q. \tag{1.16}$$

Definition 1.3 The process described by (1.13) - (1.15) is called iterative refinement. $\tilde{x}$ as given by (1.16) is called the iteratively refined solution (IR). ∎

Remark 1.4 Under certain conditions the process (1.13) - (1.15) is convergent and $x_i \to x (i \to \infty)$. In this case $x = x_1 + \Sigma_{i=1}^{\infty} d_i$ and $d_i \to 0$. If the series converges swiftly $\|d_i\|$ can be used as an estimate of the error $\|x - x_i\|$. ∎

If convergent the iterative refinement will provide a better solution and a reasonable error estimate. The price we have to pay for this is extra storage (because a copy of A must be retained) and extra computing time (for the process (1.13) - (1.15)). The following table gives the storage and computing time for DS and IR

	DS	IR
Storage	$n^2 + O(n)$	$2n^2 + O(n)$
Time	$\frac{1}{3}n^3 + n^2 + O(n)$	$\frac{1}{3}n^3 + (2q - 1)n^2 + O(n)$

Table 1.1

Comparison of storage and time with DS and IR for dense matrices. The computation time is measured by the number of multiplications.

1.2 Sparse matrices

Until now we have tacitly assumed that we require space and time to treat all the n^2 elements of matrix A (A is dense). Table 1.1 shows that in this case both storage and time increase rapidly with n and that IR is always more expensive than DS in both respects.

In many applications, however, A is sparse, i.e. a large proportion of the elements of A are 0, and we shall in this book describe special techniques which can be used to exploit this sparsity of A.

The border-line between dense and sparse matrices is rather fluent, but we could 'define' a matrix to be sparse if we can save space and/or

time by employing the sparse matrix techniques to be described in this book.

Consider the basic formula in the factorization process (1.2)

$$(2.1) \qquad a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} \cdot a_{kj}^{(k)} / a_{kk}^{(k)} \qquad (a_{kk}^{(k)} \neq 0)$$

$$i,j = k+1(1)n, \quad k = 1(1)n-1.$$

The computation is clearly simplified if one or more of the quantities involved (except $a_{kk}^{(k)}$) is 0.

A sparse matrix technique is based on the following main principles:

A) Only the non-zero elements of matrix A are stored.

B) We attempt to perform only those computations which lead to changes, i.e. we only use formula (2.1) when $a_{ik}^{(k)} \neq 0$ and $a_{kj}^{(k)} \neq 0$.

C) The number of 'new elements' (fill-ins) is kept small. A new element is generated when $a_{ij}^{(k)} = 0$ and $a_{ij}^{(k+1)} \neq 0$.

Before we continue we shall introduce some notation and terminology.

By an element of matrix A we mean a non-zero element of the matrix. The rest of matrix A are called zeros and are treated as such.

n denotes the number of unknowns (columns).

m denotes the number of equations (rows).
(We shall only treat the case $m \neq n$ in chapter 5.)

NZ denotes the number of elements of matrix A.

NN is the length of the one-dimensional array A which is used to hold the elements ($NN \geq NZ$).

COUNT is the maximum number of elements (including fill-ins) kept in array A during the elimination process ($NN \geq COUNT$).

T is the drop tolerance (see the end of section 1.4).

We shall see that the use of sparse matrix techniques will change the contents of table 1.1 completely. More specifically, the computation time and the storage will not grow as fast with n, the storage needed for IR will not always be larger than for DS (because we introduce the drop tolerance), and the computation time will often be smaller for IR than for DS with the techniques which we are going to describe in the following chapters.

1.3 Test matrices

More often than not assertions and suggestions about sparse matrix techniques cannot be proved mathematically. We shall often have to rely on practical experiments to show that one technique is better than another - or to see under which circumstances it is better. For this purpose several classes of test matrices have been constructed, either as typical examples or generalizations of practically occurring matrices, or as nasty examples designed to make life difficult for sparse matrix programs.

We shall in this section introduce some of those test matrices which we are going to use throughout the text.

Test matrices of class D(n,c) are $n \times n$ matrices with 1 in the diagonal, three bands at the distance c above the diagonal (and reappearing cyclicly under it), and a 10×10 triangle of elements in the upper right-hand corner.

More specifically:

$a_{i,i} = 1, \quad i = 1(1)n;$

$a_{i,i+c} = i+1, \quad i = 1(1)n-c, \quad a_{i,i-n+c} = i+1, \quad i = n-c+1(1)n;$

$a_{i,i+c+1} = -i, \quad i = 1(1)n-c-1, \quad a_{i,i-n+c+1} = -i, \quad i = n-c(1)n;$

$a_{i,i+c+2} = 16, \quad i = 1(1)n-c-2, \quad a_{i,i-n+c+2} = 16 \quad i = n-c-1(1)n;$

$a_{i,n-11+i+j} = 100 \cdot j, \quad i = 1(1)11-j, \quad j = 1(1)10;$

for any $n \geq 14$ and $1 \leq c \leq n-13$.

By varying n and c we can obtain matrices of different sizes and sparsity patterns. In Fig. 1.2 we show the sparsity pattern of matrix D(20,5).

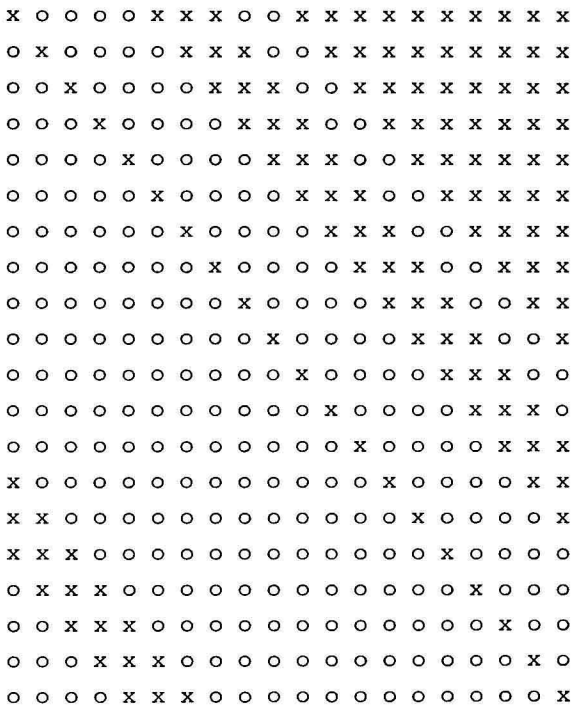

Fig. 1.2

Sparsity pattern of matrix D(20,5)

Test matrices of class E(n,c) are symmetric, positive definite, $n \times n$ matrices with 4 in the diagonal and -1 in the two sidediagonals and in two bands at the distance c from diagonal. These matrices are rather similar to matrices obtained from using the five-point formula in the discretization of elliptic partial differential equations.

$$
\begin{aligned}
& a_{ii} = 4, && i = 1(1)n; \\
(3.2) \qquad & a_{i,i+1} = a_{i+1,i} = -1, && i = 1(1)n-1; \\
& a_{i,i+c} = a_{i+c,i} = -1, && i = 1(1)n-c;
\end{aligned}
$$

where $n \geq 3$ and $2 \leq c \leq n-1$.

In Fig. 1.3 we show the matrix E(10,4)

$$\begin{pmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
-1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4
\end{pmatrix}$$

Fig. 1.3

The matrix E(10,4)

Test matrices of class F2(m,n,c,r,α) are $m \times n$ matrices which can be viewed as generalizations of the matrices of class D but with a lower left 10×10 triangle of elements added. r-1 is the width of a band located at a distance c from the main diagonal (and reappearing cyclicly under it). The elements are given by

$$a_{i,i-qn} = 1, \qquad i = 1\,(1)\,m;$$

$$a_{i,i-qn+c+s} = (-1)^s \cdot s \cdot i, \qquad s = 1\,(1)\,r-1, \quad i = 1\,(1)\,m;$$

where $q = 0,1,\ldots,\lceil m/n \rceil$ is chosen such that $1 \le i - qn \le n$ resp. $1 \le i - qn+c+s \le n$, and $\lceil m/n \rceil$ is the smallest integer greater than or equal to m/n;

$$a_{i,n-11+i+j} = j \cdot \alpha, \qquad i = 1\,(1)\,11-j, \quad j = 1\,(1)\,10;$$

$$a_{n-11+i+j,j} = i/\alpha, \qquad j = 1\,(1)\,11-i, \quad i = 1\,(1)\,10;$$

where $m \ge n \ge 22$, $11 \le c \le n-11$, $2 \le r \le \min(c-9, n-20)$, and $\alpha \ge 1$.

The smallest matrices of this class are thus F2(22, 22, 11, 2, α).

In Fig. 1.4 and 1.5 we show the sparsity pattern of matrices F2(26, 26, 12, 3, α) and F2(80, 30, 12, 4, α).

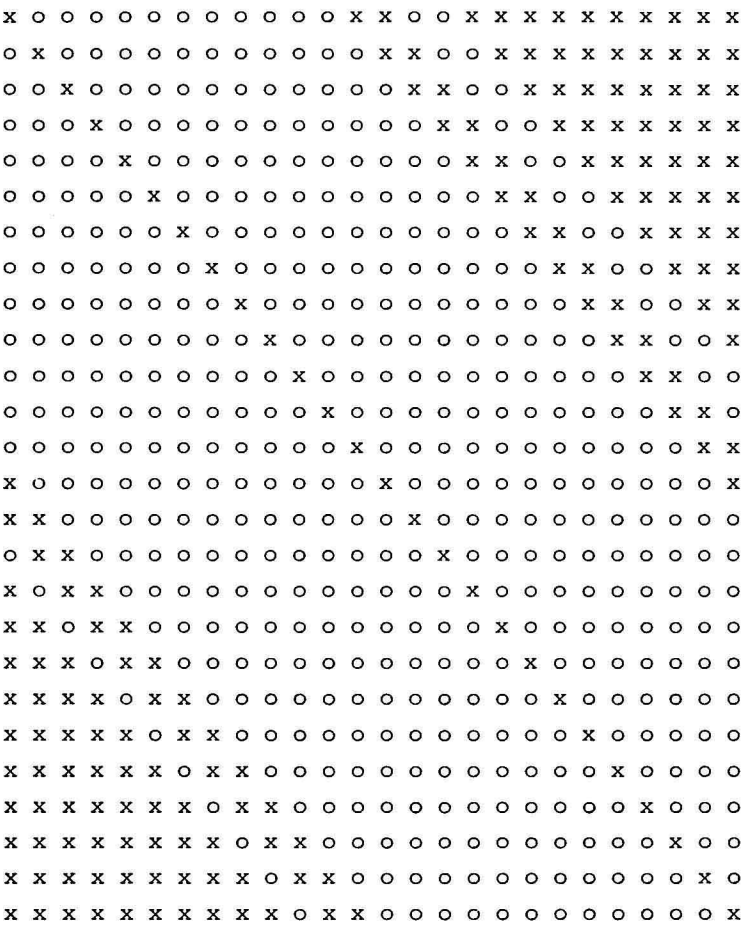

Fig. 1.4

Sparsity pattern of matrices F2(26, 26, 12, 3, α)

We emphasize here that by varying the parameters for test matrices of class F2 we can change the size n, the ratio m/n, the density NZ/n^2, the sparsity pattern, and the stability properties of the matrices and therefore carry out a rather systematic investigation of how the performance of a sparse matrix code depends on these quantities.