CHAPTER 0

Gaussian elimination for the solution of linear systems of equations

Gérard Meurant

Commissariat à l'Energie Atomique, Centre de Bruyeres–le–Chatel, BP12 91680 Bruyeres–le–Chatel, France

Contents

Enseigner la vie sans la vivre était le crime de l'innocence la plus détestable

Albert Cossery
Mendiants et orgueilleux, Paris 1955

This approximately translates into:
Teaching life without living it was the crime of the most hateful innocence

Foreword

I have tried to do my best and to provide a review of the main stream of research on Gaussian elimination that was done during the last thirty years. I hope I have been fair to everybody. Anyway there are probably some mistakes and some topics or some people could have been forgotten and I apologize for that.

Notice that some of the given references are not cited in the text. We give them for completeness. Some interested readers could like to learn more on some particular points and may want to have a look at these papers.

The method of successive elimination of the unknowns for linear systems is quite old. It is stated in [236] that in the second century the Chinese were already solving linear systems. Many works and papers have been published since then. From [335], we know that Lagrange was already using what we call Gaussian elimination before Gauss. However, this method is so well known under the name of Gaussian elimination that we choose to continue the tradition.

This work is dedicated to the fond memory of James H. Wilkinson (1919–1986). I had the privilege to know him quite well in 1982 when he was a professor in Stanford. He was certainly a great numerical analyst that has inspired many of us, particularly for his work about round off error analysis for Gaussian elimination but, more importantly, he was a very nice human being.

For an outline of his career and his work, see the interesting paper by Beresford Parlett in [283].

James H. Wilkinson, Stanford, 1982

This work is also dedicated to the memory of my father Georges Meurant (1921–1994).

1. Numerical solution of general linear systems

1. Introduction to Gaussian elimination

The problem we are concerned with is obtaining the numerical solution of a linear system

$$Ax = b \tag{1}$$

on a computer, where A is a square non singular matrix (i.e. $\det(A) \neq 0$) of order n, b is a given vector and x is the vector we are looking for. The entries of A and the elements of b and x can be real or complex numbers denoted respectively by $a_{i,j}$, b_i and x_i, $i, j = 1, \ldots, n$.

Of course, the solution x of eq.(1) is given by

$$x = A^{-1}b$$

where A^{-1} denotes the inverse of A. Unfortunately, in most cases, A^{-1} is not explicitly known, except for some special problems and/or for small values of n. But, as we all know, the solution can be expressed by Cramer's formulae (see for instance Gantmacher[1959]):

$$x_i = \frac{1}{\det(A)} \begin{vmatrix} a_{1,1} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & a_{2,i-1} & b_2 & a_{2,i+1} & \cdots & a_{2,n} \\ \vdots & \ldots & \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & b_n & a_{n,i+1} & \cdots & a_{n,n} \end{vmatrix}, \quad i = 1, \ldots, n \tag{2}$$

The computation of the solution x by eq.(2) requires the evaluation of $n+1$ determinants of order n. Clearly, this implies that this method will require more than $(n+1)!$ operations (multiplications and additions) to deliver the solution. This is far too much even for small values of n. It gives already more than $4\,10^7$ operations for $n = 10$. Today, it is well known and recognized that there are much better methods than Cramer's rule which is almost never used for solving linear systems.

There are two main classes of algorithms to obtain a solution to eq.(1): iterative methods and direct methods. Iterative methods define a sequence of approximations that are expected to be closer and closer in some given norm to the true solution, stopping the iterations by using some predefined criterion, obtaining a vector which is only an approximation of the solution. Direct methods try to compute the solution doing some combinations and modifications of the equations and after a finite number of floating point operations. Of course, as computer floating point operations are only done with a certain precision, the computed solution is generally different from the exact solution even with a direct method.

The most used direct methods for general matrices belong to a class collectively known as Gaussian elimination. There are many variations around the same basic idea and we will describe some of them in the next sections.

The basis of the method is easily explained on a small example and then, it can be extended to any value of n. The main idea is to successively eliminate the unknowns. Consider the following set of linear equations,

$$\begin{aligned} x_1 + x_2 &= 2 \\ 4x_1 + 5x_2 + 3x_3 &= 12 \\ 4x_1 + 6x_2 + 7x_3 &= 17 \end{aligned} \tag{3}$$

whose unique solution is $x_1 = x_2 = x_3 = 1$. Eqs.(3) can be written in matrix form as

$$Ax = \begin{pmatrix} 1 & 1 & 0 \\ 4 & 5 & 3 \\ 4 & 6 & 7 \end{pmatrix} x = \begin{pmatrix} 2 \\ 12 \\ 17 \end{pmatrix}. \tag{4}$$

The determinant of A is equal to 1. Therefore, as we said, there is a unique solution to eq.(3). The first equation of eq.(3) is used to express x_1 as a function of x_2,

$$x_1 = 2 - x_2. \tag{5}$$

Then, using eq.(4), x_1 is eliminated in the second and third equations giving a new (reduced) system of order 2, involving only x_2 and x_3,

$$\begin{aligned} x_2 + 3x_3 &= 4 \\ 2x_2 + 7x_3 &= 9 \end{aligned} \tag{6}$$

In the second step, x_2 is expressed as a function of x_3 using the first equation of eq.(6),

$$x_2 = 4 - 3x_3. \tag{7}$$

Using eq.(7) in the last equation of eq.(6), we get a linear system involving only x_3 whose solution is immediately given,

$$x_3 = 1.$$

Knowing the value of x_3, we can compute x_2 with eq.(7) which gives

$$x_2 = 1.$$

Finally, eq.(5) gives

$$x_3 = 1.$$

The previous elementary elimination method can be cast in a matrix framework. Eliminating x_1 from the second equation of eq.(3) amounts to do a linear combination of the first two equations. This is obtained by (left) multiplying A by the matrix

$$E_{2,1} = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Multiplying the system of eq.(4) by $E_{2,1}$ replaces the second equation by the second equation minus four times the first one, leaving the two others invariant,

$$E_{2,1}A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 4 & 6 & 7 \end{pmatrix}.$$

Elimination of x_1 from the last equation of eq.(3) is obtained by (left) multiplying by

$$E_{3,1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix}$$

and we obtain

$$E_{3,1}(E_{2,1}A) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 2 & 7 \end{pmatrix}.$$

Then, all the elements of the first column of A below the diagonal element have been reduced to 0. Remark that the subsystem that corresponds to the second and third rows and columns of this matrix is precisely the same as in eq.(6).

Elimination of x_2 from the third equation is obtained by (left) multiplying by

$$E_{3,2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}$$

and

$$E_{3,2}(E_{3,1}E_{2,1}A) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix}.$$

Therefore, when A is successively (left) multiplied by $E_{2,1}$, $E_{3,1}$ and $E_{3,2}$, it is reduced to an upper triangular form. Notice that $E_{3,1}$ and $E_{2,1}$ commute, as we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix}$$

Remark that the first column of the product is the "superposition" of the first columns of $E_{3,1}$ and $E_{2,1}$. Matrices $E_{2,1}$, $E_{3,1}$ and $E_{3,2}$ are non singular as their determinants are equal to 1. Therefore, the linear system of eq.(4) can be transformed into the equivalent system,

$$E_{3,2}E_{3,1}E_{2,1}Ax = E_{3,2}E_{3,1}E_{2,1}b, \tag{8}$$

where

$$b = \begin{pmatrix} 2 \\ 12 \\ 17 \end{pmatrix}.$$

Let

$$L^{-1} = E_{3,2}E_{3,1}E_{2,1} = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 4 & -2 & 1 \end{pmatrix}.$$

Eq.(8) reduces to

$$Ux = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} x = L^{-1}b = \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}. \tag{9}$$

As the matrix U of eq.(9) is upper triangular, this system which is equivalent to eq.(4), is easily solved starting with the last equation and moving backwards. This process is usually called backward (or back) substitution.

Besides solving the linear system, we have also seen that

$$L^{-1}A = U.$$

Therefore,

$$A = LU.$$

L^{-1} being lower triangular, its inverse L is also lower triangular. The matrix A has been factored into the product of a lower and an upper triangular matrices. The matrix L is easily obtained from L^{-1}. First, we verify that if

$$E_{3,1}E_{2,1} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix},$$

then

$$E_{2,1}^{-1}E_{3,1}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & 0 & 1 \end{pmatrix}.$$

Moreover, if

$$E_{3,2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \gamma & 1 \end{pmatrix},$$

then

$$E_{3,2}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\gamma & 1 \end{pmatrix},$$

and, hence

$$L = E_{2,1}^{-1} E_{3,1}^{-1} E_{3,2}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\gamma & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & -\gamma & 1 \end{pmatrix}.$$

L is obtained straightforwardly from the elementary matrices $E_{i,j}$ by simply changing the signs of the non zero off diagonal elements.

Of course, not all systems are as simple as the previous example. Remark that:
1) in this example, we were able to work only with integers. In real life problems, we have to work with the computer representations of real (or complex) numbers. We will look later on at the perturbation and stability problems related to this fact. This sometimes can make the computed solution to greatly differ from the exact solution.
2) not all systems have an LU factorization, even if they are non singular. Consider, for instance, a system with the following matrix,

$$A' = \begin{pmatrix} 1 & 1 & 0 \\ 4 & 4 & 3 \\ 4 & 6 & 7 \end{pmatrix},$$

which differs only from A by the (2,2) coefficient. If we use the same right hand side as in eq.(4), the first reduced system is

$$\begin{aligned} 0x_2 + 3x_3 &= 4 \\ 2x_2 + 7x_3 &= 9 \end{aligned}$$

The coefficient of x_2 in the first equation is 0. So, we cannot use this equation to express x_2 as a function of the other unknowns. Of course, the solution is to use another equation and this corresponds to applying a permutation to the matrix A'. In this simple example, we directly get $x_3 = 4/3$ and then, we compute x_2 using the last equation.

As we shall see, the general result for all non singular matrices is that there exists a permutation matrix P such that,

$$PA = LU,$$

where L is lower triangular and U is upper triangular.

2. Examples of problems

An important source of problems which require the solution of linear systems is the numerical solution of partial differential equations (PDEs). These equations (or systems of equations) modeling some physical phenomena are discretized with different kind of methods like finite differences, finite elements or spectral methods. Much in favor today are the finite element methods. However, finite differences or finite volume methods are still in use, particularly in computational fluid dynamics. All these discretization schemes lead to some large linear (and sometimes non linear) systems of equations. Generally, the matrices involved in these systems contain many zero entries. They are called sparse systems.

Of course, PDEs are not the only source of sparse matrices. Many examples are given in the book by Duff, Erisman and Reid ([1986]) like, for instance, power networks, problems in chemistry, etc...

However, solving large dense (with no or a few zero entries) linear systems still occur. An example is given by boundary integral methods. A problem that is of great importance today is solving the Maxwell equations for computing Radar Cross Sections giving the response of some objects to incoming radar waves. This problem is posed in an unbounded domain surrounding the object to study. Therefore, one way to get back to a bounded region is to write down an integral equation on the surface of the object which is discretized with finite elements and gives rise to a large dense linear system. Spectral methods also need to solve dense linear systems. Other examples of dense systems are given by A. Edelman ([1993]).

It is generally thought that solving linear systems is one of the most important operations in scientific computing as most physical problems lead to linear systems. At least, this is probably where most of the computer time is spent for scientific computing.

3. The general form of Gaussian elimination

In this Section, we generalize the method we have introduced on a small example in the first Section of this Chapter. First, we describe the method without permutations, exhibiting the necessary and sufficient conditions for a matrix to have an LU factorization. Then, we will introduce permutations to handle the general case.

Gaussian elimination without permutations

Let

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix},$$

and b be given. The problem to be solved is the linear system

$$Ax = b.$$

The first step of the algorithm is the elimination of x_1 into the equations 2 to n. This is done through $n-1$ steps. Suppose that $a_{1,1} \neq 0$, $a_{1,1}$ is called the first pivot. To eliminate x_1 from the second equation, we (left) multiply A by

$$E_{2,1} = \begin{pmatrix} 1 & & & & \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ 0 & 0 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}.$$

More generally, to eliminate x_1 from the ith equation, we (left) multiply by

$$E_{i,1} = \begin{pmatrix} 1 & & & & & & & \\ 0 & 1 & & & & & & \\ \vdots & & \ddots & & & & & \\ 0 & & & \ddots & & & & \\ -\frac{a_{i,1}}{a_{1,1}} & 0 & \dots & 0 & 1 & & & \\ 0 & & & & & \ddots & & \\ \vdots & & & & & & \ddots & \\ 0 & \dots & & & & \dots & 0 & 1 \end{pmatrix},$$

the non zero terms of the first column being in positions $(1,1)$ and $(i,1)$.

Lemma 3.1. *Let $j > i$, then*

$$E_{i,1}E_{j,1} = E_{j,1}E_{i,1} = \begin{pmatrix} 1 & & & & & & & & & & \\ 0 & 1 & & & & & & & & & \\ \vdots & & \ddots & & & & & & & & \\ 0 & & & \ddots & & & & & & & \\ -\frac{a_{i,1}}{a_{1,1}} & & & & \ddots & & & & & & \\ 0 & & & & & \ddots & & & & & \\ \vdots & & & & & & \ddots & & & & \\ 0 & & & & & & & \ddots & & & \\ -\frac{a_{j,1}}{a_{1,1}} & & & & & & & & \ddots & & \\ 0 & & & & & & & & & \ddots & \\ \vdots & & & & & & & & & & \ddots \\ 0 & & & & & & & & & & 1 \end{pmatrix}$$

Proof. Straightforward matrix multiply. □

Let $L_1 = E_{n,1}E_{n-1,1}\cdots E_{2,1}$ and $A_2 = L_1 A$. We denote the elements of A_2 by $a^{(2)}_{i,j}$. A_2 has the following structure,

$$A_2 = \begin{pmatrix} a_{1,1} & \mathrm{x} & \dots & \mathrm{x} \\ 0 & \mathrm{x} & \dots & \mathrm{x} \\ \vdots & \vdots & & \vdots \\ 0 & \mathrm{x} & \dots & \mathrm{x} \end{pmatrix},$$

where the x's correspond to (possibly) non zero elements that are defined in the following lemma.

Lemma 3.2.

$$a^{(2)}_{i,j} = a_{i,j} - \frac{a_{i,1}a_{1,j}}{a_{1,1}}, \quad 2 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant n$$
$$a^{(2)}_{1,j} = a_{1,j}, \quad 1 \leqslant j \leqslant n$$

Proof. This is just the result of the multiplication by L_1. □

Now, we describe the k–th step of the algorithm. Suppose we have zeroed the elements below the diagonal in the $k-1$ first columns and let A_k be the matrix that has been obtained,

$$A_k = \begin{pmatrix} a^{(k)}_{1,1} & \dots & \dots & \dots & \dots & a^{(k)}_{1,n} \\ & \ddots & & & & \vdots \\ & & a^{(k)}_{k,k} & \dots & \dots & a^{(k)}_{k,n} \\ & & a^{(k)}_{k+1,k} & \dots & \dots & a^{(k)}_{k+1,n} \\ & & \vdots & \vdots & \vdots & \vdots \\ & & a^{(k)}_{n,k} & \dots & \dots & a^{(k)}_{n,n} \end{pmatrix}$$

and $a^{(k)}_{k,k} \neq 0$, $a^{(k)}_{k,k}$ is called the k–th pivot.

For further references, we define a quantity g_A which is called the growth factor,

$$g_A = \frac{\max_{i,j,k} |a^{(k)}_{i,j}|}{\|A\|_\infty}.$$

Sometimes, g_A is defined as

$$g_A = \frac{\max_{i,j,k} |a^{(k)}_{i,j}|}{\max_{i,j} |a_{i,j}|}.$$

Let

$$E_{i,k} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & 0 & 1 & & & & & \\ & & \vdots & & \ddots & & & & \\ & & 0 & & & \ddots & & & \\ & & -\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} & & & & \ddots & & \\ & & 0 & & & & & \ddots & \\ & & \vdots & & & & & & \ddots \\ & & 0 & & & & & & & 1 \end{pmatrix}, \quad i > k$$

where the element $\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}$ is in row i and column k. The non diagonal elements that are not explicitly given are 0. Let $L_k = E_{n,k}E_{n-1,k}\cdots E_{k+1,k}$ and $A_{k+1} = L_k A_k$.

Lemma 3.3.

$$L_k = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & & \\ & & \vdots & & \ddots & \\ & & -\frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & & 1 \end{pmatrix}$$

and A_{k+1} has the following structure

$$A_{k+1} = \begin{pmatrix} a_{1,1}^{(k+1)} & \dots & \dots & \dots & \dots & a_{1,n}^{(k+1)} \\ & \ddots & & & & \vdots \\ & & a_{k,k}^{(k+1)} & \dots & \dots & a_{k,n}^{(k+1)} \\ & & 0 & a_{k+1,k+1}^{(k+1)} & \dots & a_{k+1,n}^{(k+1)} \\ & & \vdots & \vdots & & \vdots \\ & & 0 & a_{n,k+1}^{(k+1)} & \dots & a_{n,n}^{(k+1)} \end{pmatrix}$$

Proof. Straightforward. □

As for the first column, the elements of the j–th column are given by similar expressions.

Lemma 3.4.

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)} a_{k,j}^{(k)}}{a_{k,k}^{(k)}}, \quad k+1 \leqslant i \leqslant n, \quad k \leqslant j \leqslant n$$
$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)}, \quad 1 \leqslant i \leqslant k, \quad 1 \leqslant j \leqslant n$$
$$\text{and } k+1 \leqslant i \leqslant n, \quad 1 \leqslant j \leqslant k-1$$

Proof. This is the formula we obviously get when multiplying by L_k. □

Lemma 3.5. *L_k is non singular and*

$$L_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & \\ & & \vdots & & \ddots \\ & & \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & 1 \end{pmatrix}$$

Proof. Let

$$l_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} \\ \vdots \\ \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} \end{pmatrix}.$$

Clearly $L_k = (I - l_k e_k^{\mathrm{T}})$ where $e_k = (\,0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0\,)^{\mathrm{T}}$ with the 1 in the k–th position. Then, we verify that

$$L_k^{-1} = I + l_k e_k^{\mathrm{T}},$$

because,

$$\begin{aligned} L_k^{-1} L_k &= (I + l_k e_k^{\mathrm{T}})(I - l_k e_k^{\mathrm{T}}), \\ &= I - l_k e_k^{\mathrm{T}} l_k e_k^{\mathrm{T}}, \end{aligned}$$

and $e_k^{\mathrm{T}} l_k = 0$. □

The previous algorithm is summarized in the following lemma.

Lemma 3.6. *If for all k, $1 \leqslant k \leqslant n-1$, $a_{k,k}^{(k)} \neq 0$, then there exists a factorization*

$$A = LU,$$

where L is lower triangular and has a unit diagonal and U is upper triangular.

Proof. The elimination process goes as follows,

$$\begin{aligned} A_1 &= A, \\ A_2 &= L_1 A_1, \\ &\vdots \\ A_n &= L_{n-1} A_{n-1}. \end{aligned}$$

A_n is upper triangular and is therefore denoted by U. Hence,

$$L_{n-1} L_{n-2} \cdots L_1 A = U.$$

The matrices $L_i, 1 \leqslant i \leqslant n-1$ are non singular. Then,

$$A = (L_1^{-1} \cdots L_{n-1}^{-1}) U.$$

The product of unit lower triangular matrices is unit lower triangular. Hence,

$$L = L_1^{-1} \cdots L_{n-1}^{-1}$$

is a lower triangular matrix with a unit diagonal. Moreover, it is easy to see that

$$L = \begin{pmatrix} 1 & & & & & \\ \frac{a_{2,1}^{(1)}}{a_{1,1}^{(1)}} & \ddots & & & & \\ \vdots & \ddots & 1 & & & \\ \vdots & & \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & & \\ \vdots & & \vdots & \ddots & \ddots & \\ \frac{a_{n,1}^{(1)}}{a_{1,1}^{(1)}} & \cdots & \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & \cdots & \frac{a_{n,n-1}^{(n-1)}}{a_{n-1,n-1}^{(n-1)}} & 1 \end{pmatrix}.$$

As $L_i^{-1} L_{i+1}^{-1} = I + l_i e_i^{\mathrm{T}} + l_{i+1} e_{i+1}^{\mathrm{T}}$, we have

$$L = L_1^{-1} \cdots L_{n-1}^{-1} = I + l_1 e_1^{\mathrm{T}} + \cdots l_{n-1} e_{n-1}^{\mathrm{T}}$$

□

Lemma 3.7. *If the factorization $A = LU$ exists, it is unique.*

Proof. Suppose there exist two such factorizations, $A = L_1U_1 = L_2U_2$, then

$$L_2^{-1}L_1 = U_2U_1^{-1}.$$

The matrix on the left hand side is lower triangular with a unit diagonal and the matrix on the right hand side is upper triangular. Therefore,

$$L_2^{-1}L_1 = U_2U_1^{-1} = I$$

and the decomposition is unique. □

Now, we derive the conditions under which there exists an LU factorization.

Theorem 3.1. *A non singular matrix A has a unique LU factorization if and only if all the principal minors of A are non zero. That is*

$$A\begin{pmatrix} 1 & 2 & \dots & k \\ 1 & 2 & \dots & k \end{pmatrix} \neq 0, \quad k = 1, \dots, n$$

where

$$A\begin{pmatrix} i_1 & i_2 & \dots & i_p \\ k_1 & k_2 & \dots & k_p \end{pmatrix} = \begin{vmatrix} a_{i_1,k_1} & a_{i_1,k_2} & \dots & a_{i_1,k_p} \\ a_{i_2,k_1} & a_{i_2,k_2} & \dots & a_{i_2,k_p} \\ \vdots & \vdots & & \vdots \\ a_{i_p,k_1} & a_{i_p,k_2} & \dots & a_{i_p,k_p} \end{vmatrix}.$$

Moreover,

$$a_{k,k}^{(k)} = \frac{A\begin{pmatrix} 1 & 2 & \dots & k \\ 1 & 2 & \dots & k \end{pmatrix}}{A\begin{pmatrix} 1 & 2 & \dots & k-1 \\ 1 & 2 & \dots & k-1 \end{pmatrix}},$$

and

$$a_{k,j}^{(k)} = \frac{A\begin{pmatrix} 1 & 2 & \dots & k-1 & k \\ 1 & 2 & \dots & k-1 & j \end{pmatrix}}{A\begin{pmatrix} 1 & 2 & \dots & k-1 \\ 1 & 2 & \dots & k-1 \end{pmatrix}}, \quad j > k.$$

Proof. Suppose that there exists an LU factorization. We know from the proof of Lemma 3.6 that

$$A_{k+1} = L_k \dots L_1 A.$$

Therefore,

$$A = L_1^{-1} \dots L_k^{-1} A_{k+1},$$

and we have also

$$A = LU.$$

We can write these matrices in block form:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \quad L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}, \quad U = \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{pmatrix},$$

and from Lemma 3.6,

$$L_1^{-1} \dots L_k^{-1} = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I \end{pmatrix}, \quad A_{k+1} = \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & W_{2,2} \end{pmatrix},$$

where all the matrices in position $(1,1)$ are square of order k. By identification, we have

$$\begin{aligned} A_{1,1} &= L_{1,1} U_{1,1} \\ A_{2,2} &= L_{2,1} U_{1,2} + L_{2,2} U_{2,2} \\ A_{2,2} &= L_{2,1} U_{1,2} + W_{2,2} \end{aligned}$$

Therefore, $L_{1,1}U_{1,1}$ is the LU factorization of the leading principal submatrix of order k of A and $L_{2,2}U_{2,2}$ is the factorization of the matrix $W_{2,2}$ in the bottom right hand corner of A_{k+1} as we have $W_{2,2} = L_{2,2}U_{2,2}$.

Notice that $\det(A) = \det(A_{k+1})$. We have $\det(L_{1,1}) = 1$ and $\det(A_{1,1}) = \det(U_{1,1})$. $U_{1,1}$ being upper triangular, its determinant is equal to the product of the diagonal elements. Therefore, for all k,

$$\det(A_{1,1}) = a_{1,1}^{(1)} \cdots a_{k,k}^{(k)}.$$

This shows that the principal minors are non zero and the first formula. Now, we proceed in the same way as in Wilkinson ([1965]). We have

$$\begin{pmatrix} A_{1,1} \\ A_{2,2} \end{pmatrix} = \begin{pmatrix} L_{1,1} \\ L_{2,1} \end{pmatrix} U_{1,1}.$$

Let $A_{1,1}^i$ denotes the matrix formed by the first $k-1$ rows and the i–th row of the first k columns of A and let $L_{1,1}^i$ be defined in a similar way, then

$$A_{1,1}^i = L_{1,1}^i U_{1,1}.$$

It is easy to see that $L^i_{1,1}$ is triangular and

$$\det(L^i_{1,1}) = l_{i,k} = \frac{a^{(k)}_{i,k}}{a^{(k)}_{k,k}}.$$

Therefore, as $\det(A^i_{1,1}) = l_{i,k}\det(U_{1,1})$,

$$l_{i,k} = \frac{\det(A^i_{1,1})}{\det(A_{1,1})} = \frac{A\begin{pmatrix}1 & 2 & \cdots & k-1 & i\\ 1 & 2 & \cdots & k-1 & k\end{pmatrix}}{A\begin{pmatrix}1 & 2 & \cdots & k\\ 1 & 2 & \cdots & k\end{pmatrix}}.$$

Similarly, we have

$$\begin{pmatrix} A_{1,1} & A_{1,2}\end{pmatrix} = L_{1,1}\begin{pmatrix} U_{1,1} & U_{1,2}\end{pmatrix}.$$

This leads to

$$u_{k,i} = \frac{A\begin{pmatrix}1 & 2 & \cdots & k-1 & k\\ 1 & 2 & \cdots & k-1 & i\end{pmatrix}}{A\begin{pmatrix}1 & 2 & \cdots & k-1\\ 1 & 2 & \cdots & k-1\end{pmatrix}},$$

for the elements of U. The converse of the proof is easily derived by induction. □

Gaussian elimination with permutations (partial pivoting)

We now allow for possible zero pivots at each step. If the first pivot $a_{1,1}$ is zero, we permute the first row with a row p such that $a_{p,1} \neq 0$. This is always possible as $\det(A) \neq 0$. This is done by left multiplication of A by a permutation matrix P_1. P_1 is equal to the identity matrix except that rows 1 and p have been exchanged,

$$P_1 = \begin{pmatrix}
0 & 0 & \dots & 0 & 1 & 0 & \dots & 0\\
0 & 1 & 0 & \dots & 0 & \dots & & \vdots\\
\vdots & \ddots & \ddots & \ddots & \vdots & & & \\
0 & \dots & 0 & 1 & 0 & \dots & & \\
1 & 0 & \dots & 0 & 0 & 0 & \dots & \\
0 & \dots & & \dots & 0 & 1 & \ddots & \vdots\\
\vdots & & & & \vdots & \ddots & \ddots & 0\\
0 & \dots & & \dots & 0 & \dots & 0 & 1
\end{pmatrix}.$$

Notice that $P_1^{-1} = P_1$. On the permuted matrix, the algorithm is the same as without permutations. We construct L_1 such that $A_2 = L_1 P_1 A$.

Let us describe the k–th step. The main difference with what we saw before is that the pivot can possibly be zero. If this is the case, it is possible to find a row p such that $a_{p,k}^{(k)} \neq 0$. The reason for this being that $\det(A_k) = \det(A) \neq 0$ (as seen from the proof of Theorem 3.1) and the determinant $\det(A_k)$ is equal to the product of the first $k-1$ (non zero) pivots and the determinant of the matrix in the right hand bottom corner. Therefore, this matrix is non singular. In fact, we choose the non zero element which has the maximum modulus. This strategy of choosing the pivot in the k–th column is called partial pivoting.

Then, we multiply A_k by the corresponding permutation matrix P_k and apply the elimination algorithm,

$$A_{k+1} = L_k P_k A_k.$$

So, finally, we have

$$U = L_{n-1} P_{n-1} \cdots L_2 P_2 L_1 P_1 A.$$

It may seem that we have lost the good properties of the Gaussian algorithm as permutation matrices come in, even if some of them are equal to the identity matrix. However, we have the following result.

Lemma 3.8. *Let P_p be a permutation matrix representing the permutation between indices p and $q > p$ then, $\forall k < p$*

$$L_k P_p = P_p L_k'.$$

where L_k' is deduced from L_k by the permutation of entries in rows p and q in column k.

Proof. Recall that $P_p^{-1} = P_p$ and $L_k = I - l_k e_k^{\mathrm{T}}$,

$$L_k' = P_p L_k P_p = P_p (I - l_k e_k^{\mathrm{T}}) P_p = I - P_p l_k e_k^{\mathrm{T}} P_p$$

As $p > k$, $P_p e^k = e^k$, therefore

$$L_k' = I - l_k' e_k^{\mathrm{T}}$$

where $l_k' = P_p l_k$. Notice that the same is true for L_k^{-1} as $L_k^{-1} = I + l_k e_k^{\mathrm{T}}$. □

Now, we have the main result of this Section.

Theorem 3.2. *Let A be a non singular matrix. Then, there exists a permutation matrix P such that*

$$PA = LU$$

where L is lower triangular with a unit diagonal and U is upper triangular.

Proof. We have seen that

$$A = P_1 L_1^{-1} P_2 \cdots P_{n-1} L_{n-1}^{-1} U.$$

Then, we have from Lemma 3.8,

$$A = P_1 P_2 \cdots P_{n-1} (L_1'')^{-1} \cdots (L_{n-1}'')^{-1} U,$$

where

$$(L_k'')^{-1} = P_{n-1} \cdots P_{k+1} L_k^{-1} P_{k+1} \cdots P_{n-1},$$

corresponding to a permutation of the coefficients of column k. □

We notice that, by definition, in the factorization of Theorem 3.2, we have $|l_{i,j}| \leqslant 1$ as we have chosen the pivot as the element of maximum modulus.

Once the factorization has been obtained and the permutation matrix P is known (usually stored as a vector of indices as row permutations are not explicitly done during the factorization), the linear system

$$Ax = b,$$

is transformed into

$$PAx = LUx = Pb,$$

and is solved as

$$\begin{aligned} Ly &= Pb, \\ Ux &= y, \end{aligned}$$

by two triangular solves.

Gaussian elimination with other pivoting strategies

Pivoting strategies different from partial pivoting can be used. We can search for the pivot not only in the lower part of the k–th column but in all the current submatrix, the pivot being the element that realize $\max_{i,j} |a_{i,j}^{(k)}|$. This is called complete pivoting. Then, we have to introduce not only row permutations but also column permutations. This is obtained by multiplying from the right by a permutation matrix. Finally, we obtain two permutation matrices P and Q such that

$$PAQ = LU.$$

The solution of the linear system is obtained through

$$\begin{aligned} Ly &= Pb, \\ Uz &= y, \\ x &= Q^{\mathrm{T}} z. \end{aligned}$$

We will see later that complete pivoting have some advantages regarding stability. However, the cost of finding the pivot is much larger than for partial pivoting.

Another strategy called rook's pivoting has been introduced by Poole and Neal ([1992]). At the k–th step, the algorithm is the following. Let

$$r_1 = \min\{r|\ |a_{r,k}^{(k)}| \geqslant |a_{i,k}^{(k)}|,\ k \leqslant i \leqslant n\}$$

and

$$c_1 = \min\{c|\ |a_{r_1,c}^{(k)}| \geqslant |a_{r_1,j}^{(k)}|,\ k \leqslant j \leqslant n\}.$$

If $c_1 = k$, then $a_{r_1,k}^{(k)}$ is the selected pivot. If $c_1 \neq k$, column c_1 is searched for the entry with maximum modulus. Let

$$r_2 = \min\{r|\ |a_{r,c_1}^{(k)}| \geqslant |a_{i,c_1}^{(k)}|,\ k \leqslant i \leqslant n\}$$

and

$$c_2 = \min\{c|\ |a_{r_2,c}^{(k)}| \geqslant |a_{r_2,j}^{(k)}|,\ k \leqslant j \leqslant n\}$$

and so on.

Therefore, rook's pivoting searches for coefficients of maximum modulus in rows, then columns and then, rows and columns until an entry $a_{r,c}^{(k)}$ verifies $|a_{r,c}^{(k)}| \geqslant |a_{i,c}^{(k)}|,\ k \leqslant i \leqslant n$ and $|a_{r,c}^{(k)}| \geqslant |a_{r,j}^{(k)}|,\ k \leqslant j \leqslant n$. Numerical experiments using this strategy are given in Neale and Poole [1992].

Operation counts

We are interested in computing (approximately) the number of floating point operations that must be done to obtain the LU factorization of A.

For computing the k–th column of L, we need 1 division by the pivot and $n-k$ multiplications. To compute the updated matrix A_{k+1}, we need (after having computed the multipliers $-a_{i,k}^{(k)}/a_{k,k}^{(k)}$ which are the elements of L) $(n-k)^2$ additions and the same number of multiplications. To get the total number of operations, we sum up these numbers from 1 to $n-1$

$$\sum_{k=1}^{n-1}(n\ -\ k) = n(n-1) - \frac{1}{2}n(n-1) = \frac{1}{2}n(n-1)$$

$$\begin{aligned} \sum_{k=1}^{n-1}(n-k)^2 &= n^2 \sum_{k=1}^{n-1} 1 - 2n \sum_{k=1}^{n-1} k + \sum_{k=1}^{n-1} k^2 \\ &= n^2(n-1) - n^2(n-1) + \frac{(n-1)^3}{3} + \frac{(n-1)^2}{2} + \frac{n-1}{6} \\ &= \frac{1}{3}n(n-1)(n-\frac{1}{2}) \end{aligned}$$

Theorem 3.3. *To obtain the factorization*

$$PA = LU$$

of Theorem 3.1, we need

$$\frac{2}{3}n(n^2-1)$$

floating point operations (multiplies and adds) and $n-1$ divisions. The solutions of the triangular systems to obtain the solution x give $n(n-1)$ floating point operations for L and $n(n-1)+n$ for U.

4. Gaussian elimination for symmetric systems

The factorization of symmetric matrices is an important special case that we are going to consider in more details. Let us specialize the algorithm of Section 3 to the symmetric case. We are looking for a factorization

$$A = LDL^{\mathrm{T}},$$

where L is lower triangular with a unit diagonal and D is diagonal. There are several possibilities to obtain this factorization. We are going to study three different algorithms that will lead to several ways of programming the factorization.

The outer product algorithm

The first method to construct the LDL^{T} factorization is in the same way as we have seen for general systems, columns by columns. Suppose $a_{1,1} \neq 0$, let

$$L_1 = \begin{pmatrix} 1 & 0 \\ l_1 & I \end{pmatrix}, \quad D_1 = \begin{pmatrix} a_{1,1} & 0 \\ 0 & A_2 \end{pmatrix},$$

and

$$A = \begin{pmatrix} a_{1,1} & a_1^{\mathrm{T}} \\ a_1 & B_1 \end{pmatrix} = L_1 D_1 L_1^{\mathrm{T}}.$$

By identification, we obtain

$$\begin{aligned} l_1 &= \frac{a_1}{a_{1,1}} \\ A_2 &= B_1 - \frac{1}{a_{1,1}} a_1 a_1^{\mathrm{T}} = B_1 - a_{1,1} l_1 l_1^{\mathrm{T}} \end{aligned}$$

Notice that A_2 is a symmetric matrix. If we suppose that the $(1,1)$ element of A_2 is non zero, we can reiterate this and we write

$$A_2 = \begin{pmatrix} a_{2,2}^{(2)} & a_2^{\mathrm{T}} \\ a_2 & B_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_2 & I \end{pmatrix} \begin{pmatrix} a_{2,2}^{(2)} & 0 \\ 0 & A_3 \end{pmatrix} \begin{pmatrix} 1 & l_2^{\mathrm{T}} \\ 0 & I \end{pmatrix},$$

$$l_2 = \frac{a_2}{a_{2,2}^{(2)}},$$
$$A_3 = B_2 - \frac{1}{a_{2,2}^{(2)}} a_2 a_2^{\mathrm{T}} = B_2 - a_{2,2}^{(2)} l_2 l_2^{\mathrm{T}}.$$

We remark that if we denote,

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & l_2 & I \end{pmatrix},$$

then,

$$D_1 = \begin{pmatrix} a_{1,1} & 0 \\ 0 & A_2 \end{pmatrix} = L_2 \begin{pmatrix} a_{1,1} & 0 & 0 \\ 0 & a_{2,2}^{(2)} & 0 \\ 0 & 0 & A_3 \end{pmatrix} L_2^{\mathrm{T}} = L_2 D_2 L_2^{\mathrm{T}}.$$

Therefore, we have

$$A = L_1 L_2 D_2 L_2^{\mathrm{T}} L_1^{\mathrm{T}}.$$

We notice that

$$L_1 L_2 = \begin{pmatrix} 1 & 0 \\ l_1 & \begin{pmatrix} 1 & 0 \\ l_2 & I \end{pmatrix} \end{pmatrix}$$

If all the pivots are non zero, we can go on and at the last step, we have

$$A = A_1 = L_1 L_2 \cdots L_{n-1} D L_{n-1}^{\mathrm{T}} \cdots L_1^{\mathrm{T}} = L D L^{\mathrm{T}},$$

where L is unit lower triangular and D is diagonal.

There is a variant of this algorithm where a decomposition

$$A = \bar{L} \bar{D}^{-1} \bar{L}^{\mathrm{T}}$$

is obtained with $\bar{L}$ being lower triangular, $\bar{D}$ diagonal and $\mathrm{diag}(\bar{L}) = \mathrm{diag}(\bar{D})$. We can obtain this variant from the first algorithm by writing

$$A = L D L^{\mathrm{T}} = (LD) D^{-1} (D L^{\mathrm{T}}),$$

and $\bar{D} = D$, $\bar{L} = LD$.

In the previous algorithm, the matrix L is obtained column by column. This method is called the outer product form of the algorithm as, at each step, an outer product aa^{T} is involved.

The bordering algorithm

We partition the matrix A in a different way as

$$A = \begin{pmatrix} C_n & a_n \\ a_n^{\mathrm{T}} & a_{n,n} \end{pmatrix},$$

with obvious notations. Suppose that C_n has already been factored as

$$C_n = L_{n-1} D_{n-1} L_{n-1}^{\mathrm{T}},$$

with L_{n-1} unit lower triangular and D_{n-1} diagonal. We can write,

$$A = \begin{pmatrix} L_{n-1} & 0 \\ l_n^{\mathrm{T}} & 1 \end{pmatrix} \begin{pmatrix} D_{n-1} & 0 \\ 0 & d_{n,n} \end{pmatrix} \begin{pmatrix} L_{n-1}^{\mathrm{T}} & l_n \\ 0 & 1 \end{pmatrix}.$$

By identification,

$$\begin{aligned} l_n &= D_{n-1}^{-1} L_{n-1}^{-1} a_n, \\ d_{n,n} &= a_{n,n} - l_n^{\mathrm{T}} D_{n-1} l_n. \end{aligned}$$

Therefore, by induction we can start by the decomposition of the 1×1 matrix $a_{1,1}$, keeping on adding rows and obtaining at each step the factorization of an enlarged matrix. The only operation we have to perform at each step is solving a triangular system. To be able to proceed to the next step, we need the diagonal entries of D_n (i.e. $d_{n,n}$) to be non zero. For obvious reasons, this method is called the bordering form of the algorithm.

The inner product algorithm

Another way to compute the factorization is simply to write down the formulas for the matrix product

$$A = LDL^{\mathrm{T}}$$

Suppose $i \geqslant j$, we have

$$a_{i,j} = \sum_{k=1}^{j} l_{i,k} l_{j,k} d_{k,k}.$$

If we set $i = j$ in this formula as $l_{i,i} = 1$, we obtain

$$d_{j,j} = a_{j,j} - \sum_{k=1}^{j-1} (l_{j,k})^2 d_{k,k},$$

and for $i > j$,

$$l_{i,j} = \frac{1}{d_{j,j}} (a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k} d_{k,k}).$$

As, locally, we have to consider the product of the transpose of a vector times a vector, this method is called the inner (or scalar) product form of the algorithm.

Considering the number of floating point operations, these three variants all need about $\frac{1}{2}$ of the number of operations for the general algorithm, i.e. about $\frac{n^3}{6}$ multiplies and the same number of adds.

Programming the factorization algorithms

We are considering the different ways of coding the three algorithms that we have described for general (dense) symmetric matrices. Then, we will discuss the consequences of these different implementations, depending on the computer architecture.

The codes are written in Matlab–like language, although for clarity, we do not always use the most compact and efficient Matlab constructs.

We consider first the outer product algorithm. In this variant the matrix L is constructed column by column. At step k, the column k is constructed by multiplying by the inverse of the pivot and then, the columns at the right of column k are modified using the values of the entries of column k. This is summarized in figure 4.1.

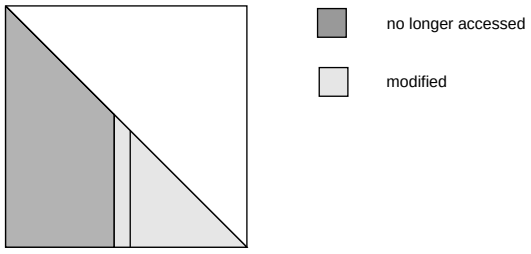

fig. 4.1: The outer product algorithm data layout

The modification of columns $k+1$ to n can be done by rows or by columns and this leads to the two codes given below.

For clarity, we store the matrix D in a vector denoted by d and L in a separate matrix although in practice, it can be stored in the lower triangular part of A (if A is not to be saved). `temp` is a temporary vector whose use can be avoided sometimes. We use it mainly for the clarity of the presentation. Notice that the codings are different from the ones in Dongarra, Gustavson, Karp [1984]. They are arranged such that in the main loop, `i` is a row index, `j` is a column index and `k` can be both.

The strictly lower triangular part of L is initialized to that of A by

```
function [l]=init(a)
[m,n]=size(a)
for i=1:n
  for j=1:i-1
    l(i,j)=a(i,j)
```

```
  end
end
```

Outer product kij algorithm

```
function [l,d]=kij(a)
[m,n]=size(a)
d=zeros(n,1)
temp=zeros(n,1)
d(1)=a(1,1)
l(1,1)=1
for k=1:n-1
  dki=1./d(k)
  for i=k+1:n
    temp(i)=l(i,k)*dki
  end
  for i=k+1:n
    for j=k+1:i
      l(i,j)=l(i,j)-temp(i)*l(j,k)
    end
  end
  for i=k+1:n
    l(i,k)=temp(i)
  end
  d(k+1)=l(k+1,k+1)
  l(k+1,k+1)=1.
end
```

To reflect the way the three loops are nested, this algorithm is called the kij form. We can get rid of the temporary vector `temp` by using the upper part of the matrix `l`, leading to the following code. However, we think the coding is clearer using `temp`.

```
function [l,d]=kijbis(a)
[m,n]=size(a)
d=zeros(n,1)
temp=zeros(n,1)
l=init(a)
d(1)=a(1,1)
l(1,1)=1
for k=1:n-1
  dki=1/d(k)
  for i=k+1:n
    l(k,i)=l(i,k)
    l(i,k)=l(i,k)*dki
  end
  for i=k+1:n
    for j=k+1:i
```

```
      l(i,j)=l(i,j)-l(i,k)*l(k,j)
    end
  end
  for i=k+1:n
    l(k,i)=0.
  end
  d(k+1)=l(k+1,k+1)
  l(k+1,k+1)=1;
end
```

Modifying by rows, we get

Outer product kji algorithm

```
function [l,d]=kji(a)
[m,n]=size(a)
d=zeros(n,1)
temp=zeros(n,1)
l=init(a)
d(1)=a(1,1)
l(1,1)=1
for k=1:n-1
  dki=1/d(k)
  for i=k+1:n
    temp(i)=l(i,k)*dki
  end
  for j=k+1:n
    for i=j:n
      l(i,j)=l(i,j)-temp(i)*l(j,k)
    end
  end
  for i=k+1:n
    l(i,k)=temp(i)
  end
  d(k+1)=l(k+1,k+1)
  l(k+1,k+1)=1.
end
```

Now, we turn ourselves to the bordering algorithm whose data accesses are summarized in figure 4.2.

For each row i, we have to solve a triangular system. There are two algorithms to do this. One is column oriented, the other is row oriented.

Bordering ijk algorithm

```
function [l,d]=ijk(a)
[m,n]=size(a)
d=zeros(n,1)
temp=zeros(n,1)
```

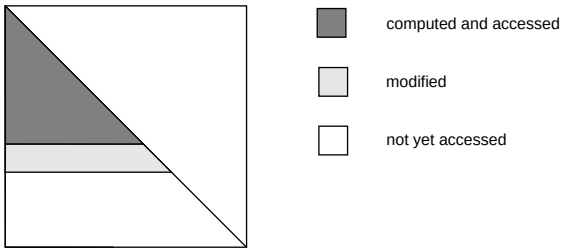

fig. 4.1: The bordering algorithm data layout

```
l=init(a)
d(1)=a(1,1)
l(1,1)=1.
for i=2:n
  for k=1:i
    temp(k)=a(i,k)
  end
  for j=1:i
    if j  ~= i
      l(i,j)=temp(j)/d(j)
    end
    for k=j+1:i
      temp(k)=temp(k)-l(k,j)*temp(j)
    end
  end
  d(i)=temp(i)
  l(i,i)=1
end
```

Notice there are too many divisions in the previous coding as the divisions are in a `k` loop. They can be avoided by storing the inverses of d as they are computed.

Bordering ikj algorithm

```
function [l,d]=ikj(a)
[m,n]=size(a)
d=zeros(n,1)
temp=zeros(n,1)
l=init(a)
d(1)=a(1,1)
l(1,1)=1
for i=2:n
  for k=1:i
    temp(k)=a(i,k)
  end
  for k=1:i
```

```
    for j=1:k-1
      temp(k)=temp(k)-temp(j)*l(k,j)
    end
    if k  ~= i
      l(i,k)=temp(k)/d(k)
    else
      d(i)=temp(i)
      l(i,i)=1
    end
  end
end
```

Finally, we consider the inner product algorithm. This algorithm is schematically depicted in figure 4.3.

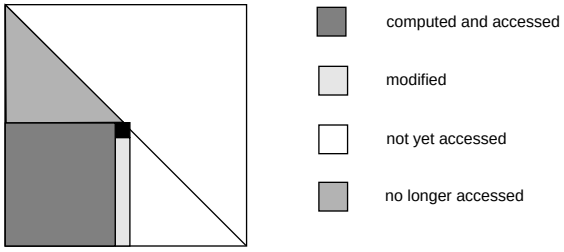

fig. 4.3: The inner product algorithm data layout

Inner product *jik* algorithm

```
function [l,d]=jik(a)
[m,n]=size(a)
l=init(a)
for j=1:n
  for k=1:j-1
    l(j,k)=l(j,k)/d(k)
  end
  d(j)=a(j,j)
  for k=1:j-1
    d(j)=d(j)-l(j,k) ^ 2*d(k)
  end
  for i=j+1:n
    for k=1:j-1
      l(i,j)=l(i,j)-l(i,k)*l(j,k)
    end
  end
  l(j,j)=1.
end
```

In the computation of $a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k} d_{k,k}$, one can compute $a_{i,j} - l_{i,k} l_{j,k} d_{k,k}$ for a fixed value of k looping on i provided that the divide by $d_{j,j}$ is done afterwards. Then, we obtain the following algorithm.

Inner product jki algorithm

```
function [l,d]=jki(a)
[m,n]=size(a)
l=init(a)
for j=1:n
  for k=1:j-1
    l(j,k)=l(j,k)/d(k)
  end
  d(j)=a(j,j)
  for k=1:j-1
    d(j)=d(j)-l(j,k) ^ 2*d(k)
  end
  for k=1:j-1
    for i=j+1:n
      l(i,j)=l(i,j)-l(i,k)*l(j,k)
    end
  end
  l(j,j)=1.
end
```

We have obtained six different ways to code the LDL^{T} factorization of a symmetric matrix A. The same can be done for the LU factorization of a non symmetric matrix. Of course, we are interested in knowing what is the best implementation, that is the one giving the smallest computing time. Unfortunately, this is dependent on the computer architecture and also on the languages that are used for coding and executing the algorithm. It also depends on the data structure that is chosen for storing L, as for today computers the performance depends mainly on the way the data is accessed in the computer memory.

Consider first L to be stored in a two dimensional array or in the lower triangular part of A. In the Fortran language, which is up to now the most widely used for scientific computing, two dimensional arrays are stored by columns, that is, consecutive elements in a column of an array have consecutive memory addresses. Therefore, to avoid memory access conflicts, it is much better to use algorithms that access data by columns. This could be different for other languages. For instance, in C two dimensional arrays are stored by rows. Moreover, in computers with data caches, it pays to do operations on data in consecutive memory locations. This increases the cache hit ratio as the data is moved into the cache by blocks of consecutive addresses.

The data access is by columns for algorithms kji and jki, by rows for ikj and jik and by rows and columns for kij and ijk. This favors algorithms kji and jki.

Form kji accesses the data by columns and the basic operation involved is a so

called SAXPY (for single precision a times x plus y) i.e.

$$y = y + \alpha x$$

where x and y are vectors (columns of L) and α is scalar. Notice that the vector y has to be stored after it is computed. This particular form was used in the famous LINPACK package, Bunch, Dongarra, Moler and Stewart [1979].

Form jki also accesses that data by columns and the basic operation is also a SAXPY. However, the same column (j) is successively accessed many times. This is called a generalized SAXPY or GAXPY. This can sometimes be exploited when coding in assembly language (one can keep the data in registers). These algorithms are analyzed for a vector computer in Dongarra, Gustavson and Karp [1984], their notations being slightly different. On this type of vector architectures, the GAXPY jki form is generally the best one.

If L is not stored in the lower triangular part of A, it is better to store it in a one dimensional array of dimension $n(n-1)/2$. Consecutive elements can be chosen by rows or columns. If consecutive elements are chosen by rows, it is better to use algorithms ikj and jik as the data accesses are going to be in consecutive addresses. kji and jki forms will be chosen if the data is stored by columns.

As an example, we consider the form jki, when the matrix is stored by columns. The matrix L is stored in a one dimensional array ll. To simplify a little bit the coding, we explicitly store the ones on the diagonal. Therefore, we have,

$$l(i,j) = ll(k),$$

with $k = n(j-1) + \frac{j-j^2}{2} + i$.

```
function [ll,d]=lljki(a)
[m,n]=size(a)
d=zeros(n,1)
ll=initll(a)
for j=1:n
  nj=n*(j-1)++(j-j ^ 2)/2
  for k=1:j-1
    nk=n*(k-1)+(k-k ^ 2)/2
    ll(nk+j)=ll(nk+j)/d(k)
  end
  d(j)=a(j,j)
  for k=1:j-1
    nk=n*(k-1)+(k-k ^ 2)/2
    d(j)=d(j)-ll(nk+j) ^ 2*d(k)
  end
  for k=1:j-1
    nk=n*(k-1)+(k-k ^ 2)/2
    for i=j+1:n
      ll(nj+i)=ll(nj+i)-ll(nk+i)*ll(nk+j)
```

```
    end
  end
  ll(nj+j)=1.
end
```

So far, we have supposed it was not necessary to do any pivoting for a symmetric system. We will study some particular cases where it can be shown that pivoting is not needed, at least to be able to run the algorithm to completion.

Positive definite systems

Let A be symmetric and positive definite. We are looking for an LDL^{T} factorization. The fundamental result is the following.

Lemma 4.1. *Let A be a symmetric positive definite matrix partitioned as*

$$A = \begin{pmatrix} A_{1,1} & A_{2,1}^{\mathrm{T}} \\ A_{2,1} & A_{2,2} \end{pmatrix}$$

where the blocks $A_{1,1}$ and $A_{2,2}$ are square. Then,

$$S_{2,2} = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{2,1}^{\mathrm{T}}$$

is symmetric and positive definite. $S_{2,2}$ is called the Schur complement (of $A_{2,2}$ in A).

Proof. This can be proved in many different ways. Perhaps, the simplest one is to consider A^{-1} and to compute the bottom right hand block of A^{-1}. Let

$$\begin{pmatrix} A_{1,1} & A_{2,1}^{\mathrm{T}} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Then,

$$\begin{aligned} & A_{1,1}x_1 + A_{2,1}^{\mathrm{T}}x_2 = b_1 \\ \Rightarrow \quad & x_1 = A_{1,1}^{-1}(b_1 - A_{2,1}^{\mathrm{T}}x_2) \end{aligned}$$

Therefore,

$$(A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{2,1}^{\mathrm{T}})x_2 = b_2 - A_{2,1}A_{1,1}^{-1}b_1$$

This means that the inverse of A can be written as

$$A^{-1} = \begin{pmatrix} X & Y \\ Z & S_{2,2} \end{pmatrix}.$$

A being positive definite, the diagonal blocks of A and A^{-1} are also positive definite, so $S_{2,2}$ is positive definite. □

If we look at the outer product algorithm of Section 4, we see that in the first step, $A_2 = B_1 - (1/a_{1,1})a_1 a_1^{\mathrm{T}}$ is a Schur complement, the matrix A being partitioned as

$$A = \begin{pmatrix} a_{1,1} & a_1^{\mathrm{T}} \\ a_1 & B_1 \end{pmatrix}.$$

Therefore, if A is positive definite, A_2 is also positive definite and the next pivot is non zero. The process can be continued until the last step. All the square matrices involved are positive definite and so the diagonal elements are strictly positive. All the pivots are non zero and the algorithm can go through without any pivoting. This is summarized in the following result.

Theorem 4.1. *A matrix A has a factorization $A = LDL^{\mathrm{T}}$, where L is a unit lower triangular matrix and D is a diagonal matrix with* $\mathrm{diag}(D) > 0$*, if and only if A is symmetric and positive definite.*

Proof. Lemma 4.1 and the discussion afterwards have shown that if A is positive definite, it can be factored as LDL^{T}. Reciprocally, if $A = LDL^{\mathrm{T}}$, then of course A is symmetric and if $x \neq 0$, then

$$x^{\mathrm{T}}Ax = x^{\mathrm{T}}LDL^{\mathrm{T}}x = y^{\mathrm{T}}Dy,$$

where $y = L^{\mathrm{T}}x \neq 0$. Notice that

$$y^{\mathrm{T}}Dy = \sum_{i=1}^{n} d_{i,i}y_i^2 > 0,$$

as the diagonal elements of D are strictly positive. □

In the factorization of Theorem 4.1, as the diagonal elements of D are strictly positive, we can introduce a diagonal matrix S such that $s_{i,i} = \sqrt{d_{i,i}},\ i = 1, \ldots, n$. Therefore, $S^2 = D$ and let $\bar{L} = LS$. Then,

$$A = LDL^{\mathrm{T}} = LSSL^{\mathrm{T}} = \bar{L}\bar{L}^{\mathrm{T}}.$$

Strictly speaking this is what is called the Cholesky factorization of A. However, this form of factorization is not much used today as the computation involves square roots. On modern computers, square roots are much slower than multiplies and adds and must be avoided whenever possible. Factorizations like LDL^{T} have sometimes been called root free Cholesky. Nowadays, the generic name Cholesky factorization is often used for any LDL^{T} factorization.

An interesting property of positive definite matrices is that there is no growth of the entries of the reduced matrices during the factorization.

Theorem 4.2. *Let A be symmetric positive definite. Consider the matrices $D_i,\ i = 1, \ldots, n$ of the outer product algorithm, then,*

$$\max_k(\max_{i,j} |(D_k)_{i,j}|) \leqslant \max_{i,j} |a_{i,j}| = \max_i(a_{i,i}).$$

We first prove a well known but useful Lemma.

Lemma 4.2. *Let A be a symmetric positive definite matrix. Then,*

$$\max_{i,j} |a_{i,j}| = \max_i (a_{i,i}).$$

Proof. It is obvious that the diagonal entries of A are positive. Suppose there is a couple i_0, j_0 such that $|a_{i_0,j_0}| > |a_{i,j}|, i_0 \neq j_0,\ \forall i, j$ different from i_0, j_0. There are two cases:

i) suppose $a_{i_0,j_0} > 0$, then let

$$x = (0 \ldots, 0, 1, 0, \ldots, 0, -1, 0, \ldots, 0)^{\mathrm{T}},$$

the 1 being in position i_0 and the -1 in position j_0. We have,

$$x^{\mathrm{T}} A x = a_{i_0,i_0} + a_{j_0,j_0} - 2a_{i_0,j_0} < 0.$$

Therefore, A is not positive definite which is a contradiction;

ii) suppose $a_{i_0,j_0} < 0$, we choose

$$x = (0 \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0)^{\mathrm{T}},$$

and get

$$x^{\mathrm{T}} A x = a_{i_0,i_0} + a_{j_0,j_0} + 2a_{i_0,j_0} < 0,$$

which, again, is a contradiction. □

Proof of Theorem 4.2. By Lemma 4.1, we already know that the matrices D_k are positive definite. Therefore, by Lemma 4.2, it is enough to consider the diagonal to find the maximum of the absolute values of the entries. It also suffices to consider the first step as the proof is the same for the other steps.

As the diagonal entries are positive, we have

$$\mathrm{diag}(A_2) \leqslant \mathrm{diag}(B_1).$$

Therefore, either $\max_i(a_{i,i}) = a_{1,1}$ and then, $\max_i(D_1)_{i,i} = a_{1,1}$ or the maximum is on the diagonal of B_1 and then,

$$\max_i (D_1)_{i,i} = \max(a_{1,1}, \max_i[\mathrm{diag}(A_2)_{i,i}]),$$

with $\max_i[\mathrm{diag}(A_2)_{i,i}] \leqslant \max_i[\mathrm{diag}(B_1)_{i,i}]$. In both cases,

$$\max_i (D_1)_{i,i} \leqslant \max_{i,j} |a_{i,j}|.$$

□

Indefinite systems

When factorizing an indefinite matrix, there can be some problems as the following example from Golub and Van Loan [1989] shows if ϵ is small,

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 0 \\ 0 & -1/\epsilon \end{pmatrix} \begin{pmatrix} 1 & 1/\epsilon \\ 0 & 1 \end{pmatrix},$$

The term $1/\epsilon$ can be very large and the factorization can be spoiled. One can use pivoting to avoid this. However, if symmetry is to be preserved, pivoting must be done from the diagonal and does not always solve the problems. Moreover, zero pivots can be the only alternative sometimes, as for instance in the following example

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

One possibility to overcome these difficulties is to use a method due to Aasen [1971]. Here, a factorization

$$PAP^{\mathrm{T}} = LTL^{\mathrm{T}},$$

where L is unit lower triangular and T is tridiagonal is obtained. This is not really Gaussian elimination, therefore, we will not give more details on this. Another idea, introduced in Bunch and Parlett [1971] and further developed in Bunch and Kaufman [1977] is to used diagonal pivoting with either 1×1 or 2×2 pivots. Suppose

$$P_1 A P_1^{\mathrm{T}} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^{\mathrm{T}} & A_{2,2} \end{pmatrix},$$

where $A_{1,1}$ is of order s with $s = 1$ or 2 and P_1 is a permutation matrix. Then, this matrix can be factored as

$$P_1 A P_1^{\mathrm{T}} = \begin{pmatrix} I_s & 0 \\ A_{1,2}^{\mathrm{T}} A_{1,1}^{-1} & I_{n-s} \end{pmatrix} \begin{pmatrix} A_{1,1} & 0 \\ 0 & A_{2,2} - A_{1,2}^{\mathrm{T}} A_{1,1}^{-1} A_{1,2} \end{pmatrix} \begin{pmatrix} I_s & A_{1,1}^{-1} A_{1,2} \\ 0 & I_{n-s} \end{pmatrix}.$$

The algorithm can go through provided that $A_{1,1}$ is non singular. It can be proved that A being non singular, it is always possible to find a non zero pivot ($s = 1$) or a non singular 2×2 block ($s = 2$). A strategy has been devised in Bunch and Kaufmann [1977], see also Golub and Van Loan [1989] to find the block pivots.

5. Gaussian elimination for H–matrices

There are other types of matrices (not necessarily symmetric) than positive definite matrices for which there is no necessity to use pivoting (at least to obtain a factorization without permutations). Let us first introduce a few definitions.

• A matrix A is reducible if and only if there exists a permutation matrix P such that

$$P^{-1}AP = \begin{pmatrix} D_1 & 0 \\ F & D_2 \end{pmatrix},$$

D_1 and D_2 being square matrices. A matrix that is not reducible is said to be irreducible.

• – A is (row) diagonally dominant if

$$|a_{i,i}| \geqslant \sum_{j=1, j\neq i}^{n} |a_{i,j}|, \ \forall i.$$

– A is (row) strictly diagonally dominant if

$$|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}|, \ \forall i.$$

– A is irreducibly (row) diagonally dominant if, (a) A is irreducible, (b) A is (row) diagonally dominant and (c) there exists an i_0 such that

$$|a_{i_0,i_0}| > \sum_{j=1, j\neq i_0}^{n} |a_{i_0,j}|.$$

Notice that similar definitions hold for column oriented diagonal dominance. Matrices which are symmetric and strictly diagonally dominant or irreducibly diagonally dominant are positive definite. Their cases are covered in Section 4.

• A is an M–matrix if and only if $a_{i,j} \leqslant 0$ for $i \neq j$ and $A^{-1} \geqslant 0$.

If A is a symmetric M–matrix then, A is positive definite. Once again, this is covered in Section 4.

Let A be a matrix, define $M(A)$ as the matrix having entries $m_{i,j}$ such that

$$m_{i,i} = |a_{i,i}|, \quad m_{i,j} = -|a_{i,j}|, \ \forall i, j, \ i \neq j.$$

• A is an H–matrix if and only if $M(A)$ is an M–matrix.

The previous definitions for diagonally dominant matrices can be slightly generalized. This will lead to a characterization of H–matrices.

• A matrix A is generalized (row) diagonally dominant if there exists a vector d with $d_i > 0, \forall i$ such that

$$|a_{i,i}|d_i \geqslant \sum_{j=1, j\neq i}^{n} |a_{i,j}|d_j, \ \forall i.$$

A is generalized (row) strictly diagonally dominant if

$$|a_{i,i}|d_i > \sum_{j=1,j\neq i}^{n} |a_{i,j}|d_j, \ \forall i.$$

Clearly, this means that if we denote

$$D = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix},$$

then, AD is (row) diagonally dominant or (row) strictly diagonally dominant. The same is true of $D^{-1}AD$.

We have the following results that we state without proof, see Berman and Plemmons [1979].

Theorem 5.1. *A is an M–matrix if and only if $a_{i,j} \leqslant 0, \forall i \neq j$ and A is generalized (row or column) strictly diagonally dominant.*

Theorem 5.2. *A is an H–matrix if and only if A is generalized (row or column) strictly diagonally dominant.*

It is obvious that strictly diagonally dominant, irreducibly diagonally dominant and M–matrices are H–matrices. For these types of matrices, the important fact is that at each step of Gaussian elimination, the property in question is maintained, that is $A_k, \forall k$ possesses the same property as A.

Let us first consider A being diagonally dominant.

Theorem 5.3. *If A is (row or column) diagonally dominant, then*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular.

Proof. Suppose A is (row) diagonally dominant. Then, $a_{1,1} \neq 0$, otherwise all the elements in the first row are 0 and A is singular. We shall prove that A_2 is also (row) diagonally dominant and then, the proof will go on by induction. The case of the first row is already handled. Now, we have

$$a_{i,j}^{(2)} = a_{i,j} - \frac{a_{i,1}a_{1,j}}{a_{1,1}}, \ 2 \leqslant i \leqslant n, \ 2 \leqslant j \leqslant n, \quad a_{i,1}^{(2)} = 0, \ 2 \leqslant i \leqslant n,$$

$$\sum_{j,j\neq i} |a_{i,j}^{(2)}| = \sum_{j,j\neq i,j\neq 1} |a_{i,j}^{(2)}| \leqslant \sum_{j,j\neq i,j\neq 1} |a_{i,j}| + \left|\frac{a_{i,1}}{a_{1,1}}\right| \sum_{j,j\neq i,j\neq 1} |a_{1,j}|.$$

But,

$$|a_{1,1}| \geqslant \sum_{j,j\neq i,j\neq 1} |a_{1,j}| + |a_{1,i}|.$$

Therefore,

$$\begin{aligned}
\sum_{j,j\neq i} |a_{i,j}^{(2)}| &\leqslant \sum_{j,j\neq i,j\neq 1} |a_{i,j}| + \left|\frac{a_{i,1}}{a_{1,1}}\right|(|a_{1,1}| - |a_{1,i}|),\\
&\leqslant \sum_{j,j\neq i} |a_{i,j}| - \frac{|a_{i,1}a_{1,i}|}{|a_{1,1}|},\\
&\leqslant |a_{i,i}| - \frac{|a_{i,1}a_{1,i}|}{|a_{1,1}|},\\
&\leqslant \left|a_{i,i} - \frac{a_{i,1}a_{1,i}}{a_{1,1}}\right| = |a_{i,i}^{(2)}|.
\end{aligned}$$

This shows that all the pivots are non zero and the computations can go through. If A is column diagonally dominant, the same proof can be done with A^{T}.

From the above discussion, it is obvious that $|l_{i,j}| \leqslant 1$. □

We then consider M–matrices. The following result has been proved by Fiedler and Ptàk [1962].

Theorem 5.4. *If A is an M–matrix, then*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular.

Proof. The proof can be found for instance in Fiedler's book [1986] or in Bermann and Plemmons [1979]. □

This can be proved through the following Lemma.

Lemma 5.1. *Let A be an M–matrix written in block form as*

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix},$$

where B and C are square matrices, then the Schur complement $S = C - EB^{-1}F$ is an M–matrix.

Proof. It is obvious that the principal submatrices of an M–matrix are M–matrices. Therefore, B is an M–matrix and $B^{-1} > 0$. As, by definition, the entries of E and

F are non positive, the entries of $EB^{-1}F$ are non negative. Therefore, the non diagonal entries of S are non positive.

Now, as A is an M–matrix, we know there is a diagonal matrix D with strictly positive diagonal entries such that AD is (row) strictly diagonally dominant.

Let

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix},$$

AD being (row) strictly diagonally dominant means that if $e = (1 \ \ldots \ 1)^{\mathrm{T}}$, then $ADe > 0$. But,

$$AD = \begin{pmatrix} BD_1 & FD_2 \\ ED_1 & CD_2 \end{pmatrix}$$

and let $e = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$. The Schur complement of AD is (row) strictly diagonally dominant (Concus, Golub and Meurant [1985]). This means that,

$$0 < [CD_2 - ED_1(BD_1)^{-1}FD_2]e_2 = SD_2e_2.$$

This shows that S is (row) generalized strictly diagonally dominant. Hence, S is an M–matrix. □

Proof of Theorem 5.4. We apply readily Lemma 5.1 that shows that starting from an M–matrix, the reduced matrices that are obtained at each step are M–matrices. Moreover, L and U are M–matrices. □

We now consider H–matrices. Let B be an M–matrix, we define

$$\Omega_B = \{A | B \leqslant M(A)\}.$$

That is

$$|a_{i,i}| \geqslant b_{i,i}, \ 1 \leqslant i \leqslant n,$$
$$|a_{i,j}| \leqslant |b_{i,j}|, \ i \neq j, \ 1 \leqslant i, j \leqslant n.$$

This means that A is at least as diagonally dominant as B.

Lemma 5.2. *Let B be an M–matrix. Each $A \in \Omega_B$ is (row) generalized strictly diagonally dominant.*

Proof. There exists a diagonal matrix D (with $\mathrm{diag}(D) > 0$) such that BD is strictly diagonally dominant and let $A \in \Omega_B$, we have

$$BD \leqslant M(A)D = M(AD).$$

Therefore,

$$0 < BDe \leqslant M(AD)e,$$

which implies that AD is (row) strictly diagonally dominant. □

Theorem 5.5. *Let B be an M–matrix. For each $A \in \Omega_B$,*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular. In particular, for every H-matrix, there exists an LU factorization.

Proof. We have seen in the proof of Lemma 5.2 that AD is (row) strictly diagonally dominant. Then, by Theorem 5.3, there exist $\bar{L}$ and $\bar{U}$, lower and upper triangular matrices such that

$$AD = \bar{L}\bar{U}.$$

We have,

$$A = \bar{L}\bar{U}D^{-1},$$

and the result follows. □

In Funderlic, Neumann, Plemmons [1982], it is proved that if $A = LU$ and $B = L'U'$, then

$$\begin{aligned}
&|l_{i,j}| \leqslant |l'_{i,j}|,\ 1 \leqslant i,j \leqslant n,\\
&|u_{i,j}| \leqslant |u'_{i,j}|,\ i \neq j,\ 1 \leqslant i,j \leqslant n,\\
&|u_{i,i}| \geqslant |u'_{i,i},|\ 1 \leqslant i \leqslant n,
\end{aligned}$$

and let

$$\beta_D = \frac{\max_i(D_{i,i})}{\min_i(D_{i,i})}.$$

Then,

$$\begin{aligned}
&|l_{i,j}| \leqslant \beta_D,\\
&|u_{i,j}| \leqslant 2\beta_D \max_i |a_{i,i}|.
\end{aligned}$$

This gives

$$g_A \leqslant 2\beta_D,$$

and for an M–matrix, Funderlic, Neumann and Plemmons [1982] proved that $g_A \leqslant \beta_D$. The proofs of these inequalities are easily obtained by induction.

The proof that an H–matrix possesses an LU factorization can also be established by showing, as in Bermann and Plemmons [1979], that all the leading principal minors are non singular. Similar results for the case where A is singular have been studied in Varga and Cai [1981], Funderlic and Plemmons [1981] and Funderlic, Neumann and Plemmons [1982].

For the case of H–matrices, we have seen that the growth factor is bounded

$$g_A \leqslant 2\beta_D.$$

It has been shown also that any symmetric permutation of A has an LU factorization. However, even if g_A is bounded, it can be large. Consider, for instance, the following example (Ahac, Buoni, Oleski, [1988]),

$$A_x = \begin{pmatrix} 2 & 0 & -x \\ -x & x & -1 \\ 0 & -1 & x \end{pmatrix}, \ x > 0.$$

The matrix A_x is an M–matrix if $x > \sqrt{2}$ and

$$A_x = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{x}{2} & 1 & 0 \\ 0 & -\frac{1}{x} & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & -x \\ 0 & x & -\frac{x^2}{2} - 1 \\ 0 & 0 & \frac{x}{2} - \frac{1}{x} \end{pmatrix}.$$

If x is large, the growth factor is large (in fact $O(x)$). We will see in the next sections that is bad for the stability of the algorithm. This can be avoided by using some form of symmetric pivoting. For M–matrices, Ahac and Oleski [1986] chose (in the reduced matrix) the column which has the largest column sum.

In the example, we choose the second column. The permuted matrix is

$$A' = \begin{pmatrix} x & -1 & -x \\ -1 & x & 0 \\ -x & 0 & 2 \end{pmatrix},$$

and

$$A' = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{x} & 1 & 0 \\ -1 & \frac{x}{1-x^2} & 1 \end{pmatrix} \begin{pmatrix} x & -1 & -x \\ 0 & x - \frac{1}{x} & 1 \\ 0 & 0 & 2 - x + \frac{x}{x^2-1} \end{pmatrix}.$$

The growth factor of A' is bounded independently of x. In Ahac and Oleski [1986], it is proved that $g_A \leqslant n$ for M–matrices. This result is extended to H–matrices in Ahac, Buoni, Oleski [1988].

6. Block methods

Block methods are obtained by partitioning the matrix A into blocks (submatrices). Consider, for instance, a 3×3 block partitioning. Then, A is written as

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}.$$

The matrices $A_{i,i}$ are square of order n_i, $1 \leqslant n_i \leqslant n$. Then, a block LU factorization can be obtained. There are several ways to do that. One is the following,

$$A = \begin{pmatrix} I & & \\ L_{2,1} & I & \\ L_{3,1} & L_{3,2} & I \end{pmatrix} \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ & U_{2,2} & U_{2,3} \\ & & U_{3,3} \end{pmatrix}.$$

The factorization proceeds in the same way as for the standard (point) factorization. Therefore, to understand the algorithms, it is enough to look at a 2×2 case,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L_{2,1} & I \end{pmatrix} \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & S \end{pmatrix}.$$

Then,

$$U_{1,1} = A_{1,1}.$$

$L_{2,1}$ is obtained by solving $L_{2,1}A_{1,1} = A_{2,1}$ and finally

$$S = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2} = A_{2,2} - L_{2,1}A_{1,2}.$$

Then, the algorithm is repeated on S. Here, no pivoting takes place. The stability is investigated in Demmel, Higham and Schreiber [1992]. Block LU factorization (without pivoting) is unstable in general, although it has been found to be stable for matrices that are block diagonally dominant by columns, that is

$$\|A_{j,j}^{-1}\|^{-1} \geqslant \sum_{i \neq j} \|A_{i,j}\|.$$

7. Particular systems

Tridiagonal matrices

Tridiagonal matrices are particularly easy to handle. Let T be a symmetric tridiagonal matrix,

$$T = \begin{pmatrix} a_1 & -b_2 & & & \\ -b_2 & a_2 & -b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -b_{n-1} & a_{n-1} & -b_n \\ & & & -b_n & a_n \end{pmatrix}.$$

We will also look at a particular case of a tridiagonal Toeplitz matrix,

$$T_a = \begin{pmatrix} a & -1 & & & \\ -1 & a & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & a & -1 \\ & & & -1 & a \end{pmatrix}.$$

The Cholesky factorization of T is easily obtained,

$$T = LD_L^{-1}L^{\mathrm{T}},$$

$$L = \begin{pmatrix} \delta_1 & & & & \\ -b_2 & \delta_2 & & & \\ & \ddots & \ddots & & \\ & & -b_{n-1} & \delta_{n-1} & \\ & & & -b_n & \delta_n \end{pmatrix}, \quad D_L = \begin{pmatrix} \delta_1 & & & & \\ & \delta_2 & & & \\ & & \ddots & & \\ & & & \delta_{n-1} & \\ & & & & \delta_n \end{pmatrix}.$$

By inspection, we have

$$\delta_1 = a_1, \quad \delta_i = a_i - \frac{b_i^2}{\delta_{i-1}}, \ i = 2, \ldots, n.$$

This involves only $n-1$ additions, multiplications and divisions. Extensions are easily obtained to non symmetric tridiagonal matrices as long as pivoting is not needed. If $T = T_a$, then we have

$$\delta_1 = a, \quad \delta_i = a - \frac{1}{\delta_{i-1}}, \ i = 2, \ldots, n$$

and the explicit solution of this recurrence is known, see, for instance, Meurant [1992],

$$\delta_i = \frac{r_+^{i+1} - r_-^{i+1}}{r_+^i - r_-^i},$$

where,

$$r_\pm = \frac{a \pm \sqrt{a^2 - 4}}{2},$$

are the two solutions of the quadratic equation $r^2 - ar + 1 = 0$ if $a \neq 2$. If $a = 2$, then $\delta_i = (i+1)/i$. Of course, there are more efficient methods to deal with Toeplitz matrices, see Golub and Van Loan [1989].

The Cholesky factorization of tridiagonal matrices has been used by Meurant [1992] to characterize the inverse of such matrices.

Block tridiagonal matrices

The previous method is easily extended to block tridiagonal symmetric matrices. Let

$$A = \begin{pmatrix} D_1 & -A_2^{\mathrm{T}} & & & \\ -A_2 & D_2 & -A_3^{\mathrm{T}} & & \\ & \ddots & \ddots & \ddots & \\ & & -A_{n-1} & D_{n-1} & -A_n^{\mathrm{T}} \\ & & & -A_n & D_n \end{pmatrix},$$

each block being of order n. Denote by L the block lower triangular part of A then, if such a factorization exists we have,

$$A = (\Delta + L)\Delta^{-1}(\Delta + L^{\mathrm{T}}),$$

where Δ is a block diagonal matrix whose diagonal blocks are denoted by Δ_i. By inspection, we have

$$\Delta_1 = D_1, \quad \Delta_i = D_i - A_i(\Delta_{i-1})^{-1}A_i^{\mathrm{T}},\ i = 2, \ldots, n$$

Therefore, obtaining this block factorization involves only solving (small) linear systems with matrices Δ_i and several right hand sides. Notice that whatever are the structures of the matrices D_i, the Δ_is are dense matrices.

Again, this block factorization can be used (Meurant [1992]) to characterize the inverse of block tridiagonal matrices.

2. Error analysis

8. Round off error analysis

The algorithms that we have seen so far are supposed to give the LU factorization of a given matrix with real or complex coefficients after $n-1$ steps (and possibly some permutations). These algorithms are programmed using some languages like Fortran, C or Matlab and the codes that are produced are run on different computers which are ranging from PCs to supercomputers.

Unfortunately, all the parts in a computer are finite. In particular, registers and memory words designed to store the data and the intermediate and final results have a finite length or capacity and cannot store all real numbers. Moreover, when computations are performed on a computer, each arithmetic operation $(+, -, *, /)$ is generally affected by round off errors.

The subject of round off error analysis is to try to understand what are the effects of these limitations on the result of solving a problem, in our case, using Gaussian elimination. Before going into these problems, we must define the floating point arithmetic model we are using.

Floating point arithmetic model

Here, we follow the expositions of Forsythe and Moler [1967] and Golub and Van Loan [1989]. The numbers that can be represented in the computer are a (finite) subset of the real line and are denoted by F. This set is characterized by four integers: the base β, the number t of base–β digits in the fractional part (also called the mantissa) and the exponent range $[e_L, e_U]$. Then, (normalized) numbers in F consists of all real numbers f of the form

$$f = \pm .d_1 d_2 \cdots d_t \ \beta^e, \quad 0 \leqslant d_i < \beta, \ d_1 \neq 0, \ e_L \leqslant e \leqslant e_U,$$

where e is an integer, to which we add zero and a representation for results whose absolute value will be smaller (resp. larger) than the smallest (resp. largest) absolute value of a non zero number in F.

For a non zero $f \in F$, we have

$$m = \beta^{e_L - 1} \leqslant |f| \leqslant M = \beta^{e_U}(1 - \beta^{-t}).$$

On the real number line, the elements of F are not equally spaced (see figure 8.1 that shows the elements of F for $\beta = 2, t = 3, e_L = 0, e_U = 2$).

A real number x that we would like to represent is approximated by a number $fl(x)$ that can be defined as an operator from

$$G = \{x \in R, m \leqslant |x| \leqslant M\} \cup \{0\},$$

into F, by

$$fl(x) = \begin{cases} \text{nearest } x_{\mathrm{R}} \in F \text{ to } x \text{ if rounded arithmetic is used,} \\ \text{nearest } x_{\mathrm{C}} \in F \text{ s. t. } |x_{\mathrm{C}}| \leqslant |x| \text{ if chopped arithmetic is used.} \end{cases}$$

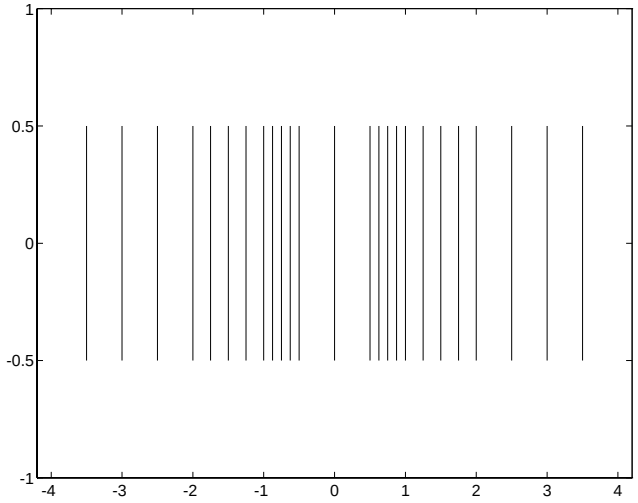

fig. 8.1: Elements of F for $\beta = 2, t = 3, e_L = 0, e_U = 2$

Today, most computers use $\beta = 2$, although there had been in the past computers with $\beta = 8$ or 16. In the examples, we will use $\beta = 10$ as this is more familiar to most people. Typical values of the other parameters for some (1996) computers are given in the following table.

Examples of floating point formats

	t	e_L	e_U
CRAY single	48	-8192	8191
IEEE single	24	-125	128
IEEE double	53	-1021	1024

For more details on floating point arithmetic particularly IEEE formats, see Golberg [1991]. The following results are proved in Forsythe and Moler [1967].

Theorem 8.1. *If $x \in G$, then*

$$x_{\mathrm{R}} = x(1 + \epsilon_{\mathrm{R}}),\ |\epsilon_{\mathrm{R}}| \leqslant \frac{1}{2}\beta^{1-t},$$
$$x_{\mathrm{C}} = x(1 + \epsilon_{\mathrm{C}}),\ |\epsilon_{\mathrm{C}}| \leqslant \beta^{1-t}.$$

Next, we have to define the operations on elements of F. Let $\bullet$ be one of the four operations $+, -, *, /$ and $\odot$ its implementation on a computer. We say that a computer arithmetic is correct if

$$x, y \in F,\ fl(x \bullet y) = x \odot y.$$

Not all arithmetics are correct. The one used on CRAYs Y–MP and C90 is not correct. The IEEE norm defines a correct arithmetic.

We define the unit round off u as,

$$u = \begin{cases} u_{\mathrm{R}} = \frac{1}{2}\beta^{1-t} & \text{for rounded arithmetic,} \\ u_{\mathrm{C}} = \beta^{1-t} & \text{for chopped arithmetic.} \end{cases}$$

For IEEE single precision arithmetic, the unit round off (rounded arithmetic) is $u_{\mathrm{R}} = 5.9605\ 10^{-8}$ and $u_{\mathrm{R}} = 1.1102\ 10^{-16}$ for double precision. We have,

$$fl(x) = x(1+\epsilon),\ |\epsilon| \leqslant u.$$

We have the following result,

$$fl(x \bullet y) = (x \bullet y)(1+\epsilon),\ |\epsilon| \leqslant u, \tag{10}$$

and then

$$\frac{|fl(x \bullet y) - (x \bullet y)|}{|x \bullet y|} \leqslant u.$$

This shows there is a small relative error associated with each operation (provided t is large enough). Note that (10) is not verified by CRAY arithmetic because of the lack of guard digits. Relaxed assumptions have to be made. Fortunately however, most of the results carry over to this case.

From this last result we can construct some other error bounds. We use the following Lemma from Forsythe and Moler [1967].

Lemma 8.1. *If* $0 \leqslant u < 1$ *and* $n \in N$,

$$1 - nu \leqslant (1-u)^n.$$

If $0 \leqslant nu \leqslant 0.01$,

$$(1+u)^n \leqslant 1 + 1.01nu.$$

The first statement comes from Taylor's formula and the second one is because we have

$$e^x \leqslant 1 + 1.01x \text{ for } 0 \leqslant x \leqslant 0.01$$

Notice that the largest n that satisfies the inequality $0 \leqslant nu \leqslant 0.01$ is $n \approx 167772$ for IEEE single precision arithmetic and $n \approx 9\ 10^{13}$ for double precision.

Then, we can show

Theorem 8.2. *Let $w = x - yz$ and $e = fl(w) - w$. If $nu \leqslant 0.01$, then*

$$|e| \leqslant 3.02\ u \max(|x|, |w|).$$

Proof. $fl(w)$ is defined as $fl(x - fl(yz))$. But, $fl(yz) = yz(1 + \epsilon_1),\ |\epsilon_1| \leqslant u$ and

$$\begin{aligned} fl(w) &= (x - yz(1 + \epsilon_1))(1 + \epsilon_2),\ |\epsilon_2| \leqslant u, \\ &= x(1 + \epsilon_2) - yz(1 + \epsilon_1)(1 + \epsilon_2). \end{aligned}$$

This can be written with the help of Lemma 8.1,

$$\begin{aligned} fl(w) &= x(1 + \theta_2 u) - yz(1 + 2.02\,\theta_1 u),\ |\theta_i| < 1, \\ &= x - yz + u(x\theta_1 - 2.02\,yz\theta_1). \end{aligned}$$

Therefore,

$$|e| \leqslant u(|x| + 2.02\ |w - x|),$$

From which the results follows. □

The last result can also be written as

$$|e| \leqslant u(2.02|yz| + |x|).$$

We have also,

$$|e| \leqslant u(2|yz| + |x|) + O(u^2).$$

The values of the constants involved in these inequalities like 2.02 or 3.02 are not really important, therefore, most of the time in the statements of the results, we will replace them by a generic constant C or C_i. However, for the sake of completeness, we will indicate their possible values.

Now, we are able to analyze the effect of round off errors in Gaussian elimination. There are several different ways to approach this problem. The one that is more popular today is called inverse or backward error analysis. It has been introduced by W. Givens and developed, in particular for Gaussian elimination, by J. Wilkinson [1965]. In this type of analysis the rounding errors are related to the data. The result of floating point operations is interpreted as the exact result of ordinary arithmetic on some perturbed data and some bounds are derived for the perturbations. This allows to be able to use ordinary arithmetic.

Examples of difficulties

Let us give a few small examples showing some of the problems that can happen with Gaussian elimination and finite precision computations.

We suppose $\beta = 10$, $t = 3$ (we do not care about limits on the exponent range) and, for the sake of simplicity, chopped arithmetic, the operations being done on normalized numbers. Although these assumptions are not realistic, the examples will give us an idea of what could happen on real problems. Consider the following linear system (Example 1),

$$\begin{aligned} 3x_1 + 3x_2 &= 6 \\ x_1 \ + \ \delta x_2 &= \delta + 1 \end{aligned}$$

where δ is a given real number ($\in F$). The exact solution is $x_1 = x_2 = 1$.

Choosing 3 as the pivot and noticing that the computed value of the multiplier $1/3$ is $0.333\ 10^0$, we obtain for the second equation

$$(\delta - 0.999\ 10^0)x_2 = \delta + 1 - 6 \times (0.333\ 10^0) = \delta - 0.990\ 10^0.$$

Therefore, if $\delta = 0.990$, then $x_2 = 0$ and $x_1 = 0.199\ 10^1$!

One can argue that this system is close to being singular, but this is not so as

$$\det(A) = 3 \times 0.99 - 3 = -0.03$$

Notice that even though the solution is wrong, the residual $b - Ax$ is small being $(0.03\ 0)^{\mathrm{T}}$. Remark also that the pivot is not small and that, in this case, if we consider the permuted system (Example 2),

$$\begin{aligned} x_1 + 0.99x_2 &= 1.99 \\ 3x_1 \ + \ 3x_2 &= 6 \end{aligned}$$

then, the reduced system is

$$(3 - 3 \times 0.99)x_2 = 6 - 3 \times 1.99$$

$$0.03x_2 = 0.03$$

giving $x_2 = 1$ and then $x_1 = 1$.

Notice that the first system is just obtained by using partial pivoting on the second one. This gives an example where it does not pay to use partial pivoting. If one considers a more general 2×2 system (Example 3),

$$\begin{aligned} \alpha x_1 + \beta x_2 &= \alpha + \beta \\ \gamma x_1 \ + \ \delta x_2 &= \gamma + \delta \end{aligned}$$

Then, it can be easily shown that if $fl(x \bullet y) = (x \bullet y)(1 + \epsilon)$, the computed value of x_2 is

$$1 + \epsilon \left(1 - \frac{2\alpha\gamma}{\alpha\delta - \beta\gamma} \right) + O(\epsilon^2).$$

Therefore (as it is well known) it is not necessary to have a small determinant $\alpha\delta - \beta\gamma$ to have a large error. This can happen also in this case if the product $\alpha\gamma$ is large.

Consider, for instance, the system (Example 4),

$$\begin{aligned} 30x_1 + x_2 &= 31 \\ 10x_1 + x_2 &= 11 \end{aligned}$$

The determinant is 20. The computed value of 10/30 is 0.333, therefore the reduced equation is

$$\begin{aligned} 0.667x_2 &= 11 - 0.310\ 10^2 \times 0.333\ 10^0 \\ &= 11 - 0.103\ 10^2 \\ &= (0.110 - 0.103)10^2 = 0.700\ 10^0 \end{aligned}$$

and $x_2 = 0.104\ 10^1$.

This is a 4% error in the second component and

$$\begin{aligned} x_1 &= (0.310\ 10^2 - 0.104\ 10^1) \times 0.333\ 10^{-1} \\ &= ((0.310 - 0.010)\ 10^2 \times 0.333\ 10^{-1} \\ &= 0.3\ 10^2 \times 0.333\ 10^{-1} \\ &= 0.990\ 10^0 \end{aligned}$$

a 1% error. However, small pivots may also lead to large errors. Consider (Example 5),

$$\begin{aligned} 0.3\ 10^{-3}x_1 + x_2 &= 1 \\ x_1 \quad + \quad x_2 &= 2 \end{aligned}$$

Then, the computed value of x_2 is 1 and x_1 is 0. The "exact" solution is $x_1 = 1.0003$ and $x_2 = 0.9997$.

Partial pivoting has been invented to prevent this to happen. If we permute the equations

$$\begin{aligned} x_1 \quad + \quad x_2 &= 2 \\ 0.3\ 10^{-3} + x_2 &= 1 \end{aligned}$$

then, $x_2(1 - 0.3\ 10^{-3}) = 1 - 0.6\ 10^{-3} = 1$, giving $x_2 = 1$ and the first equation gives also $x_1 = 1$.

Errors in the LU factorization

Of course, the rounding errors depend on the order in which the operations are done. Therefore, all the variants and implementations of Gaussian elimination are different in this respect.

Let us analyze the standard algorithm that has been described in Section 3. Notice that permutations have no effects on rounding errors (they are just recorded by pointers). For convenience in this Section we will denote by the same notations as before the computed quantities as there will be no ambiguities.

Let us first analyze the kth–step.

Theorem 8.3.

$$L_k A_k = A_{k+1} + E_k,$$

where

$$|(E_k)_{i,j}| \leqslant Cu \, \max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|),$$

with $C = 3.02$.

Proof. The multipliers (e.g. the elements of L) that we denote by $l_{i,k}$ are

$$l_{i,k} = fl\left(\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}\right), \; i \geqslant k+1,$$

$$a_{i,j}^{(k+1)} = \begin{cases} 0 & i \geqslant k+1, \; j = k, \\ fl(a_{i,j}^{(k)} - l_{i,k} a_{k,j}^{(k)}), & i \geqslant k+1, \; j \geqslant k+1, \\ a_{i,j}^{(k)} & \text{otherwise.} \end{cases}$$

Let us first consider the multipliers, $i \geqslant k+1$,

$$l_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}(1 + \epsilon_{i,k}), \; |\epsilon_{i,k}| \leqslant u.$$

This translates into

$$a_{i,k}^{(k)} - l_{i,k} a_{k,k}^{(k)} + a_{i,k}^{(k)} \epsilon_{i,k} = 0.$$

If we denote by $e_{i,k}^{(k)}$, the elements of E_k, this shows that

$$e_{i,k}^{(k)} = a_{i,k}^{(k)} \epsilon_{i,k}, \; i \geqslant k+1.$$

For $i \geqslant k+1$ and $j \geqslant k+1$, we have

$$a_{i,j}^{(k)} = fl(a_{i,j}^{(k)} - fl(l_{i,k} a_{k,j}^{(k)})).$$

From Theorem 8.2, we have

$$|e_{i,j}^{(k)}| \leqslant Cu \, \max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|).$$

As this bound is certainly also true for $e_{i,k}^{(k)}$, we have

$$|e_{i,j}^{(k)}| \leqslant Cu \max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|),\ i \geqslant k+1,\ j \geqslant k,$$

and the other entries of E_k are zero. □

With the notations of Lemma 3.5, we have seen that L_k has the form

$$L_k = I - l_k e_k^{\mathrm{T}}.$$

Lemma 8.2. *If B_k is a matrix whose first k rows are zero, then*

$$L_i B_k = B_k,\ i \leqslant k.$$

Similarly, $(L_i)^{-1} B_k = B_k$.

Proof.

$$L_i B_k = (I - l_i e_i^{\mathrm{T}}) B_k = B_k - l_i e_i^{\mathrm{T}} B_k = B_k,$$

as $e_i^{\mathrm{T}} B_k = 0$. □

Theorem 8.4. *Let F be defined as*

$$F = F_1 + \cdots + F_{n-1},$$

where

$$(F_k)_{i,j} = \begin{cases} 1 & i \geqslant k+1,\ j \geqslant k \\ 0 & \textit{otherwise} \end{cases}$$

that is,

$$F = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2 & \dots & 2 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2 & 3 & \dots & n-1 \end{pmatrix}.$$

Then,

$$A = LU + E,$$

with

$$|E| \leqslant Cu \max_{k,i,j} |a_{i,j}^{(k)}|\ F,$$

with $C = 3.02$.

Proof. We have

$$L_k A_k = A_{k+1} + E_k \Longrightarrow A_k = L_k^{-1} A_{k+1} + E_k.$$

By Lemma 8.2,

$$(L_1)^{-1} \ldots (L_{k-1})^{-1} A_k = (L_1)^{-1} \ldots (L_k)^{-1} A_{k+1} + E_k.$$

We sum up these equalities for $k = 1$ to $n - 1$. Most of the terms cancel and we obtain,

$$A = (L_1)^{-1} \ldots (L_{k-1})^{-1} A_n + E_1 + \ldots + E_{n-1}.$$

Therefore,

$$A = LU + E,$$

with

$$E = E_1 + \ldots + E_{n-1}.$$

Our next task is to bound the elements of E. It is easy to see that

$$|E| \leqslant Cu \max_{k,i,j} |a_{i,j}^{(k)}| \; F.$$

□

Theorem 8.4 means that the matrices L and U are the exact factors of the factorization of a perturbed matrix $A - E$. As the elements of F are bounded at most by $n - 1$, $|E|$ being small depends on $\max_{i,j,k} |a_{i,j}^{(k)}|$.

The previous bound can be written in terms of the growth factor g_A defined in Chapter 1,

$$|E| \leqslant Cu \, g_A \|A\|_\infty F.$$

It can be shown easily that,

$$\|E\|_\infty \leqslant Cu \, g_A \, n^2 \|A\|_\infty.$$

Errors in the triangular solves

It is a little easier to bound the error arising when solving a triangular system. Let us consider

$$Ly = b.$$

Then, on the computer we get,

$$y_1 = fl\left(\frac{b_1}{l_{1,1}}\right),$$
$$y_i = fl\left(\frac{b_i - l_{i,1}y_1 - \cdots - l_{i,i-1}y_{i-1}}{l_{i,i}}\right),\ i = 2, \ldots, n.$$

The analysis is done in Forsythe and Moler [1967] by the same technique as in the last Section. One can see also Higham [1989a] where the following result is proved.

Theorem 8.5. *We have,*

$$(L + K)y = b,$$

with

$$|K| \leqslant Cu \begin{pmatrix} |l_{1,1}| & & & & \\ |l_{2,1}| & 2|l_{2,2}| & & & \\ 2|l_{3,1}| & 2|l_{3,2}| & 2|l_{3,3}| & & \\ \vdots & \vdots & \vdots & \ddots & \\ (n-1)|l_{n,1}| & (n-1)|l_{n,2}| & (n-2)|l_{n,3}| & \ldots & 2|l_{n,n}| \end{pmatrix},$$

with $C = 1.01$ *and*

$$|K| \leqslant Cnu|L|,$$
$$\|K\|_\infty \leqslant \frac{n(n+1)}{2} Cu \max_{i,j} |l_{i,j}|.$$

Errors in the solution of a linear system

When the inexact factors are used to obtain the computed solution of the linear system, what we get is

$$(L + K)y = b,$$
$$(U + G)x = y,$$

with

$$|K| \leqslant Cnu|L|,$$
$$|G| \leqslant Cnu|U|.$$

Theorem 8.6. *The computed solution* x *satisfies*

$$(A + H)x = b,$$

with

$$|H| \leqslant C_1 u \max_{k,i,j} |a_{i,j}^{(k)}| F + C_2 n u |L||U| + O(u^2),$$

with $C_1 = 3.02$ *and* $C_2 = 2.02$.

Proof.

$$(L + K)(U + G)x = (LU + KU + LG + KG)x = b,$$

and

$$LU = A - E.$$

Then, if we denote by H the following matrix

$$H = KU + LG + KG - E,$$

we have $(A + H)x = b$ and

$$\begin{aligned} |H| &\leqslant |E| + |K||U| + |L||G| + |K||G|, \\ |H| &\leqslant C_1 u \max_{k,i,j} |a_{i,j}^{(k)}| F + C_2 n u |L||U| + C_3 n^2 u^2 |L||U|, \\ \|H\|_\infty &\leqslant (C_3 n^3 + C_4 n^2) u g_A \|A\|_\infty. \end{aligned}$$

□

The role of pivoting

Pivoting techniques are not usually only used to insure that the pivots are non zero but, also to reduce the growth factor. If partial pivoting is used, then

$$|l_{i,j}| \leqslant 1.$$

Therefore, $\|L\|_\infty \leqslant n$ and

$$|a_{i,j}^{(k+1)}| \leqslant |a_{i,j}^{(k)}| + |a_{k,j}^{(k)}| + \epsilon \max(|a_{i,j}^{k+1)}|, |a_{i,j}^{(k)}|).$$

Let $\rho_k = \max_{i,j} |a_{i,j}^{(k)}|$, then

$$\rho_{k+1} \leqslant 2(1 + \epsilon)\rho_k.$$

Hence, by recurrence

$$\rho_n \leqslant 2^{n-1}(1 + \epsilon)^{n-1}\rho_1.$$

The term $(1+\epsilon)^{n-1}$ is insignificant as ϵ is small. In fact, Wilkinson [1965] exhibited some contrived examples where an exponential growth is actually obtained. A well known example is

$$A = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 & 1 \\ -1 & 1 & 0 & \dots & 0 & 1 \\ -1 & -1 & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 & 1 \\ -1 & -1 & \dots & -1 & 1 & 1 \\ -1 & -1 & \dots & -1 & -1 & 1 \end{pmatrix}.$$

The bound is also attained for all $B = DAD$ with $D = \mathrm{diag}(1, -1, \dots, (-1)^{n+1})$. Although there exist such examples, in most practical cases the growth factor is small and Gaussian elimination with partial pivoting is a safe algorithm. However, there are cases where large growth can occur and the algorithm must be used cautiously.

Wright [1993] has exhibited problems arising from solving two point boundary value problems with multiple shooting methods for which an exponential growth is observed. Due to the structure of the problem, no pivoting occurs when using partial pivoting. Moreover, not only the growth factor is large but also, large errors are measured.

Foster [1994] has also given practical examples where g_A grows exponentially. These examples come from Volterra integral equations. Again, no row interchange is needed by partial pivoting. Analytic asymptotic value of the growth factor is obtained as well as lower bounds involving the coefficients of the equation. Examples are given in Foster [1994] from population dynamics.

The average case stability of Gaussian elimination with partial pivoting has been studied in Trefethen and Schreiber [1990]. Random matrices of order $\leqslant 1024$ have been looked at. The average growth factor was approximately $n^{3/2}$ for partial pivoting and $n^{1/2}$ for complete pivoting.

These examples show that users of partial pivoting must carefully analyzed the results.

Another possible choice is complete pivoting where the pivot is searched in $i, j > k$. Then, Wilkinson [1965] showed that the growth factor is bounded, as

$$|a_{i,j}^{(k)}| \leqslant k^{\frac{1}{2}}(2 \cdot 3^{\frac{1}{2}} \cdots k^{\frac{1}{k-1}})^{\frac{1}{2}} \max_{i,j} |a_{i,j}|.$$

It was conjectured that in this case $g_A \leqslant n$.

Cryer [1968] proved that this is true for $n \leqslant 4$. However, the conjecture has been shown to be false for $n > 4$ if rounding error is allowed by Gould [1991]. Edelman and Ohlroch [1991] modified the Gould counterexample to show that the conjecture is also false in exact arithmetic.

N. Higham and D. Higham [1989] have exhibited some matrices of practical interest which have a growth factor at least $\frac{n}{2}$ for complete pivoting. They proved the following result.

Theorem 8.7. *Let* $\alpha = \max_{i,j} |a_{i,j}|$, $\beta = \max_{i,j} |(A^{-1})_{i,j}|$, $\theta = \frac{1}{\alpha\beta}$.

Then, $\theta \leqslant n$ *and for any permutation matrix* P *and* Q *such that* PAQ *has an* LU *factorization, the growth factor* g *without pivoting on* PAQ *satisfies*

$$g \geqslant \theta.$$

Proof.

$$|(U^{-1})_{n,n}| = |e_n^{\mathrm{T}} U^{-1} e_n| = |e_n^{\mathrm{T}} U L^{-1} e_n| = |e_n^{\mathrm{T}} Q^{\mathrm{T}} A^{-1} P^{\mathrm{T}} e_n|,$$

as $L^{-1}e_n = e_n$. Therefore,

$$|(U_{n,n})^{-1}| = |(U^{-1})_{n,n}| = |(A^{-1})_{i,j}| \leqslant \beta,$$

for some (i, j). Clearly,

$$\max_{k,i,j} |a_{i,j}^{(k)}| \geqslant |U_{n,n}| \geqslant \frac{1}{\beta}.$$

□

With no pivoting at all, $(U^{-1})_{n,n} = (A^{-1})_{n,n}$, therefore it is easy to construct examples with a large growth factor by building matrices whose inverses have a small (n, n) entry. Notice that a large growth factor is a property of the matrix itself and not of the algorithm.

Perturbation analysis

So far, we have been concerned with the consequences of running the Gaussian elimination algorithm but we should also look at the consequences of perturbations on the data. When the matrix A or the right hand side b are inputted (from scratch or from some other computations) some errors could also be introduced.

There are different ways to measure these perturbations (see Chatelin and Fraysse [1993]) and a lot of literature on this topic. One of the oldest method has been introduced by A. Turing in 1949 and then developed by J. Wilkinson [1965]. It is called normwise analysis.

• *Normwise error analysis*

To the data (matrix A and right hand side b), we associate the computed solution y:

$$(A, b) \to y.$$

We have to choose how to measure distances in both spaces. Let x and y be such that

$$\begin{aligned} Ax &= b, \\ (A + \Delta A)y &= b + \Delta b. \end{aligned}$$

ΔA and Δb will be chosen such that

$$\|\Delta A\| \leqslant \alpha\omega, \quad \|\Delta b\| \leqslant \beta\omega,$$

ω is given, α (resp. β) will be 0 or $\|A\|$ (resp. $\|b\|$) depending on whether A, or b, or both are perturbed. ω defines the normwise relative perturbation. Following Golub and Van Loan [1989], we have

Lemma 8.3. *If $\xi = \alpha\omega\|A^{-1}\| < 1$, then*

$$\frac{\|y\|}{\|x\|} \leqslant \frac{1}{1-\xi}\left(1 + \frac{\xi\beta}{\alpha\|x\|}\right).$$

Proof. The first question is to know if $A + \Delta A$ is non singular. But, we notice that

$$A + \Delta A = A(I + A^{-1}\Delta A),$$

and, with the hypothesis

$$\|A^{-1}\Delta A\| \leqslant \alpha\omega\|A^{-1}\| = \xi < 1.$$

Then, $A + \Delta A$ is non singular by Lemma 2.3.3 of Golub and Van Loan [1989]. We have

$$(I + A^{-1}\Delta A)y = A^{-1}(b + \Delta b) = x + A^{-1}\Delta b.$$

Taking norms,

$$\|y\| \leqslant \frac{1}{1 - \alpha\omega\|A^{-1}\|}(\|x\| + \beta\omega\|A^{-1}\|).$$

But,

$$\omega \leqslant \frac{1}{\alpha\|A^{-1}\|} \Longrightarrow \|y\| \leqslant \frac{1}{1 - \alpha\omega\|A^{-1}\|}\left(\|x\| + \xi\frac{\beta}{\alpha}\right).$$

Therefore,

$$\frac{\|y\|}{\|x\|} \leqslant \frac{1}{1 - \alpha\omega\|A^{-1}\|}\left(1 + \xi\frac{\beta}{\alpha\|x\|}\right).$$

If $\alpha = \|A\|$ and $\beta = \|b\|$, then

$$\frac{\|y\|}{\|x\|} \leqslant \frac{1+\xi}{1-\xi}.$$

□

Theorem 8.8. *Under the conditions of Lemma 8.3,*

$$\frac{\|x-y\|}{\|x\|} \leqslant \omega \left(\|A^{-1}\| \frac{\alpha\|x\|+\beta}{\|x\|} \right) \left(\frac{1}{1-\xi} \right).$$

Proof. There are different ways to prove this result. For instance,

$$\begin{aligned}
y - x &= A^{-1}\Delta b - A^{-1}\Delta A y, \\
\|y - x\| &\leqslant \beta\omega\|A^{-1}\| + \alpha\omega\|A^{-1}\| \, \|y\|, \\
\frac{\|y-x\|}{\|x\|} &\leqslant \|A^{-1}\| \left(\frac{\beta\omega}{\|x\|} + \frac{\alpha\omega}{1-\alpha\omega\|A^{-1}\|} \left(1 + \frac{\xi\beta}{\alpha\|x\|} \right) \right), \\
&\leqslant \frac{\|A^{-1}\|}{\|x\|}(\alpha\|x\|+\beta) \left(\frac{\omega}{1-\alpha\omega\|A^{-1}\|} \frac{\alpha\|x\|+\beta\xi}{\alpha\|x\|+\beta} + \frac{\beta\omega}{\alpha\|x\|+\beta} \right), \\
&\leqslant \omega\|A^{-1}\| \frac{\alpha\|x\|+\beta}{\|x\|} \left(\frac{1}{1-\xi} \frac{\alpha\|x\|+\beta\xi}{\alpha\|x\|+\beta} + \frac{\beta}{\alpha\|x\|+\beta} \right), \\
&\leqslant \omega\|A^{-1}\| \frac{\alpha\|x\|+\beta}{\|x\|} \left(\frac{1}{1-\xi} \right).
\end{aligned}$$

If $\alpha = \|A\|$ and $\beta = \|b\|$, then

$$\alpha + \frac{\beta}{\|x\|} = \|A\| \left(1 + \frac{\|b\|}{\|A\| \, \|x\|} \right) \leqslant 2\|A\|,$$

and

$$\frac{\|y-x\|}{\|x\|} \leqslant 2\omega\|A\| \, \|A^{-1}\| \left(\frac{1}{1-\xi} \right).$$

□

Let us define

$$K_{\mathrm{T}}(A,b) = \|A^{-1}\| \frac{\alpha\|x\|+\beta}{\|x\|}.$$

This is the (normwise) condition number of the problem that "measures" the sensitivity of the solution of $Ax = b$ to perturbations. The subscript T refers to Turing. Notice that if there are only perturbations in A, then $\beta = 0$ and $K_{\mathrm{T}}(A) = \|A^{-1}\| \, \|A\|$.

Now, let η_{T} be defined on the set of all possible perturbations such that $(A + \Delta A)y = b + \Delta b$, where y is the computed solution (see Chatelin and Fraysse [1993]),

$$\eta_{\mathrm{T}} = \inf\{\omega |\ \omega \geqslant 0,\ \|\Delta A\| \leqslant \omega\alpha,\ \|\Delta b\| \leqslant \omega\beta,\ (A + \Delta A)y = b + \Delta b\}.$$

Theorem 8.9. *Let $r = b - Ay$ be the residual,*

$$\eta_{\mathrm{T}} = \frac{\|r\|}{\alpha\|y\| + \beta}.$$

Proof. As $(A + \Delta A)y = b + \Delta b$, we have

$$\Delta A y - \Delta b = r.$$

Therefore,

$$\|r\| \leqslant \omega(\alpha\|y\| + \beta),$$

which implies that for all cases

$$\omega \geqslant \frac{\|r\|}{\alpha\|y\| + \beta}.$$

Now, we only have to exhibit a case for which equality occurs. Consider $\alpha = 0$ (e.g. $\Delta A = 0$) and $\Delta b = \omega b$. Then,

$$Ay = (1 + \omega)b \quad \text{or} \quad \omega b = -r.$$

This gives

$$\omega = \frac{\|r\|}{\|b\|},$$

which is the equality we were looking for. □

η_{T} is sometimes called the (normwise) backward error. It measures the minimal distance to a perturbed problem that is solved exactly by the computed solution. We have seen that (approximately), the forward error ($\frac{\|y-x\|}{\|x\|}$) is the condition number times the backward error. Unfortunately, most of the time, the bounds that we have just established are too pessimistic and do not reflect the reality of the computation.

A topic that has been widely studied (see for instance Cline, Moler, Stewart and Wilkinson [1979]) is finding estimates for the normwise condition number. This is much less interesting today as people are mainly considering other ways to measure sensitivity to perturbations.

Beginning in the sixties, notably in the work of F.L. Bauer, a new style of perturbation analysis has been developed considering componentwise perturbations. This has later been studied by Skeel [1979] and is much in favor today. This perturbation analysis tries to assess the consequences of having perturbations on individual elements of A or b.

• *Componentwise error analysis*
We consider perturbations ΔA and Δb such that

$$|\Delta A| \leqslant \omega E, \quad |\Delta b| \leqslant \omega f,$$

and

$$(A + \Delta A)y = b + \Delta b,$$

where E and f are given matrix and vector of non negative entries and for a matrix B, $|B|$ denotes the matrix whose entries are the absolute values or modulus of the entries of B. Clearly, if $E_{i,j} = 0$ (resp. $f_i = 0$), then $(\Delta A)_{i,j} = 0$ (resp. $(\Delta b)_i = 0$), e.g. the corresponding entry is not perturbed.

Common choices for E and f are $(|A|, |b|)$, $(0, |b|)$, $(|A|, 0)$. This allows also special choices for sparse matrices taking into account the sparsity structure. We choose to measure distances with the $\|\cdot\|_\infty$ norm. The analysis is almost the same as for the normwise case.

Theorem 8.10. *If* $\omega \| \, |A^{-1}|E| \|_\infty < 1$,

$$\frac{\|y - x\|_\infty}{\|x\|_\infty} \leqslant \omega \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty} \frac{1}{1 - \omega \| \, |A^{-1}|E| \|_\infty}.$$

Proof. We have

$$y - x = A^{-1}\Delta b - A^{-1}\Delta A(y - x) - A^{-1}\Delta A x.$$

Taking absolute values,

$$|y - x| \leqslant \omega |A^{-1}|(f + E|x|) + \omega |A^{-1}|E|y - x|.$$

By taking norms, we obtain the result. □

The (componentwise) condition number of the problem is defined as

$$K_{\mathrm{BS}}(A, b) = \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty}.$$

The subscript BS refers to Bauer and Skeel.

Other condition numbers can be exhibited depending on the metric that is chosen, see Chatelin and Fraysse [1993]. Remark that if $f = 0$ and $E = |A|$ then,

$$K_{\mathrm{BS}} \leqslant \| \, |A^{-1}| \, |A| \, \|.$$

Let

$$\eta_{\mathrm{BS}} = \inf\{\omega |, \omega \geqslant 0, \ |\Delta A| \leqslant \omega E, \ |\Delta b| \leqslant \omega f, \ (A + \Delta A)y = b + \Delta b\},$$

be the componentwise backward error. Oettli and Prager [1964] proved the following result.

Theorem 8.11.

$$\eta_{\mathrm{BS}} = \max_i \frac{|(b - Ay)_i|}{(E|y| + f)_i}.$$

An algorithm is said to be backward stable when the backward error is of the order of the machine precision u. Gaussian elimination with partial pivoting is both normwise and componentwise backward unstable. As we have said, there are examples where η_{T} or η_{BS} are large compared to machine precision.

Despite this fact, the method can be used safely on most practical examples. We will also see later that there are some remedies to this backward instability.

• *Componentwise condition numbers*

Chandrasekaran and Ipsen [1995] analyzed the errors in components of the solution of a linear system when the right hand side is perturbed.

Theorem 8.12. *Let*

$$Ax = b$$
$$Ay = b + \Delta b$$

and c_i^{T}, $1 \leqslant i \leqslant n$, *be the rows of* A^{-1}. *Moreover, let* β_i *be the angle between* c_i *and* b, ψ_i *be the angle between* c_i *and* Δb *and* $\epsilon_b = \frac{\|\Delta b\|}{\|b\|}$, *then (when* $x_i \neq 0$*)*

$$\frac{y_i - x_i}{x_i} = \frac{\|\Delta b\| \cos\psi_i}{\|b\| \cos\beta_i} = \frac{\|b\|}{\|A\|\,\|x\|} \frac{\|x\|}{x_i} \|A\|\,\|c_i\|\,\epsilon_b \cos\psi_i.$$

Proof. We have

$$y = A^{-1}(b + \Delta b).$$

Then,

$$y - x = A^{-1}\Delta b.$$

Therefore,

$$y_i - x_i = c_i^{\mathrm{T}} \Delta b = \|\Delta b\|\,\|c_i\|\,\cos\psi_i,$$

and

$$x_i = c_i^{\mathrm{T}} b = \|c_i\|\,\|b\|\,\cos\beta_i,$$

from which the result follows. □

Perturbing the matrix, we get the following result.

Theorem 8.13. *Let*

$$\begin{aligned} Ax &= b \\ (A + \Delta A)y &= b \end{aligned}$$

Moreover, let ψ_i be the angle between c_i and $\Delta A\, y$ and $\epsilon_A = \frac{\|\Delta A\, y\|}{\|A\|\,\|y\|}$, then (when $x_i \neq 0$)

$$\frac{y_i - x_i}{x_i} = -\frac{1}{\cos\beta_i}\frac{\|\Delta A y\|}{\|b\|}\cos\psi_i = -\frac{\|y\|}{x_i}\|A\|\,\|c_i\|\,\epsilon_A \cos\psi_i.$$

Proof. We have

$$y - x = -A^{-1}\Delta A y.$$

Therefore,

$$y_i - x_i = -c_i^{\mathrm{T}}\Delta A y = -\|c_i\|\,\|\Delta A y\|\,\cos\psi_i,$$

from which the result follows. □

These results lead Chandrasekaran and Ipsen [1995] to define the componentwise conditions numbers as

$$\frac{\|y\|}{|x_i|}, \quad \|A\|\,\|c_i\|.$$

A geometric interpretation of these choices is given in Chandrasekaran and Ipsen [1995]. Relating these results to componentwise perturbation analysis, we have the following result.

Theorem 8.14. *If we have $|\Delta b| \leqslant \omega|b|$, then,*

$$\frac{|y_i - x_i|}{|x_i|} \leqslant \omega\frac{|c_i^{\mathrm{T}}|\,|b|}{|c_i^{\mathrm{T}} b|}.$$

If $|\Delta A| \leqslant \omega|A|$, then

$$\frac{|y_i - x_i|}{|x_i|} \leqslant \omega\frac{|c_i^{\mathrm{T}}|\,|A|\,|y|}{|x_i|}.$$

Proof. Straightforward □

A posteriori errors bounds

Suppose that the matrix A is symmetric and positive definite and we have an approximate solution x_0. Then, we know that the error e satisfies

$$Ae = r_0 = b - Ax_0,$$

and therefore

$$\|e\|^2 = (A^{-1}r_0, A^{-1}r_0) = (A^{-2}r_0, r_0).$$

As A is symmetric, it can be written as

$$A = Q\Lambda Q^{\mathrm{T}},$$

where Q is an orthonormal matrix whose columns are the normalized eigenvectors of A and Λ is a diagonal matrix whose diagonal elements are the eigenvalues λ_i,

$$a = \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n = b.$$

Then, if u and v are two vectors and f is a given function,

$$\begin{aligned}(u, f(A)v) &= (u, Qf(\Lambda)Q^{\mathrm{T}}v)\\ &= (\alpha, f(\lambda)\beta)\\ &= \sum_{i=1}^{n} f(\lambda_i)\alpha_i\beta_i.\end{aligned}$$

This last sum can be considered as a Riemann–Stieltjes integral

$$I[f] = (u, f(A)v) = \int_a^b f(\lambda)\, d\alpha(\lambda),$$

where the measure is piecewise constant and defined by

$$\alpha(\lambda) = \begin{cases} 0 & \text{if } \lambda \leqslant a\\ \sum_{j=1}^{i} \alpha_i\beta_i & \text{if } \lambda_i < \lambda \leqslant \lambda_{i+1}\\ \sum_{j=1}^{n} \alpha_i\beta_i & \text{if } b < \lambda \end{cases}$$

The case we are interested in is $u = v = r_0$ and $f(x) = \frac{1}{x^2}$. This was considered in Dahlquist, Eisenstat and Golub [1972]. Numerical methods for obtaining bounds on the integral (and therefore on the error) are given in Golub and Meurant [1994]. These methods use quadrature formulas (Gauss, Gauss–Radau and Gauss–Lobatto) to approximate the integral. These formulas rely on orthogonal polynomials(for the given measure) which are computed by running a few iterations of the Lanczos

process. Therefore, these computations involve only matrix× vector products and operations on tridiagonal matrices generated by the Lanczos algorithm. As long as only a few iterations are needed, this involves only $O(n^2)$ operations. These methods give bounds on the elements of the inverse of A or for the norm of the error e.

Back to the examples

It is interesting to look at some of these condition numbers and backward errors for some of the examples we have defined before. Remember that we have chosen $\beta = 10$ and $t = 3$.

In Example 1, we have

$$A = \begin{pmatrix} 3 & 3 \\ 1 & 0.99 \end{pmatrix},$$

the exact solution is $x = (1\,1)^{\mathrm{T}}$ and the computed solution is $y = (1.99\,0)^{\mathrm{T}}$. The (exact) inverse of A is

$$A^{-1} = 10^2 \begin{pmatrix} -0.33 & 1 \\ 1/3 & -1 \end{pmatrix}.$$

Then, $K_{\mathrm{T}}(A) = 666$ and $\eta_{\mathrm{T}} = 0.0033$. The product of these two quantities is 2.19.

Looking at componentwise analysis, we have $K_{\mathrm{BS}} = 399$, much smaller than K_{T} and $\eta_{\mathrm{BS}} = 0.005$. Then, the product of these two quantities is about 2.

Notice that we have

$$\frac{\|y - x\|}{\|x\|} = 0.995, \quad \frac{\|y - x\|_\infty}{\|x\|_\infty} = 1,$$

the bounds being only off by a factor of 2. Componentwise, we have

$$\left[\frac{y_i - x_i}{x_i}\right]_{i=1,2} = \begin{pmatrix} 0.989 \\ -1 \end{pmatrix}.$$

Notice that $|c_i^{\mathrm{T}}|$ is 105.3 for $i = 1$ and 105.4 for $i = 2$. Finally, we remark that the two lines corresponding to the two equations are almost the same explaining the problems we got computing the intersection.

Now, we consider Example 2 where,

$$A = \begin{pmatrix} 30 & 1 \\ 10 & 1 \end{pmatrix},$$

the exact solution is $x = (1\,1)^{\mathrm{T}}$ and the computed solution is $y = (0.99\,1.04)^{\mathrm{T}}$. The (exact) inverse of A is

$$A^{-1} = \begin{pmatrix} 0.05 & -0.05 \\ -0.5 & 1.5 \end{pmatrix}.$$

$K_{\mathrm{T}}(A) = 50.08$ and $\eta_{\mathrm{T}} = 0.00587$. The product of the condition number and the backward error is 0.294.

For componentwise analysis, $K_{\mathrm{BS}} = 31.999$ and $\eta_{\mathrm{BS}} = 0.0084$. Then, the product of these two quantities is about 0.27.

Looking at relative errors, we have

$$\frac{\|y - x\|}{\|x\|} = 0.0291, \quad \frac{\|y - x\|_\infty}{\|x\|_\infty} = 0.04.$$

Notice this is much smaller than the bounds involving the condition numbers. Componentwise, we have

$$\left[\frac{y_i - x_i}{x_i}\right]_{i=1,2} = \begin{pmatrix} -0.01 \\ 0.04 \end{pmatrix}.$$

Notice that $|c_i^{\mathrm{T}}|$ is 0.0707 for $i = 1$ and 1.5811 for $i = 2$. This indicates that the error should be larger on the second component, which is what we observed.

Let us look at Example 5,

$$A = \begin{pmatrix} 03\,10^{-2} & 1 \\ 1 & 1 \end{pmatrix},$$

the exact solution is $x = (1.0003\,0.9997)^{\mathrm{T}}$ and the computed solution is $y = (0\,1)^{\mathrm{T}}$. The (exact) inverse of A being

$$A^{-1} = \begin{pmatrix} -1.003 & 1.003 \\ 1.003 & -0.003 \end{pmatrix}.$$

$K_{\mathrm{T}}(A) = 2.63$ and $\eta_{\mathrm{T}} = 0.618$. The product of the condition number and the backward error is 1.6237.

For componentwise analysis, $K_{\mathrm{BS}} = 3$ and $\eta_{\mathrm{BS}} = 1$. Then, the product $K_{\mathrm{BS}}\eta_{\mathrm{BS}}$ is 3.

For relative errors, we have

$$\frac{\|y - x\|}{\|x\|} = 0.709, \quad \frac{\|y - x\|_\infty}{\|x\|_\infty} = 1.$$

Componentwise, we have

$$\left[\frac{y_i - x_i}{x_i}\right]_{i=1,2} = \begin{pmatrix} -1 \\ 0.003 \end{pmatrix}.$$

We note that $|c_i^{\mathrm{T}}|$ is 1.418 for $i = 1$ and 1.003 for $i = 2$, indicating that we have a larger error in the first component.

Scaling

Scaling is a transformation of the linear system to be solved trying to give a better behaved system before using Gaussian elimination. Let D_1 and D_2 be two non singular diagonal matrices. The system $Ax = b$ is transformed into

$$A'y = (D_1 A D_2)y = D_1 b,$$

and the solution x is recovered as $x = D_2 y$. Notice that left multiplication by D_1 is a row scaling and right multiplication by D_2 is a column scaling. The entry $a_{i,j}$ of A is changed into $d_i^1 d_j^2 a_{i,j}$ where $d_i^l,\ l = 1, 2$ are the diagonal entries of D_l.

Notice that if one uses Gaussian elimination with partial pivoting to solve the scaled system, row scaling influences the choice of the pivot. Classical strategies can be found in Curtis and Reid [1972]. Other proposals were done by Hager [1984]. A common strategy for row scaling is to divide the entries of a row by the infinity norm of the row. Skeel [1979] showed that a good scaling matrix is choosing the diagonal elements of D_1 as $d_i = (|A|\,|y|)_i$ where y is the computed solution. Of course, this is impractical as the solution y depends on the scaling. However, if an approximation c of the solution is known, then A could be scaled by $(|A|\,|c|)_i$.

Today, no scaling strategy has been shown to give consistently better results than not using any scaling at all although many different strategies have been proposed over the years. If the diagonal elements of D_1 and D_2 are (as it has been suggested) powers of the machine base then, if there are no overflows or underflows and no pivoting is used, the computed solution is the same as without scaling.

It has been argued that the real role of scaling is to alter the pivoting sequence. This can result in a better or worst solution. The rule of thumb given in Poole and Neal [1992] is that when scaling can lead to a system which is more diagonally dominant, it could be useful and otherwise it should be avoided.

Further remarks

• *Nearness to singularity*

When perturbing only the matrix entries, the normwise condition number is

$$K_I(A) = \|A\|\,\|A^{-1}\|,$$

the index I referring to the fact that this is also the condition number for matrix inversion. The singular value decomposition of A can be written, see Golub and Van Loan [1989],

$$A = U\Sigma V^{\mathrm{T}},$$

where U and V are n by n orthogonal matrices and

$$\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n),\ \sigma_1 \geqslant \cdots \geqslant \sigma_n \geqslant 0.$$

σ_n is the distance to the nearest singular matrix, if we measure distances in the l_2 norm or in the Frobenius norm. If we use the l_2 norm and if $\sigma_n > 0$, it is easy to see that

$$K_I(A) = \frac{\sigma_1}{\sigma_n}.$$

If we normalize A such that $\|A\|_2 = 1$, then $K_I(A) = \frac{1}{\sigma_n}$. The condition number is the inverse of the distance to the nearest singular matrix. In Demmel [1992c], J. Demmel studied the possible extension of these results to componentwise analysis. We briefly state these results informally. If we only perturb A, the condition number is

$$K(A, E) = \| \, |A^{-1}|E\|.$$

Let $\omega(A, E)$ be the smallest ω satisfying $|\Delta A| \leqslant \omega E$ and such that $A + \Delta A$ is singular. It has been proved (Rohn [1990]), that

$$\omega(A, E) = \frac{1}{\max_{S_1, S_2} \rho(S_1 A^{-1} S_2 E)},$$

where S_1 and S_2 are diagonal matrices with ± 1 on the diagonal and ρ is the spectral radius (max of modulus of the eigenvalues). If $E_{i,j} = 1$ and $\Omega = \{(x, y) | \, x_i = \pm 1, \, y_i = \pm 1\}$, then

$$\omega(A, E) = \frac{1}{\max_{(x,y)\in\Omega} |x^{\mathrm{T}} A^{-1} y|}.$$

Computing $\omega(A, E)$ is an NP–complete problem. There are no such results as for the normwise case relating the condition number to nearness to singularity. However, Demmel [1992c] proved the following bounds,

$$\omega(A, E) \geqslant \frac{1}{K(A, E)},$$

and

$$\frac{1}{\max_{i,j}(|A^{-1}_{i,j}|E_{i,j})} \geqslant \omega(A, E).$$

9. Iterative refinement

We have seen in the round off error analysis that the computed solution satisfies

$$(A + H)y = b,$$

with

$$\|H\|_\infty \leqslant u \; C\|A\|_\infty,$$

if the growth factor is bounded. Let $r = b - Ay$ be the residual, then

$$\|r\|_\infty \leqslant \|H\|_\infty \, \|y\|_\infty \leqslant uC\|A\|_\infty \, \|y\|_\infty.$$

Hence, if C is not too large, Gaussian elimination produces a small residual. But, small residuals do not always imply high accuracy in the solution. Let $e = x - y$, then

$$Ae = b - Ay = r.$$

Therefore, one natural idea is to solve

$$Ae = r.$$

This will produce a computed solution $\tilde{e}$, satisfying

$$(A + H)\tilde{e} = r$$

and we set $\tilde{x} = y + \tilde{e}$ as the new approximation to the solution. If we like, we can iterate this process. This algorithm is known as iterative refinement (or iterative improvement).

Here, the main question is to know to which precision the residual r has to be computed as we are not able to get the exact answer. If we have $\tilde{r} = fl(b - Ay)$ and $(A + H)\tilde{e} = \tilde{r}$, Skeel [1979] has shown that computing the residual with the same precision as the computations is enough to make Gaussian elimination with partial pivoting backward stable. This is stated in the following theorem.

Theorem 9.1. *If $u|A|\,|A^{-1}|$ is small enough, one step of iterative refinement with single precision residual computation is componentwise backward stable.*

Proof. We just give a rough informal sketch of the proof, see Skeel [1980] for details. All the constants below are functions of n only. The computed solution y verifies $(A + H)y = b$ and

$$|b - Ay| \leqslant uC\,|A|\,|x|,$$

with $|Hy| = O(u)$. More specifically

$$|Hy| \leqslant Cu\,|A|\,|x| + O(u^2).$$

The residual $\tilde{r} = fl(b - Ay) = Hy + z$ is such that

$$|z| \leqslant Cu|A|\,|x| + O(u^2).$$

When solving for the error, we have

$$(A + H)\tilde{e} = \tilde{r}.$$

Clearly, $|H\tilde{e}| = O(u^2)$. Finally

$$\begin{aligned} b - A\tilde{x} &= b - Ay - A\tilde{e}, \\ &= b - Ay + H\tilde{e} - Hy - z. \end{aligned}$$

Therefore,

$$|b - A\tilde{x}| \leqslant Cu\,|A|\,|x| + O(u^2).$$

We also have,

$$|A|\,|\tilde{x}| = |A|\,|x| + O(u).$$

These two statements imply the componentwise backward stability. □

However, one step of iterative refinement only gives a small backward error. It does not guarantee a better accuracy. If this is wanted, the residual must be computed in double precision.

10. Geometric analysis

In Poole and Neal [1991] and Neal and Poole [1992], was studied the geometry of Gaussian elimination. The solution of a linear system of order n is viewed as finding the intersection of n hyperplanes.

Consider the following simple example, using three decimal digits arithmetic, close to an example given in Poole and Neal [1991],

$$\mathrm{H}_1 : 5x_1 - x_2 = 24$$
$$\mathrm{H}_2 : 14x_1 + 80x_2 = 470$$

whose exact solution using 3–digits arithmetic is $(5.77, 4.86)$. The geometric interpretation of this system is shown on figure 10.1.

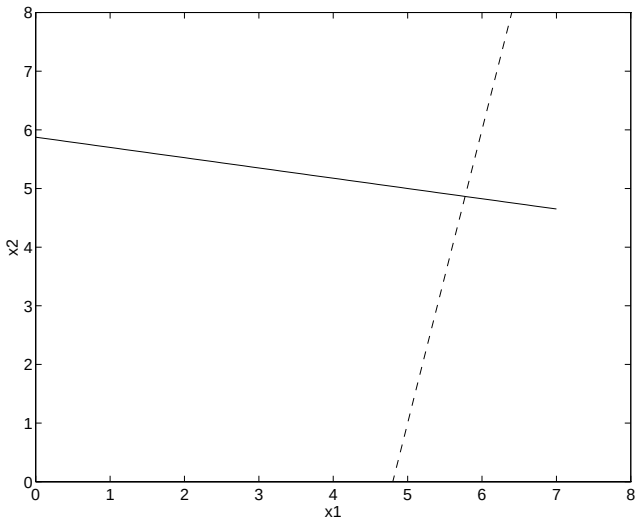

fig. 10.1

Notice that H_1 and H_2 are nearly orthogonal. If we use partial pivoting, this system is transformed into

$$\begin{aligned} H_2 &: 14x_1 + 80x_2 = 470 \\ H_1' &: \qquad 29.5x_2 = 143 \end{aligned}$$

This gives $x_2 = 4.85$

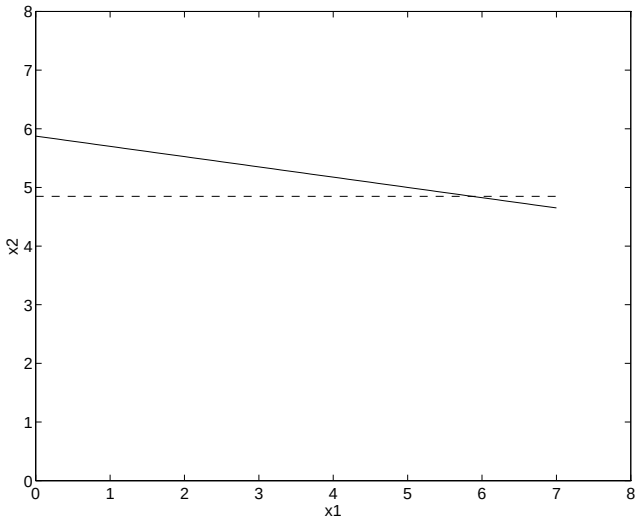

fig. 10.2

H_1' is parallel to the x_1 axis but with an approximate ordinate value. As H_2 is also nearly parallel to the x_1 axis, the error on x_2 is amplified and we find $x_1 = 5.85$

Generalizing this example, an hyperplane H_i (in the upper triangular system obtained by the forward sweep) is said to be poorly oriented with respect to the x_i–axis if there exists an entry $u_{i,j},\ i \neq j$ such that $|u_{i,j}| \geqslant |u_{i,i}|$. In this case, the backward sweep may lead to large errors.

Even if there are no rounding errors in the forward sweep, everything can be spoiled by the backward sweep. Consider the following bidiagonal matrix,

$$B = \begin{pmatrix} 1 & -\alpha & & & \\ & 1 & -\alpha & & \\ & & \ddots & \ddots & \\ & & & 1 & -\alpha \\ & & & & 1 \end{pmatrix}.$$

If we solve the system

$$B\epsilon = (0 \ldots 0\ \beta)^{\mathrm{T}},$$

then, we have

$$\begin{aligned}\epsilon_n &= \beta \\ \epsilon_i &= \alpha^{n-i}\epsilon_n\end{aligned}$$

Consequently, if $\alpha > 1$, the value of ϵ_n is amplified by this backward sweep even if the computations are done in exact arithmetic.

Therefore, in Gaussian elimination and under certain circumstances, if there is an error in the last component (coming for the forward sweep), it may be amplified by the backward sweep.

In Poole and Neal [1991], it is stated that a good pivoting strategy must address potential problems in both phases of Gaussian elimination. Notice that partial pivoting address only problems in the first phase.

The pivoting strategy should be such that during the first phase, the arrangement of equations gives the i–th hyperplane as nearly orthogonal as possible to the x_i–axis. Partial pivoting gives only a good hyperplane orientation in L. Complete pivoting addresses also the orientation problem in U. So, generally, there are less problems in the backward sweep with complete pivoting.

However, examples can be found (Poole and Neal [1991]) where partial pivoting is better than complete pivoting. It is stated in Poole and Neal [1991] that although the geometric analysis does not favor the use of partial pivoting, empirical evidence shows that for "most" linear systems partial pivoting gives an acceptable computed solution. This is attributed to the fact that in most systems hyperplanes are not nearly parallel to any axis at all and that computer precision is sufficiently good to postpone instability that could occur. Rook's pivoting was designed to address these problems.

In Neal and poole [1992], the concept of error multipliers was introduced to look at the amplification of errors in the backward sweep. Let H_k be the upper Hessenberg submatrix of U of order k such that $H_k = [u_{i,j}],\ n-k \leqslant i \leqslant n-1,\ n-k+1 \leqslant j \leqslant n$ and ϵ be the difference of the computed and exact solutions of $Ux = b$. Then,

$$\epsilon_{n-k} = \frac{(-1)^k \det H_k}{\prod_{i=n-k}^{n-1} u_{i,i}}\epsilon_n.$$

The multiplicative factor here is called the error multiplier (EM). It is clear that if some of the EMs are large then, a large amplification of errors can occur. When EMs are less or equal than 1, the backward sweep is generally without problems.

Interesting examples are shown in Neal and Poole [1992], particularly example 2.5 where a 5×5 system is given for which there is no rounding error using IEEE single precision arithmetic during the forward sweep. However, very large errors arise in the backward sweep coming from large EMs. Using IEEE double precision on this system, a good solution is obtained.

The examples in Neal and Poole [1992] show that, usually, partial pivoting produces an upper triangular system whose exact solution is very close to the solution of the original system. Partial pivoting also tends to produce hyperplanes that are well oriented with respect to each other.

From these studies, it is also clear that contrived examples exhibiting almost any behaviour (with partial pivoting, complete pivoting or rook's pivoting) can be constructed.

3. Vector and parallel algorithms for general systems

11. Introduction

In this Chapter, we consider the implementation of Gaussian elimination on vector and parallel computers.

It is always difficult to write about such a topic in a book or a review like this one, as the computer architectures evolve so fast that the risk is to be outdated almost the next day. Since the first years after World War II and the advent of the first real computers (with stored programs), there has been a tremendous progress both in the ease of use of scientific computers and in their floating point performances.

When I started learning numerical analysis in 1970, top scientific computers like the IBM 360–91 were running at about 1–2 Mflops (that is 10^6 floating point operations per second). Although there were already some forms of parallelism hidden in the computer architectures (particularly for the I/Os), they were mainly sequential machines. This culminated in the well known Control Data CDC 7600 that was running at about 5–10 Mflops on typical codes. Then, came the era of vector computers which has not ended yet in 1998. The first one commercially available (1976) was the CRAY 1 which had a peak speed of 160 Mflops and an average speed of 10 to 30 Mflops on real production codes.

These machines introduced a new kind of implementation problem. Roughly speaking, on sequential computers the most efficient algorithm was the one with good numerical properties that had the smallest total number of floating point operations. This is not true anymore on a vector machine. There is a lot of difference between the scalar and vector speed processing, typically a factor of 10. To run fast an algorithm has to be expressed (if possible) in terms of vectors. Moreover, the longer is the vector length, the faster is the speed of execution (up to an asymptotic value). The memory traffic issue is also very important to get good performances. Usually the fastest vector computers are the ones with the largest memory bandwith and the speed of algorithms depend also on the ratio of floating point operations to memory references.

The vector computers have now evolved into machines with several vector processors sharing a common large memory. Today (1998), these machines run at a few (1–100) Gflops (10^9 floating point operations per second).

Recently (end of the 80's, beginning of the 90's), a new kind of scientific computers have appeared on the commercial market. These machines are parallel computers with distributed memory. Although this is a quite old idea (the parallel computer Illiac IV was built in 1972), only recently had these machines attained a level of reliability good enough to allow their use in the industry. Right now, they deliver merely about the same performance as the top of the line parallel vector supercomputers. However, it has been shown on specific examples that they can reach Teraflops (10^{12} floating point operations per second). Now, some people are even seriously considering building Petaflops computers (10^{15} floating point operations per second) in about 20–25 years from now (1998). Of course, to fully exploit these architectures, we need to have parallel algorithms and suitable programming mod-

els.

It is remarkable that through all these changes the analysis that has been developed in the 60's by J. Wilkinson [1965] is still relevant. However, it should be noticed that together with the evolution of computers and particularly with huge shared or distributed memories, larger and larger problems are solved (see Edelman [1993]) and this can possibly raise some new computational problems concerning the accuracy of computations.

There is such a lot of different architectures available today, all with their details, that it is almost impossible to have a unified treatment of the implementation problems of Gaussian elimination on these computers. Therefore, we will concentrate on what we feel is the most important ways to construct reliable and fast software, that is basic algorithms. We will study this for parallel distributed architectures without looking too much into the details of implementation or performance issues.

12. BLAS routines

Software reuse and portability are issues that are almost as important as performances. In the 70's, Lawson, Hanson, Kincaid and Krogh [1979] described a set of basic routines commonly called the BLAS (Basic Linear Algebra Subprograms) for linear algebra problems. They are an aid to clarity and portability and also performance, as these routines can be implemented as efficiently as possible by each manufacturer, possibly in assembly language.

It turns out that the basic frequently occurring operations are most often the same in all linear algebra problems, so it was useful to develop a standard interface for these kernels. The set of operations described in 1979 is now referred as Level 1 BLAS or BLAS1. These routines are mainly concerned with vector operations like

$$
\begin{aligned}
&y = \alpha x + y, \quad \text{(Saxpy)}\\
&\alpha = x^{\mathrm{T}} y, \quad \text{(Sdot)}\\
&x = \alpha x, \quad \text{(Sscal)}\\
&x = y, \quad \text{(Scopy)}\\
&\alpha = \|x\|_2, \quad \text{(Snrm2)}
\end{aligned}
$$

where x and y are vectors and α is a scalar, the S standing for single precision. For a complete list, see Anderson and al [1992].

The well known linear algebra package LINPACK (for linear systems solves and least square problems) was written using BLAS1 and published in 1979 (Bunch, Dongarra, Moler and Stewart [1979]). BLAS1 involves $O(n)$ floating point operations on $O(n)$ data items, n being the length of the vectors.

At the time where the BLAS1 appeared and LINPACK was completed, the vector computers appeared on the market. One can think this was fine as the BLAS1 defined vector operations. Unfortunately, this is not completely true and the BLAS1 and LINPACK had poor performance on vector machines. The reason for that is the value of the ratio of floating point operations to data loads and stores. It is too low to keep the processor busy all the time, in which case the performance could be

smaller (sometimes by a large amount) than the peak theoretical speed. In BLAS1, there are very few possibilities of data reuse in vector registers or memory caches.

An additional set of routines called the Level 2 BLAS (BLAS2) was then designed, (Dongarra and al [1988]). They are based on matrix×vector operations. Examples are

$$\begin{aligned} y &= \alpha Ax + \beta y, \\ x &= Tx, \\ A &= \alpha xy^{\mathrm{T}} + A, \\ x &= T^{-1}x, \end{aligned}$$

where α, β are scalars, x, y are vectors, A and T are matrices, T being triangular.

Most algorithms of linear algebra can be coded using Level 2 BLAS including Gaussian elimination. Level 2 BLAS involves $O(n^2)$ floating point operations on $O(n^2)$ data items. Therefore, the ratio of Level 1 BLAS is not improved. But, for instance, in the first kernel above, data can be kept in the vector registers, improving the computational speed.

To improve on this point and to increase the data locality, a level 3 BLAS (BLAS3) has been proposed in 1990, see Dongarra and al [1990], that defines matrix×matrix operations. Examples are

$$\begin{aligned} C &= \alpha AB + \beta C, \\ C &= \alpha AA^{\mathrm{T}} + \beta C, \\ C &= \alpha AB^{\mathrm{T}} + \alpha BA^{\mathrm{T}} + \beta C, \\ B &= \alpha TB, \\ B &= \alpha T^{-1}B, \end{aligned}$$

α, β are scalars, A, B, C, T are matrices, T being triangular.

There, we have $O(n^3)$ floating point operations on $O(n^2)$ data items helping to improve the data reuse. Level 3 BLAS shows very good performances on vector supercomputers and computers with a memory hierarchy. Performance close to the peak speed is frequently obtained.

13. LAPACK (and follow ons)

At the end of the 80's, at the same time the Level 2 BLAS and Level 3 BLAS have appeared, a new software project has been developed. Its goal was to supersede both LINPACK and EISPACK (a well known package for eigenvalues computations) and also to obtain better performances. The computers targeted were parallel vector supercomputers with shared memory. Another goal was to improve the quality and accuracy of the algorithms, particularly for eigenvalue computations.

The first version of LAPACK (for Linear Algebra PACKage) has appeared officially in 1992, see Anderson and al [1992] and the second one in 1994. The strategy of LAPACK to obtain portable codes that are also efficient is to construct the software as much as possible using calls to the BLAS. The BLAS 2 and 3 can achieve

near peak performance on the targeted architectures. Moreover, it allows also to exploit parallelism in a transparent way.

Partitioned block forms of the LU (or other) factorizations (where point algorithms are used in which operations have been grouped together) are used in LAPACK in order to use Level 2 and 3 BLAS. Some routines also exist in LAPACK that return bounds on the componentwise backward error.

LAPACK has been extended in an almost transparent way to distributed memory parallel computers, see the ScaLAPACK library (Choi, Dongarra, Pozzo and Walker [1994]).

14. Triangular systems solvers on distributed memory computers

Introduction

Although only $O(n^2)$ operations are required for the forward and backward solves compared to the $O(n^3)$ operations of the LU factorization, the solution of triangular systems is an interesting challenge on parallel computers. Moreover, quite often, several systems with different right hand sides have to be solved and then, the cost of triangular solutions can be as large as the one for factorization.

In considering an algorithm for a triangular solve on a distributed memory parallel computer, two main issues have to be studied. The first one is to find some parallelism in a process which, at first sight, seem mostly sequential. The second issue is finding good data distributions in the local memories associated with each processor to minimize memory transfers.

Many parallel algorithms have been devised for solving triangular systems, see Heller [1978]. However, many of them assumed having $O(n^3)$ processors available and are not of practical use on present machines that have between a hundred and a thousand processors. Here, we mainly follow Heath and Romine [1988] and Eisenstat, Heath, Henkel and Romine [1988] and then, we describe some alternatives.

Let us start with serial algorithms and let

$$Lx = b,$$

to be solved, where L is lower triangular. There are basically two ways to compute the solution. The first one is the classical way, in which components of x are computed exactly one after another,

```
for i=1:n
  for j=1:i-1
    b(i)=b(i)-l(i,j)*x(j)
  end
  x(i)=b(i)/l(i,i)
end
```

This algorithm is called the scalar product algorithm as the main operation is computing the scalar product of the i–th row of L (except the diagonal element) with the vector of the components already computed.

The second algorithm uses the fact that after a component is computed, the right hand side can be modified at once,

```
for j=1:n
  x(j)=b(j)/l(j,j)
  for i=j+1:n
    b(i)=b(i)-l(i,j)*x(j)
  end
end
```

We called this algorithm the Saxpy algorithm as the main loop is clearly a Saxpy operation, $x(j)$ being the scalar. Notice that this corresponds to swapping the two loops of the algorithm.

We will first examine parallel algorithms where the data is distributed either by rows or by columns. Then, the important issue is the mapping of rows and columns to processors' memories. We define this mapping by `map(j)`: the number (or address) of the processor to which row (or column) `j` is mapped.

The most commonly used mapping is wrapping. We suppose $n >> p$ where p is the number of processors. For simplifying purposes, suppose that p divides n exactly. The simplest way of defining a wrap mapping is the following,

$$\begin{pmatrix} \texttt{j}: & 1 & 2 & 3 & \dots & p & p+1 & \dots & 2p & 2p+1 & \dots & n \\ \texttt{map(j)}: & 1 & 2 & 3 & \dots & p & 1 & \dots & p & 1 & \dots & p \end{pmatrix}$$

The advantage of this mapping is its simplicity. However, a potential problem is that each processor does not receive the same number of matrix entries, particularly if only the lower triangular part of L is stored. Suppose we distribute the (non zero) elements of columns. Processor 1 will have

$$\sum_{i=0}^{n/p-1} n - ip = \sum_{i=1}^{n/p} n - (i-1)p = (n+p)\frac{n}{p} - \frac{p}{2}\frac{n}{p}\left(\frac{n}{p}+1\right) = \frac{n^2}{2p} + \frac{n}{2},$$

elements while processor p will receive

$$\sum_{i=1}^{n/p} n - ip + 1 = (n+1)\frac{n}{p} - \frac{n}{2}\left(\frac{n}{p}+1\right) = \frac{n^2}{2p} + \frac{n}{p} - \frac{n}{2}.$$

This can cause some load imbalance. The problem can be fixed (in this simple case) by reflecting the mapping in the following way:

$$\begin{pmatrix} \texttt{j}: & 1 & 2 & 3 & \dots & p & p+1 & \dots & 2p & 2p+1 & \dots & n \\ \texttt{map(j)}: & 1 & 2 & 3 & \dots & p & p & \dots & 1 & 1 & \dots & \dots \end{pmatrix}$$

One can easily check that in two consecutive sets of indices, each processor has the same number of elements. Therefore, if $\frac{n}{p}$ is even, each processor receives the same number of entries.

These mappings can be generalized by considering blocks of consecutive rows or columns instead of individual rows or columns. Doing so decreases communication time but increases load imbalance. Methods using these mappings are called panels methods in Rothberg's Ph.D. thesis (Rothberg [1993]).

Fan out and fan in algorithms

These two algorithms seek the parallelism in the inner loops of the Saxpy and the scalar product algorithms. We consider first the Saxpy algorithm. Clearly, the components of b can be computed in parallel. Each component b_i is going to be computed by one particular processor. To be able to achieve this in parallel, b_i must be in the memory of the processor computing this entry. Therefore, the data must be distributed by rows.

Suppose that each processor has a set { `myrows` } containing the indices of rows the memory of the processor is storing. As soon as the x_j component of the solution is computed, it must be broadcasted to all other processors. This is done in a fan–out operation. `Fan-out(x,proc)` means that processor `proc` sends `x` (located in its memory) to all other processors. The algorithm (the code running on one processor) is the following, (Heath, Romine [1988])

```
for j=1:n
  if j ∈ { myrows }
    x(j)=b(j)/l(j,j)
    fan-out(x(j),map(j))
  end
  for i ∈ { myrows }
    b(i)=b(i)-l(i,j)*x(j)
  end
end
```

This code is not exactly the one that will really be used on a parallel machine as it uses a global indexing scheme. In a real code indices local to each processor may have to be used, depending on the programming model. Nevertheless, this code helps understanding what is going on.

The implementation of the fan–out (broadcast) operation depends on the computer architecture, particularly the topology of the communication network. It provides the necessary synchronization as one processor sends data and all the others wait to receive it. One problem is that only one word (x_j) is sent at a time. Usually, sending a message of l words costs

$$t = t_0 + \tau l,$$

t_0 is the start–up time (or latency). The efficiency depends on the value of t_0 relatively to τ and l. For sending only one word, the cost is essentially the start–up time. Therefore, this algorithm can be efficient only on computers with a small latency.

Now, we look at the scalar product algorithm. Parallelism is found in the inner loop, computing the scalar product. To be able to do so, the data has to be

distributed by columns. Then, each processor can compute the `l(i,j)*x(j)` term for `j` in its column index set { `mycolumns` }. These partial contributions must be added to the ones of the other processors. This is done in the fan–in operation: `Fan-in(x,proc)` means that processor `proc` receives the sum of all the `x`'s over all processors. Again, the implementation of this operation depends on the computer architecture. One naive way to do it, it that each processor sends its contribution to processor 0 that does the summation and then, broadcast the result to all processors. However, depending on the architecture, there are much more efficient ways to implement this operation. Generally, one can obtain an $\log_2 n$ computational time.

Wavefront algorithms

Fan–in and fan–out algorithms are seeking parallelism in the inner loop. The algorithms in this Section look for parallelism in the outer loop. However, here some data dependencies have to be respected, e.g. x_{i-1} has to be computed before x_i.

Consider the Saxpy algorithm with data distributed by columns this time. Informally, the algorithm is the following. Let y be an n–vector, processor `map(1)` computes x_1 and starts computing the updates $y_i = l_{i,1}x_1$. After computing σ such components ($1 \leqslant \sigma \leqslant n-1$) that we called a segment, processor `map(1)` sends them to `map(2)` and resumes computing the next σ update components. As soon as the data is received, processor `map(2)` computes x_2 and the updates with x_2 in the first segment. When it is completed, it is send to processor `map(3)`, etc...

Having completed the update for a segment, `map(j)` sends it to `map(j+1)`. After a start–up phase, all processors are working concurrently. A smaller segment size increases parallelism particularly at the beginning, but increases also the number of messages required. A value of $\sigma = n-1$ gives a purely serial algorithm.

The same idea can be applied to the scalar product algorithm with the data distributed by rows.

Cyclic algorithms and variations

The algorithms in the two previous Sections can be used with any mapping of rows or columns although this choice will have an impact on performance. In this Section, following Heath and Romine [1988], we study an algorithm that has been introduced by Li and Coleman [1989] to exploit the wrap mapping and to reduce the communication volume.

Consider the Saxpy algorithm with the columns distributed using the wrap mapping on p processors. A segment of size $p-1$ passes from processor to processor and accumulates all the necessary updates. At step j, processor `map(j)` receives the segment from processor `map(j-1)` and uses its first element to compute x_j. It deletes the first element, updates the other components and appends a new element to the segment. The processor `map(j)` sends the new segment to processor `map(j+1)` and starts computing update components involving x_j that will be used when the segment will return to `map(j)` according to the wrap mapping.

```
for i=1:n
  t=0
```

```
    for j ∈ { mycolumns } & j<i
      t=t+l(i,j)*x(j)
    end
    s=fan-in(t,map(i))
    if i ∈ { mycolumns }
      x(i)=(b(i)-s)/l(i,i)
    end
  end
```

These algorithms have several shortcomings. The main one is that they do not exploit all the parallelism that can be found in the problem. For example, take the Saxpy algorithm. After $x(j)$ has been received, each processor updates all of its b components. However, after b_{j+1} has been updated, processor `map(j+1)` can immediately compute x_{j+1} and start sending it to everybody else before updating the other components of b.

Similar modifications can be made to the scalar product algorithm. However, complete asynchronism is difficult to implement. The data dependencies of the classical forward sweep are illustrated in Figure 5 on a 5×5 example where an arrow indicates that the computation where it started from must be completed before the computation where it is pointing to can take place. It illustrates the data flow of the algorithm.

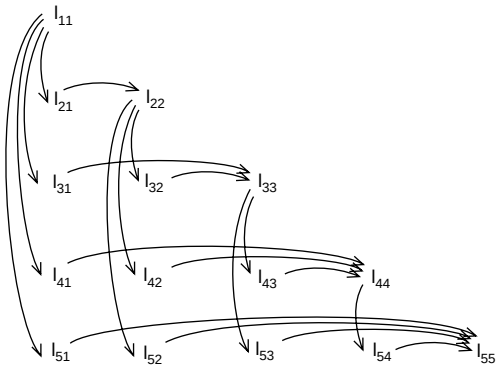

fig 14.1

As stated in Heath and Romine [1988], the efficiency of the previous algorithm depends on how much computation can be done in the time between successive appearances of the segment in a processor.

Numerical experiments in Heath and Romine [1988] show that for a given problem size, when increasing the number of processors, there is a point where the computer time increases rather than decreases.

Cyclic algorithms have been improved in this respect in Eisenstat, Hath, Henkel and Romine [1988]. One obvious modification is to use ideas from the wavefront

algorithm and break the segment into several subsegments. A subsegment is broadcasted to the next processor as soon as possible. In this modification, there are more messages to be sent but the communication pattern is still a ring–like one. This is called a pipelined cyclic algorithm.

Another modification is to send the subsegments not anymore to `map(j+1)`, but directly to `map(k)` where `k` is the index of the first element in the subsegment. Of course, additional synchronizations need to be done to be sure that processor `map(k)` has received all the necessary information before proceeding to compute x_k. Another variant is not using equally sized subsegments but instead having subsegments of size $2, 4, \ldots, \frac{p}{2}$ and one of size 1 sent to the predecessor in the ring.

Numerical experiments comparing these different algorithms are given in Eisenstat and al [1988].

Other algorithms

The algorithms in the previous Sections are parallel implementations and variations of sequential algorithms. They have the same number of floating point operations, although they are processed in a different order.

However, one can imagine other ways of computing the solution of $Lx = b$. Some are based on computing the inverse of L by various means. The following algorithm is due to Sameh and Brent [1977].

Suppose for simplicity that $l_{i,i} = 1$. Then, it is easy to see that

$$L = \prod_{i=1}^{n-1} N_i^{-1},$$

where $N_i^{-1} = I + l_i e_i^{\mathrm{T}}$ with

$$l_i = \begin{pmatrix} 0 \\ 0 \\ l_{i,i+1} \\ \vdots \\ l_{n,i} \end{pmatrix}.$$

By Lemma 3.5, we know that

$$N_i = I - l_i e_i^{\mathrm{T}}.$$

Therefore,

$$x = L^{-1}b = \prod_{i=n-1}^{1} N_i b.$$

The product on the right hand side can be computed by a logarithmic algorithm. Suppose $n = 8$, then

$$x = N_7 N_6 N_5 N_4 N_3 N_2 N_1 b.$$

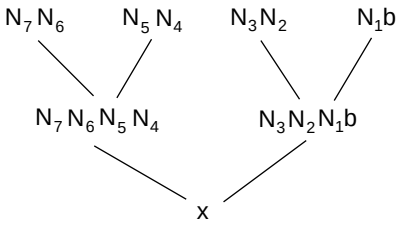

fig 14.2

This can be written as

$$x = (N_7N_6)(N_5N_4)(N_3N_2)(N_1b).$$

The products within each parenthesis can be formed in parallel, the results combined again two by two, etc...

This algorithm requires $\log_2 n$ stages. However, the total number of floating point operations is about $\frac{n^3}{10}$, that is much larger than the usual $O(n^2)$ operations of the serial algorithm. For this algorithm to be usable, a very large number of processors is required.

Another way of writing the matrix is

$$L = I - \bar{L},$$

where $\bar{L}$ is strictly lower triangular and therefore $\bar{L}^n = 0$. Then,

$$\begin{aligned} x &= (I - \bar{L})^{-1}b, \\ &= (I + \bar{L} + \bar{L}^2 + \ldots + \bar{L}^{n-1})b, \\ &= (I + \bar{L}^{2n-1})(I + \bar{L}^{2n-2}) \ldots (I + \bar{L})b. \end{aligned}$$

Again, this can be evaluated in $\log_2 n$ steps.

Finally, another algorithm is obtained by partitioning the matrix L as

$$L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}.$$

Then,

$$L^{-1} = \begin{pmatrix} L_{1,1}^{-1} & 0 \\ -L_{2,2}^{-1}L_{2,1}L_{1,1}^{-1} & L_{2,2}^{-1} \end{pmatrix}.$$

The process can be reiterated in parallel for $L_{1,1}$ and $L_{2,2}$ and so on. Then, if $x = (x_1, x_2)^{\mathrm{T}}$ and $b = (b_1, b_2)^{\mathrm{T}}$,

$$\begin{aligned} x_1 &= L_{1,1}^{-1}b_1, \\ x_2 &= L_{2,2}^{-1}b_2 + L_{2,2}^{-1}L_{2,1}L_{1,1}^{-1}b_1. \end{aligned}$$

15. LU factorization on distributed memory computers

A very large number of papers have been written over the years on the parallelization of the LU factorization algorithms. Actually, it is easier to exhibit parallelism in the LU factorization than in the triangular solves. It can be easily seen that in the variants of LU factorization we have studied, there is a phase where some elements are modified and that these modifications are independent of each other, giving rise to some parallelism. A problem that we had not to face with triangular solves is pivoting and we will see that it can cause some troubles.

There are a lot of variants that can be thought of for the parallel LU factorization. Here, we first follow the exposition of Geist and Romine [1988].

As for the triangular solves, we consider the matrix to be distributed by rows or columns. Notice that, for a better efficiency, this distribution has to be consistent with the one used for the triangular solve. Otherwise, some data redistribution has to be done between the two phases. We use partial pivoting on the kij form of the factorization.

Row storage scheme

Suppose first that the matrix is stored by rows. A possible algorithm is the following.

```
for k=1:n-1
  find pivot row r
  if r ∈ { myrows }
    broadcast pivot row
  else
    receive pivot row
  end
  for i>k & i ∈ { myrows }
    m(i,k)=a(i,k)/a(k,k)
    for j=k+1:n-1
      a(i,j)=a(i,j)-m(i,k)*a(k,j)
    end
  end
end
```

Analyzing the communications, we see that as the k–th column is scattered amongst processors, communication must take place between processors to find which row is the pivot row. The way it is done depends on the computer architecture. Note that in the previous algorithm –which is denoted by RSRP (Row Storage Row Pivoting) in Geist and Romine [1988]– no explicit exchange of rows takes place and this can cause some load imbalance depending on the pivot choices. Chu and George [1987] proposed to use explicit exchange of rows with a wrap mapping. Obviously, this will produce a communication overhead and the question is to know if this is offset by an improved load balancing. Examples in Geist and Romine [1988] demonstrate that, in some cases, it is beneficial to use explicit row exchanges.

Geist and Romine show how to decrease the number of communications by relaxing the requirement that the final distribution of rows be a wrap mapping. They ask that rows kp through $(k+1)p-1$ lie in distinct processors for each k. A processor that contains one of these pivot rows cannot have another one and otherwise must exchange rows with a processor that does not contain one. Different strategies for choosing the processor to exchange the row with are given in Geist and Romine [1988].

Column storage scheme

This algorithm is denoted by CSRP (Column Storage Row Pivoting) in Geist and Romine [1988]. Here, updating the matrix is done by columns (kji form) instead of being done by rows in RSRP. However, the main differences are when computing the multipliers and searching for the pivot row. As the coefficient matrix is stored by columns, the computation of the multipliers is done serially by the processor owning the pivot column. This, of course, reduces the parallel efficiency of the factorization. On the other hand, finding the pivot row is done without communication by only one processor. Moreover, partial pivoting has no incidence on load balancing as the mapping of columns to processors is not changed by pivoting.

```
for k=1:n-1
  if k ∈ { mycolumns }
    find pivot row r
    for i=k+1:n
      m(i,k)=a(i,k)/a(k,k)
    end
    broadcast m and pivot index
  else
    receive m and pivot index
  end
  for j> k & j ∈ { mycolumns }
    for i=k+1:n
      a(i,j)=a(i,j)-m(i,k)*a(k,j)
    end
  end
end
```

Numerical experiments in Geist and Romine [1988] show that the results of CSRP as defined before are worst than those of RSRP. This is essentially due to the serial part of the algorithm.

A way to improve this algorithm is to use some form of pipelining. For instance, the processor containing column k can compute some multipliers and send them allowing some of the other processors to start their computations. Using these modifications CSRP is competitive with RSRP.

Block storage scheme

In Dongarra and Walker [1993], it is proposed to use a block cyclic data distri-

bution and block partitioned versions of LU factorization for distributed memory parallel computers. In the approach of Dongarra and Walker, all the parallelism is to be find at the level of the BLAS routines, implying that the top layers of the source code of the parallel version look very similar to the ones of LAPACK.

Independent data distributions are used for rows and columns. An object m (a piece of row or column) is mapped to a couple (p, i), p being the processor number and i the location in the local memory of this processor. By using wrapping as before, we have

$$m \longrightarrow (m \bmod p, \lfloor m/p \rfloor).$$

Blocking consists of assigning contiguous entries to processors by blocks,

$$m \longrightarrow (\lfloor m/L \rfloor, m \bmod L), \quad L = \lceil m/p \rceil.$$

The block cyclic distribution is a combination of both. Blocks of consecutive data are distributed by wrapping,

$$m \longrightarrow (q, b, i),$$

q is the processor number, b the block number in processor q and i the index in block b. If there are r data objects in a block, then

$$m \longrightarrow (\lfloor \frac{m \bmod T}{r} \rfloor, \lfloor \frac{m}{T} \rfloor, m \bmod r), \quad T = rp.$$

A slight generalization of this is given in Dongarra and Walker [1993] by adding offsets to the processor number.

To distribute the matrix, independent block cyclic distributions are applied for the rows and columns. The processors are supposed to be (logically) arranged in a two dimensional mesh and referred by couples (q_1, q_2). For general data distributions, communications are required for the pivot search and the computation of the multipliers. The communications to be done are broadcast to all processors and broadcast to all processors in the same row (or column) in the 2D mesh of processors.

Of course, the logical arrangement of processors has to be mapped to the physical layout. This is architecture dependent. In Dongarra and Walker [1993], experiments are reported on the Intel Delta computer.

Dense block–oriented factorizations are also studied in Rothberg's Ph.D. thesis (Rothberg [1993]). In his approach, if there are p processors arranged in a 2D mesh, block (i, j) of the matrix is mapped to processor $i \bmod \sqrt{p}, j \bmod \sqrt{p}$. Then, two variants are studied; the first one is a destination–computes approach where all updates for a block are computed on the processor that owns the destination block. The second one is a source–computes approach where updates are computed by a processor that owns one of the source blocks. It is shown that the first approach gives better results. Details can be found in Rothberg [1993].

4. Gaussian elimination for sparse linear systems

16. Introduction

An area that has seen a rapid development since the end of the 60's is research on sparse matrices. So far, in the previous chapters, we have addressed several properties of the matrices we considered like symmetry or positive definiteness, but we did not care if some of the matrix entries where zero or not.

Now, we are going to look at this possibility in details. A sparse matrix is one with many zero entries. However, it is difficult to give a precise definition of what a sparse matrix is and to say how many zeroes must be there or even what percentage of zeroes we should have.

We will see later on that special techniques are used to store sparse matrices and that special algorithms are going to be defined in order to minimize the storage and the number of operations during Gaussian elimination. Therefore, a definition that has sometimes been given is that a matrix is sparse when it pays (either in computer storage or in computer time) to use these special sparse techniques as opposed to the more traditional dense (or general) algorithms that we described before.

Nevertheless, exploiting sparsity allows solving very large problems with even millions of unknowns by 1996.

There are several good books about sparse linear systems. Let us mention those of George and Liu [1981] for symmetric positive definite systems and Duff, Erisman and Reid [1986] for more general systems.

A drawback of these sparse techniques is that they are quite complex (actually more complex than dense algorithms) and sometimes difficult to optimize. Therefore, it is usually not feasible for the average user to write a sparse code from scratch. Fortunately, there exist some good packages available containing well tuned codes like the Harwell Library, Duff and Reid [1993], or SPARSPAK, George and Liu [1981], to mention just a few.

In this part about sparse matrices, we are going first to introduce some definitions and to stress some basic facts. We will look at the case of symmetric and particularly positive definite matrices as it is a little easier to introduce some of the techniques in this framework. Then, we will move on to the general case of non symmetric sparse matrices.

17. Basic storage schemes and fill–in

Storage schemes

Dealing with sparse matrices, our aim is to be able to avoid storing the zero entries of the sparse matrix A and to avoid doing operations on these zeroes. The most natural way to do this is to store only the non zero entries $a_{i,j}$ of A, together with the row and column indices i and j. Therefore, if nz is the number of non zeroes of A, the storage needed is nz floating point numbers and 2 nz integers. In most modern computers, integers use the same number of bits as floating point numbers.

In that case, the total storage is 3 nz words.

However, this mode of storage (which is sometimes called the coordinate scheme) is not very convenient for Gaussian elimination as we have seen that most variants of the method require easy accesses to rows and/or columns of the matrix.

One common way to store a sparse matrix which is more suited to Gaussian elimination is to hold the non zeroes of each row (resp. column) as a packed sparse vector `AA`, together with the column (resp. row) index of each element in a vector `JA`. These two vectors have a length of nz words. A third vector `IA` of integers of length $n + 1$ is needed to point to the beginning of each row in `AA` and `JA`. Let us look at this storing scheme on a small example.

Let

$$A = \begin{pmatrix} a_1 & 0 & 0 & a_2 \\ a_3 & a_4 & a_5 & 0 \\ 0 & a_6 & a_7 & 0 \\ a_8 & 0 & 0 & a_9 \end{pmatrix},$$

the stored quantities are

	1	2	3	4	5	6	7	8	9
`AA`	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
`JA`	1	4	1	2	3	2	3	1	4
`IA`	1	3	6	8	10				

Notice that `IA(n+1)=nz+1`. This allows to compute the length of row i by `IA(i+1)-IA(i)`. Equivalently, the lengths of rows can be stored in place of `IA`.

Sometimes, something special is done for the diagonal elements. As often, they are all non zero, they are stored in a special vector of length n or the diagonal entry of each row could be stored in the first (or last) position of the row. If the matrix is symmetric only the lower or upper part is stored.

Another storage scheme which is much used is linked lists. Here, each non zero element (together with the column index) has a pointer `IPA` to the location of the next element in the row. To be able to add or delete entries easily in the list, it is sometimes handy to have also a second pointer to the location of the previous entry. Finally, we must know the beginning of the list for each row in a vector `IA`. Going back to our small example, we have the following storage when using only forward links. Notice the elements could be stored in any order,

	1	2	3	4	5	6	7	8	9
`AA`	a_3	a_2	a_9	a_1	a_4	a_8	a_6	a_5	a_7
`JA`	1	4	4	1	2	1	2	3	3
`IPA`	5	0	0	2	8	3	9	0	0
`IA`	4	1	7	6					

Notice that a value of zero in `IPA(.)` indicates the end of the list for the given row.

Later on, we will see other schemes for storing sparse matrices that are more dedicated to some particular algorithms. Some other storage schemes exist that are

more suited to iterative methods, specially those which need only matrix vector products. However, whatever the choice is for the storage scheme, it must be able to deal with the phenomenon of fill–in.

The fill–in phenomenon

Remember that at the k–th step of the algorithm, we have to compute

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)} a_{k,j}^{(k)}}{a_{k,k}^{(k)}}.$$

Therefore, even if $a_{i,j}^{(k)} = 0$, $a_{i,j}^{(k+1)}$ can be non zero if $a_{i,k}^{(k)} \neq 0$ and $a_{k,j}^{(k)} \neq 0$. Non zero entries in the L and U factors in positions (i,j) for which $a_{i,j} = 0$ are called the fill–ins.

Consider the small example below where the matrix is symmetric and the x's stand for the non zero entries,

$$A = \begin{pmatrix} x & x & 0 & x & 0 \\ x & x & x & 0 & 0 \\ 0 & x & x & 0 & x \\ x & 0 & 0 & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix}.$$

We look at the different steps of Gaussian elimination where fill–ins are denoted by $\bullet$,

$$A_2 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & x & x & 0 & x \\ 0 & \bullet & 0 & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix},$$

$$A_3 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & \bullet & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix}.$$

Notice that the fill–in in position $(4,3)$ has been created by the fill–in in position $(4,2)$ at the previous step,

$$A_4 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & 0 & x & \bullet \\ 0 & 0 & 0 & \bullet & x \end{pmatrix},$$

$$A_5 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & 0 & x & \bullet \\ 0 & 0 & 0 & 0 & x \end{pmatrix},$$

and

$$L = \begin{pmatrix} x & & & & \\ x & x & & & \\ 0 & x & x & & \\ x & \bullet & \bullet & x & \\ 0 & 0 & x & \bullet & x \end{pmatrix}.$$

In this example, three elements which were initially zero in the lower triangular part of A are non zero in L.

The aim of sparse Gaussian elimination is to avoid doing operations on zero entries and therefore to try to minimize the number of fill–ins. This will have the effect of both minimizing the needed storage and the number of floating point operations.

It is clear that the number of fill–ins depends on the way the pivots are chosen if pivoting is allowed. As the following well known example proves, there can be large differences in the number of fill–ins with different pivoting strategies.

Consider

$$A = \begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix}.$$

Then,

$$L = \begin{pmatrix} x & & & \\ x & x & & \\ x & \bullet & x & \\ x & \bullet & \bullet & x \end{pmatrix},$$

that is, all the zero entries fill. If we define a permutation matrix P such that the first element is numbered last, then

$$PAP^{\mathrm{T}} = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & x & x \end{pmatrix}.$$

In this case, there is no fill–in at all. This is called a perfect elimination.

The way the fill–in problem is handled depends on the properties of the matrix. If the matrix is non symmetric (and without any special properties), we have seen that

we generally need to pivot to have an acceptable numerical accuracy. If in addition the matrix is sparse, we now have another requirement which is to minimize the fill–in. Therefore, these two goals have to be dealt with at the same time. Moreover, this implies that the data structure for the LU factors cannot be set up before the numerical factorization as the pivot rows and therefore the potential fill–in are only known when performing the numerical factorization.

If the matrix is symmetric and, for instance, positive definite, we do not need to pivot for numerical stability. Therefore, we have the freedom to choose symmetric permutations only to minimize the fill–in. Moreover, the number and indices of fill–ins can be determined before doing the numerical factorization as this depends only on the structure of the matrix. Hence, everything can be handled within a static data structure built in a preprocessing phase called the symbolic factorization.

Finding an ordering that minimizes the fill–in is an NP complete problem, see Yannakakis [1981]. Hence, we will have to rely on heuristics that can be computed quickly.

18. Definitions and graph theory

Basic definitions

It is well known that a graph can be associated with every matrix. For a general non symmetric square matrix A of order n, we associate a directed graph (or digraph). A digraph is a couple $G = (X, E)$ where X is a set of nodes (or vertices) and E is a set of directed edges. For a given matrix A of order n, there are n nodes and there is a directed edge from i to j if $a_{i,j} \neq 0$. Usually, self loops corresponding to $a_{i,i} \neq 0$ are not included.

Let

$$A = \begin{pmatrix} x & x & 0 & x \\ 0 & x & 0 & 0 \\ x & 0 & x & 0 \\ 0 & x & x & x \end{pmatrix}.$$

Then, the associated digraph is given in figure 18.1.

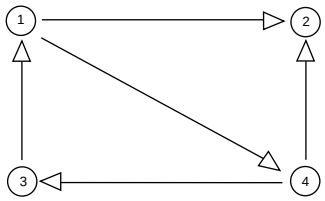

fig 18.1

Graphs are much more used in problems involving symmetric matrices. If $a_{i,j} \neq 0$, then $a_{j,i} \neq 0$. Therefore, we can look at undirected graphs and drop the arrows on the edges.

Let

$$A = \begin{pmatrix} x & x & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ 0 & x & x & x \end{pmatrix},$$

then, the graph of A is shown on figure 18.2.

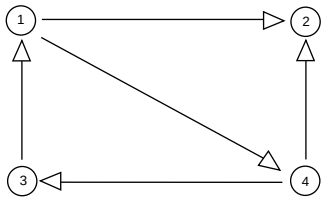

fig 18.2

Let us introduce a few definitions. Let $G = (X, E)$ be a (undirected) graph. We denote the nodes of the graph by x_i or sometimes i.

• $G' = (X', E')$ is a subgraph of G if $X' \subset X$ and $E' \subset E$.

• Two nodes x and y of G are adjacent if $\{x, y\} \in E$. The adjacency set of a node y is defined as

$$Adj(y) = \{x \in X |\ x \text{ is adjacent to } y\}.$$

If $Y \subset X$, then

$$Adj(Y) = \{x \in X |\ x \in Adj(y), x \notin Y, y \in Y\}.$$

• The degree of a node x of G is the number of its adjacent nodes in G,

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

• Let x and $y \in X$. A path of length l from x to y is a set of nodes $\{\nu_1, \nu_2, \ldots, \nu_{l+1}\}$ such that $x = \nu_1$, $y = \nu_{l+1}$ and $\{\nu_i, \nu_{i+1}\} \in E,\ 1 \leqslant i \leqslant l$. A path $\{\nu_0, \nu_1, \ldots, \nu_l, \nu_0\}$ is a (simple) cycle of length $l + 1$.

A graph is connected if for every $x, y \in X$, there exists a path from x to y. This corresponds to the matrix being irreducible.

A chord of a path is any edge joining two non consecutive vertices in the path. A graph is chordal if every cycle of length greater than three has a chord (Blair and Peyton [1993]).

• An important kind of graphs is when there is no closed paths. A particular node is labeled as the root. Then, there is a path from any node to the root. Such a

(connected) graph is called a tree. If it is not connected, we have a set of trees called a forest.

• Let $Y \subset X$, the section graph $G(Y)$ is a subgraph $(Y, E(Y))$ with

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

• A set $Y \subset X$ is a separator for G (a connected graph) if $G(X/Y)$ has two or more connected components.

• The distance $d(x, y)$ between two nodes x and y of G is the length of the shortest path between x and y.

The eccentricity of a node $e(x)$ is

$$e(x) = \max\{d(x, y) | y \in X\}.$$

The diameter δ of G is

$$\delta(G) = \max\{e(x) | x \in X\}.$$

A node x is peripheral if $e(x) = \delta(G)$.

• A clique is a subset of nodes such that they are all pairwise connected.

• A level structure of a graph G is a partition $\mathcal{L} = \{L_0, L_1, \ldots, L_l\}$ of X such that

$$\begin{aligned} &Adj(L_i) \subset L_{i-1} \cup L_{i+1},\ i = 1, \ldots, l-1, \\ &Adj(L_0) \subset L_1, \\ &Adj(L_l) \subset L_{l-1}. \end{aligned}$$

Note that each L_i is a separator for G. For each node $x \in X$, a level structure $\mathcal{L}(x)$ can be defined as

$$\mathcal{L}(x) = \{L_0(x), \ldots, L_{e(x)}(x)\},$$

$$\begin{aligned} &L_0(x) = \{x\} \\ &L_i(x) = Adj(\cup_{k=0}^{i-1} L_k(x)),\ 1 \leqslant i \leqslant e(x) \end{aligned}$$

where $e(x)$ is the eccentricity of x. The width of a level structure $\mathcal{L}(x)$ is

$$w(x) = \max\{|L_i(x)|, 0 \leqslant i \leqslant e(x)\}.$$

Characterization of the fill–in for a symmetric structure

Now, we turn to looking at the interpretation of Gaussian elimination in terms of graphs. This was first studied in Parter [1961], see also Rose [1970]. We consider a sequence of graphs $G^{(i)}$, $G^{(1)} = G$ corresponding to the different steps of the elimination on a matrix with a symmetric pattern.

Theorem 18.1. *$G^{(i+1)}$ is obtained from $G^{(i)}$ by removing the node x_i from the graph as well as all its incident edges and adding edges such that all the remaining neighbors of x_i in $G^{(i)}$ are pairwise connected.*

Proof. It is enough to look at the first step, eliminating the node x_1 (or the corresponding unknown in the linear system). Then,

$$a_{i,j}^{(2)} = a_{i,j} - \frac{a_{i,1}a_{1,j}}{a_{1,1}}.$$

$a_{i,j}^{(2)}$ is non zero if either $a_{i,j} \neq 0$ or, $a_{i,j} = 0$ and $a_{i,1}$ and $a_{1,j}$ are non zero. The last possibility translates into x_i and x_j being neighbors of x_1 in the graph. When x_1 is eliminated, they will be connected by an edge representing the new element $a_{i,j}^{(2)} \neq 0$. This occurs for all the neighbors of x_1. We did not consider zeroes that arise by cancellation. In this way, $G^{(2)}$ is obtained corresponding to the submatrix obtained from $A^{(2)}$, by deleting the first row and column. □

Taking the graph $G(A)$ of A and adding all the edges that are created in all the $G^{(i)}$s during the elimination, we obtain $G_F, (F = L + L^{\mathrm{T}})$ which is called the filled graph $G_F = (X, E^F)$. An example of elimination graphs is given below. Let

$$A = \begin{pmatrix} x & x & x & x & & & x \\ x & x & & & & & \\ x & & x & x & & & \\ x & & x & x & x & x & x \\ & & & x & x & & \\ & & & x & & x & x \\ x & & & x & & x & x \end{pmatrix},$$

then, figure 18.3 displays the graph $G(A)$. The graph $G^{(2)}$ is given in figure 18.4.

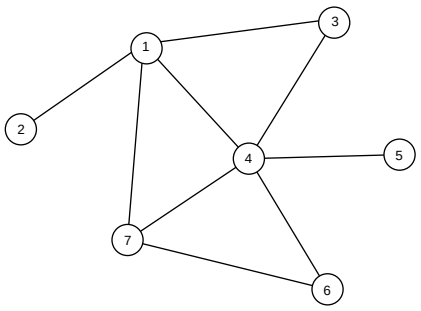

fig 18.3

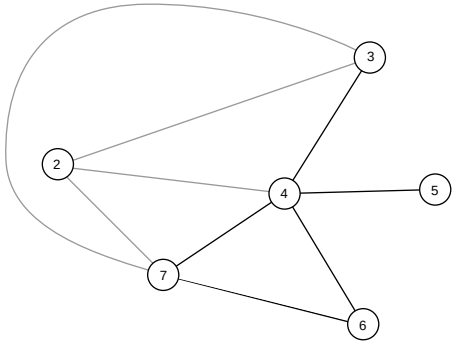

fig 18.4

Here the edges corresponding to fill–ins are denoted by grey lines. The graph $G^{(2)}$ corresponds to the matrix,

$$A^{(2)} = \begin{pmatrix} x & x & x & x & & & x \\ x & x & \bullet & \bullet & & & \bullet \\ x & \bullet & x & x & & & \bullet \\ x & \bullet & x & x & x & x & x \\ & & & x & x & & \\ & & & x & & x & x \\ x & \bullet & \bullet & x & & x & x \end{pmatrix}.$$

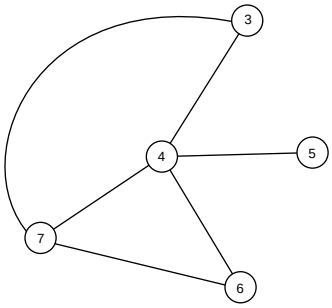

fig 18.5

Then, the elimination of x_2 does not cause any fill–in as all its neighbors form already a clique. $G^{(3)}$ is given in figure 18.5 and $A^{(3)} \equiv A^{(2)}$, $\equiv$ meaning that the two matrices have the same structure. The next step is to eliminate x_3. Here, again, there is no fill–in as x_4 and x_7 are already connected. $G^{(4)}$ is displayed on figure 18.6 and $A^{(4)} \equiv A^{(3)}$. Elimination of x_4 connects x_5, x_6 and x_7. $G^{(5)}$ is shown on figure 18.7.

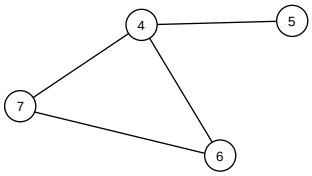

fig 18.6

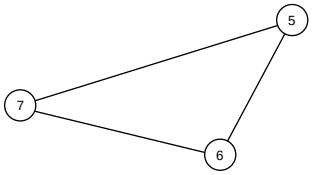

fig 18.7

$$A^{(5)} = \begin{pmatrix} x & x & x & x & & & x \\ x & x & \bullet & \bullet & & & \bullet \\ x & \bullet & x & x & & & \bullet \\ x & \bullet & x & x & x & x & x \\ & & & x & x & \bullet & \bullet \\ & & & x & \bullet & x & x \\ x & \bullet & \bullet & x & \bullet & x & x \end{pmatrix}.$$

$G^{(5)}$ is a clique, the corresponding 3×3 submatrix is dense thus there will be no other fill–in before the end of the elimination and $A^{(7)} \equiv A^{(5)}$. The filled graph is given in figure 18.8.

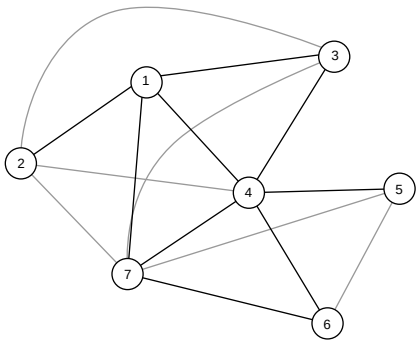

fig 18.8

In total there are six fill–ins in the elimination. An interesting point is to remark that the (position of) the fill–in entry created between x_3 and x_7 by the elimination of x_1 would have been created anyway by the elimination of x_2.

The number of fill–ins depends of the order of elimination, that is on the numbering of the vertices in the graph (or of the unknowns in the linear system) or otherwise said on the pivoting sequence.

For instance, in our example, at the beginning we can eliminate x_2 and x_5 as they have only one neighbor and their elimination do not create any fill–in. This gives the graph in figure 18.9.

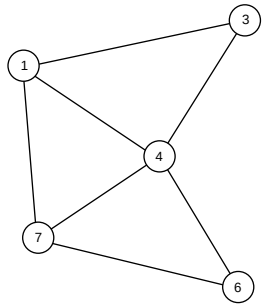

fig 18.9

Now, x_3 and x_6 can be eliminated without fill–in as they are already forming a clique with their neighbors. After eliminating x_3 and x_6, the situation is shown on figure 18.10.

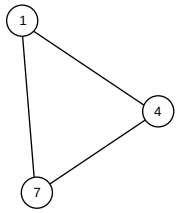

fig 18.10

Again, this is a clique and the nodes can be eliminated in any order without fill–in.

Therefore, we have just seen that for this example, there are several numberings that lead to a perfect elimination without any fill–in. For instance,

$$2, \quad 5, \quad 3, \quad 6, \quad 1, \quad 4, \quad 7$$

The permuted matrix is

$$A' = PAP^{\mathrm{T}} = \begin{pmatrix} x & & & & x & & \\ & x & & & & x & \\ & & x & & x & x & \\ & & & x & & x & x \\ x & & x & & x & x & x \\ & x & x & x & x & x & x \\ & & & x & x & x & x \end{pmatrix}.$$

Of course, this is not a general situation. One thing that we must realize is that the more fill–ins we create in the early stages, the more fill-ins we will get later on. Thus, an heuristic rule is that it is likely to be beneficial to start by eliminating nodes that do not create much fill–in. These are the nodes with a small number of neighbors or nodes in cliques.

Let us now introduce a few more definitions.

• The elimination tree of A, symmetric matrix of order n, is a graph with n nodes such that the node p is the parent of node j, if and only if

$$p = \min\{i | i > j,\, l_{i,j} \neq 0\}$$

where L is the Cholesky factor of A. The elimination tree of A will be denoted by $T(A)$ or simply T if the context makes it clear that we refer to A.

Clearly, p is the index of the first non zero element in column j of L. For the previous example without renumbering, the elimination tree $T(A)$ is simply a chain shown on figure 18.11.

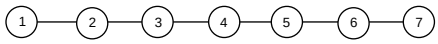

fig 18.11

$T(A')$ is given in figure 18.12 (renumbering the unknowns according to P).

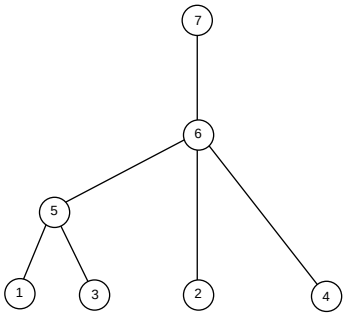

fig 18.12

This exhibits the fact that x'_1, x'_2, x'_3, x'_4 (corresponding to x_2, x_5, x_3, x_6 in the initial ordering) can be eliminated in any order (or even in parallel) as there are no dependencies between these variables.

It is of course interesting to know if there will be some fill–in between two nodes of $G(A)$ before doing the elimination. This has been studied by George some time ago, see George and Liu [1981] for a detailed exposition. It can be understood intuitively in the following way.

We have seen that there can be a fill–in between x_j and x_k only if at some step m, they are not already connected together and both neighbors of $x_m, m < j$, $m < k$. Therefore, either they were already neighbors of x_m in G or they were put in this situation by the elimination of other nodes $x_l, l < m$.

Recursively, we see that at some stage, x_j was a neighbor of one of these nodes and the same for x_k with another of these nodes. This means that in G, there is at least one path between x_j and x_k and that all nodes on this path have numbers smaller than j and k. If there is no such path, there won't be a fill–in between x_j and x_k.

In our example and in the initial ordering, we see that x_2 and x_3 are linked by a path of length 2 through x_1. The same is true for x_2 and x_4 and also x_2 and x_7. No other node can be reached from x_2 in this way as nodes x_3, x_4 or x_7 are on the paths to the other nodes and they have numbers greater than 2. There will be a fill–in between x_3 and x_7 as they can be reached through x_1.

For the permuted matrix A', we have the graph of figure 18.13.

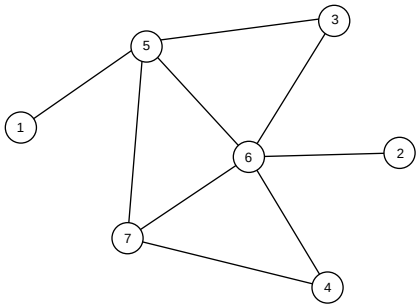

fig 18.13

It can be seen that on any path from one node to another one which is not a neighbor, there is always a node with a larger number. This discussion can be formalized in the following way.

• Let $S \subset X$ and $x \in X, x \notin S$, x is said to be reachable from $y \notin S$ through S if there exists a path $(y, v_1, \ldots, v_k, x)$ from y to x in G such that $v_i \in S$, $i = 1, \ldots, k$. We define

$$Reach(y, S) = \{x | x \notin S, x \text{ is reachable from } y \text{ through } S\}$$

Theorem 18.2 (George)**.** *Let $k > j$, there will be a fill–in between x_j and x_k if*

and only if

$$x_k \in Reach(x_j, \{x_1, \ldots, x_{j-1}\}).$$

We first prove a Lemma due to Parter [1961].

Lemma 18.1. *$\{x_i, x_j\} \in E^F$ if and only if $\{x_i, x_j\} \in E$ or $\{x_i, x_k\} \in E^F$ and $\{x_k, x_j\} \in E^F$ for some $k < \min\{i, j\}$.*

Proof. If $\{x_i, x_k\} \in E^F$ (filled graph) and $\{x_k, x_j\} \in E^F$ for some $k < \min\{i, j\}$, then the elimination of x_k will create a fill–in between x_i and x_j. Therefore, $\{x_i, x_j\} \in E^F$.

Conversely, if $\{x_i, x_j\} \in E^F$ and $\{x_i, x_j\} \notin E$, then we have seen in the previous discussion that at some stage, x_i and x_j must be neighbors of a node, say x_k, that is going to be eliminated before x_i and x_j. Thus, $k < \min\{i, j\}$. □

Proof of Theorem 18.2, see George (George and Liu [1981]), Rose, Tarjan and Lueker [1976].

Suppose $x_k \in Reach(x_j, \{x_1, \ldots, x_{j-1}\})$. There exists a path $\{x_j, v_1, \ldots, v_l, x_k\} \in G$ with $v_i \in \{x_1, \ldots, x_{j-1}\}$, $1 \leqslant i \leqslant l$. If $l = 0$ or $l = 1$, the result follows from Lemma 18.1. If $l > 1$, it is easy to show that $\{x_k, x_j\} \in E^F$ by induction.

Conversely, we assume $\{x_i, x_j\} \in E^F$, $j < k$. The proof goes by induction on j. For $j = 1$, $\{x_1, x_k\} \in E^F$ implies $\{x_1, x_k\} \in E$ as there is no fill–in with the first node. Moreover, the set $\{x_1, \ldots, x_{j-1}\}$ is empty.

Suppose the result is true up to $j-1$. By Lemma 18.1, there exists some $l \leqslant j-1$ such that $\{x_j, x_l\} \in E^F$ and $\{x_l, x_k\} \in E^F$. By the assumption, there exists a path between x_j and x_l and another one from x_l to x_k. Clearly, this implies that there is a path from x_j to x_k whose nodes have numbers $\leqslant l \leqslant j-1$. □

George (George and Liu [1981]) demonstrated that reachable sets can be implemented efficiently by using the notion of quotient graph.

• Let $\mathcal{P}$ be a partition of X,

$$\mathcal{P} = \{X_1, X_2, \ldots, X_p\}, \quad \cup_{k=1}^{p} X_k = X, \quad X_i \cap X_j = \emptyset \text{ if } i \neq j.$$

The quotient graph of G is $(\mathcal{P}, \mathcal{E})$, where $\{X_i, X_j\} \in \mathcal{E}$ if and only if

$$Adj(X_i) \cap X_j \neq \emptyset,\ i \neq j.$$

It is denoted by $G/\mathcal{P}$.

Let $S \subset X$, following George (George and Liu [1981]), we define a partitioning of X by

$$\mathcal{C}(S) = \{C \subset S | G(C) \text{ is a connected component in the subgraph } G(S)\}$$
$$\bar{\mathcal{C}}(S) = \{y | y \in X - S\} \cup \mathcal{C}(S)$$

$G/\bar{\mathcal{C}}(S)$ can be viewed as the graph obtained by amalgamating nodes in connected sets in S.

Let $S_i = \{x_1, x_2, \ldots, x_i\}$, $1 \leqslant i \leqslant n$. S_i induces a partitioning $\bar{\mathcal{C}}(S_i)$ and a quotient graph $\mathcal{G}_i = G/\bar{\mathcal{C}}(S_i)$. George proved the following result.

Theorem 18.3. $\forall y \in X - S_i$,

$$Reach_G(y, S_i) = Reach_{\mathcal{G}_i}(y, \mathcal{C}(S_i)).$$

The advantages of the quotient graph are
1) elimination graphs are easily obtained from quotient graphs
2) the quotient graph can be implemented in place. It requires no more space than the original graph structure.

Usually, the quotient graph structure is implemented via linked lists, see George and Liu [1981].

Characterization of the fill–in through elimination trees

In Liu [1990] the use of elimination trees in sparse factorization was reviewed. We will follow Liu's exposition to explain how elimination trees can be used to describe the fill–in.

Remark first that if the graph of a matrix A is a tree, then there exists a permutation P such that a perfect elimination is possible on PAP^{T}. A topological ordering of a rooted tree is defined as a numbering that orders children nodes before their parents.

The elimination tree $T(A)$ of a matrix A has, as we have seen, the same nodes as G and is a spanning tree of the filled graph G_F. Notice that $T(A)$ and $T(F)$ are identical.

We denote by $T[x]$, the subtree of $T(A)$ rooted at node x. $y \in T[x]$ is a descendant of x and x is an ancestor of y. From the definition of $T(A)$, we have that if x_i is a proper ancestor of x_j in $T(A)$, then $i > j$.

Theorem 18.4, [Liu]**.** *For $i > j$, the numerical values of columns i of L ($L_{*,i}$) depend on column j of L ($L_{*,j}$) if and only if $l_{i,j} \neq 0$.*

Proof. This is obvious from what we have seen before. □

In the filled graph G_F, we can use a directed edge from x_j to x_i to indicate that column i depends on column j. The result is a digraph of the Cholesky factor. From this digraph, Liu [1990] derived the transitive reduction: if there is a directed path greater than one from x_j to x_i and a directed edge from x_j to x_i, this last edge is removed. Removing all these redundant edges gives the transitive reduction.

The transitive reduction of the filled graph generates the elimination tree structure (Liu [1990]). This is a way to compute the elimination tree from G_F. Algorithms for determining the elimination tree structure are given in Liu [1990].

Moreover,

Theorem 18.5 (Schreiber [1982]). *If $i > j$ and $l_{i,j} \neq 0$, then x_i is an ancestor of x_j in $T(A)$.*

As a consequence, if $x_s \in T[x_i]$ and $x_t \in T[x_j]$ a disjoint subtree from $T[x_i]$, then $l_{s,t} = 0$. Now, we have a result related to Theorem 18.2

Theorem 18.6. *Let $i > j$, $l_{i,j} \neq 0$ if and only if there exists a path*

$$x_i, v_1, \ldots, v_k, x_j, \ v_i \in X$$

in $G(A)$ such that $\{v_1, \ldots, v_k\} \subseteq T[x_j]$.

The row structure of L is characterized in the following result.

Theorem 18.7 (Liu [1990]). *Let $i > j, l_{i,j} \neq 0$ if and only if x_j is an ancestor of some x_k in $T(A)$ such that $a_{i,k} \neq 0$.*

The column structure can be characterized as well.

Theorem 18.8 (Liu [1990]). *The structure of column j of L is given by*

$$Adj_G(T[x_j]) \cup \{x_j\} = \{x_i | l_{i,j} \neq 0, \ i \geqslant j\}.$$

Proof. This is a simple consequence of Theorem 18.7. □

Elimination trees are the basis for efficient implementations of symbolic factorization (Liu [1990]).

19. Band and envelope numbering schemes for symmetric matrices

Definitions
Most of the first attempts to exploit sparsity were considering band or envelope storage schemes and were trying to minimize the storage using these schemes. This can be explained after a few definitions.

- $f_i(A) = \min\{i | a_{i,j} \neq 0\}$.
  $f_i(A)$ is the index of the column with the first non zero element of row i.
- $\beta_i(A) = i - f_i(A)$ is the bandwith of row i. The bandwith of A is

$$\beta(A) = \max_i \{\beta_i(A), \ 1 \leqslant i \leqslant n\}$$

and

$$band(A) = \{(i,j) | 0 < i - j \leqslant \beta(A), \ i \geqslant j\}.$$

These notions have led to ideas for storing the matrices A and L. If $\beta_i(A)$ is almost constant as a function of i, then it makes sense to store the entries corresponding to all the indices in $band(A)$. However, most of the time this is not the case as they are a few rows with a larger bandwith than the other ones and too much storage is wasted by the band scheme. Then, one can use the variable band or envelope storage scheme, Jennings [1977].

• $Env(A) = \{(i,j) | 0 < i - j \leqslant \beta_i(A),\ i \geqslant j\}$ is the envelope of A. The profile of A, denoted by $P_r(A)$ is given by

$$P_r(A) = |Env(A)| = \sum_{i=1}^{n} \beta_i(A).$$

It is clear that a simple storage scheme is obtained by storing for each row all the elements of the envelope in a vector. We only need another vector of integers to point to the start of each row. The invention of this storage scheme was motivated by the following result.

Theorem 19.1. *Let* $Fill(A) = \{(i,j) | i > j,\ a_{i,j} = 0,\ l_{i,j} \neq 0\}$ *be the index set of the fill–ins,*

$$Fill(A) \subset Env(A).$$

Proof. This is a consequence of Theorem 18.2 as it is clear from that result that there cannot be any fill–in from a node x_i to a node x_j whose number is smaller than the smallest number of the neighbors of x_i. All the paths going from x_i to x_j will have a node with a number larger than x_j. □

Giving these simple storage schemes, it was natural to try to devise ordering schemes that minimize the bandwith or the profile of the matrix. This is an NP–complete problem.

The Cuthill–McKee and reverse Cuthill–McKee orderings

The Cuthill–McKee algorithm is a local minimization algorithm to reduce the profile of A.

It is clear that if we number the nodes sequentially and, at some stage, we would like to minimize $\beta_i(A)$, then we must number immediately all the non numbered nodes in $Adj(x_i)$. Then, the algorithm is the following

Algorithm CM

1) we choose a starting node,
2) for $i = 1, \ldots, n-1$ we number all the (non numbered) neighbors of x_i in $G(A)$ in increasing order of degree,
3) we update the degrees of the remaining nodes.

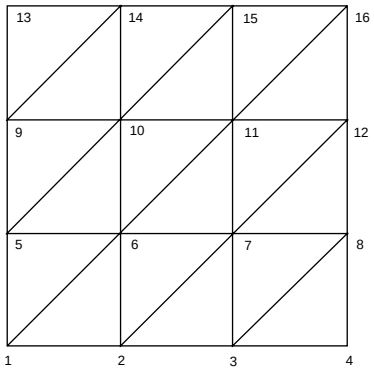

fig 19.1

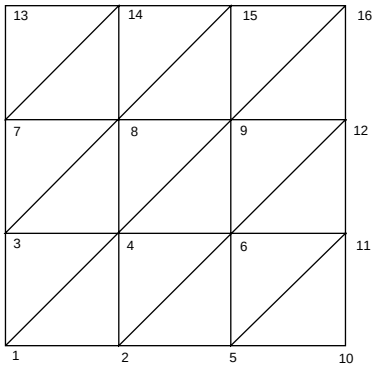

fig 19.2

The profile resulting from this numbering is quite sensitive to the choice of the starting node. Consider the initial graph shown on figure 19.1.

If we choose 1 as a starting node, we obtain figure 19.2. There are 38 fill–ins in L.

If node 4 of the initial ordering is chosen as a starting node the ordering of figure 19.3 is obtained. With this ordering, there are 14 fill–ins in L.

In the first ordering, the maximum difference of node numbers between neighbors is 7. With the second one, the maximum difference is 4.

It is interesting to look at the level structure $\mathcal{L}(1)$ for both orderings. The level structure of the first ordering is given on figure 19.4 using the numbers of the initial graph. The height is 4 and the width 7.

The level structure of the second ordering is given in figure 19.5. The height is 7 and the width is 4. Clearly, the second choice is better. The higher is the structure, the narrower it is. Starting nodes which give rise to narrow level structures are clearly better choices.

A good choice will be to choose as a starting node, a peripheral node, that is

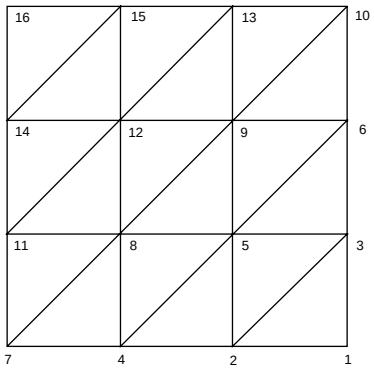

fig 19.3

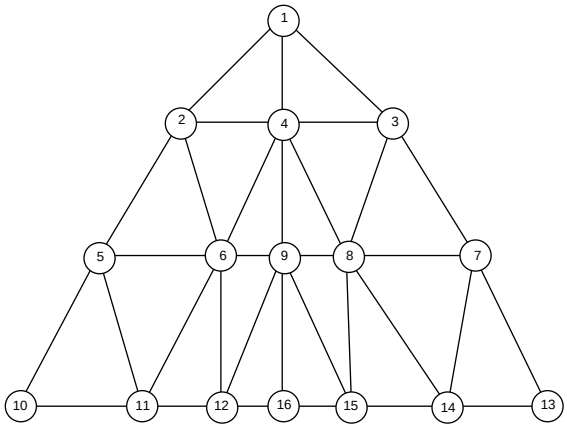

fig 19.4

one whose eccentricity equals the diameter of the graph. Peripheral nodes are not that easy to find quickly. Therefore, people have devised heuristics to find "pseudo–peripheral" nodes, i.e nodes whose eccentricities are close to the diameter of the graph. Such an algorithm was proposed by Gibbs, Poole and Stockmeyer [1976].

Algorithm GPS
1) choose a starting node r
2) build the level structure $\mathcal{L}(r)$

$$\mathcal{L}(r) = \{L_0(r), \ldots, L_{e(r)}(r)\}$$

3) sort the nodes $x \in L_{e(r)}(r)$ in increasing degree order
4) for all nodes $x \in L_{e(r)}(r)$ in increasing degree order build $\mathcal{L}(x)$. If the height of $\mathcal{L}(x)$ is greater than the height of $\mathcal{L}(r)$, choose x as a starting node ($r = x$) and go to 2)

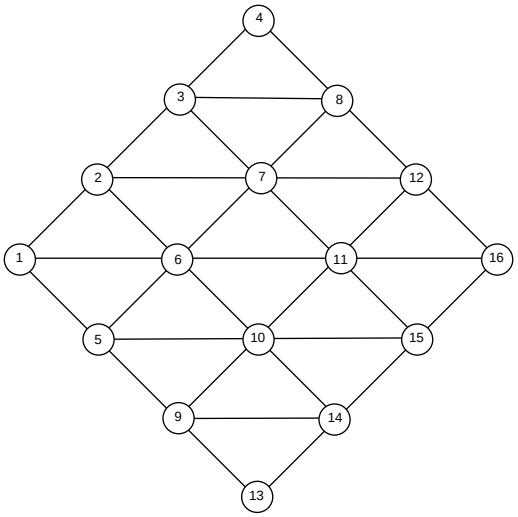

fig 19.5

We are sure that this algorithm will converge as eccentricities are bounded by the diameter of the graph. However, it can be very costly.

George and Liu [1979a] tried to shorten the computing time by eliminating structures with large width as soon as possible. Step 4) of the algorithm is modified as

4′) let $w(x)$ be the width of $\mathcal{L}(x)$. For all $x \in L_{e(r)}(r)$ in order of increasing degree, we build

$$\mathcal{L}(x) = \{L_0(x), \ldots, L_{e(r)}(x)\}.$$

At each level i if $|L_i(x)| > w(r)$, we drop the current node and we pick another one x. If $w(x) \leqslant w(r)$ and $e(x) > e(r)$, we choose x as a starting node ($r = x$) and go to 2).

George and Liu [1981] also proposed to use the following simple algorithm,

1) choose a starting node r
2) build $\mathcal{L}(r)$
3) choose a node x of minimum degree in $L_{e(r)}(r)$
4) build $\mathcal{L}(x)$. If $e(x) > e(r)$, choose x as a starting node and go to 2).

As an ordering scheme, George (George and Liu [1981]) proposed to reverse the Cuthill–McKee ordering.

Algorithm Reverse Cuthill–McKee (RCM)
1) find a pseudo–peripheral starting node
2) generate the CM ordering
3) reverse the numbering. Let $x_1, \ldots, x_n$ be the CM ordering, then the RCM ordering $\{y_i\}$ is given by $y_i = x_{n+i-1}$, $i = 1, \ldots, n$.

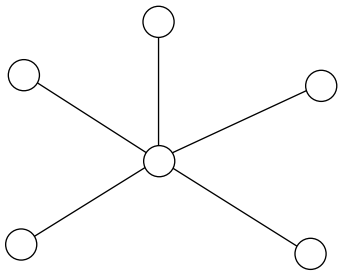

fig 19.6

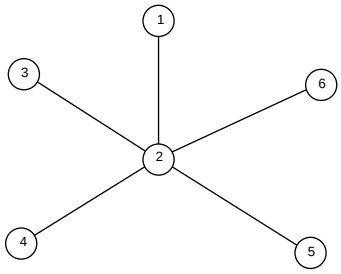

fig 19.7

Let us look at a small example. We would like to number the graph of figure 19.6. With the Cuthill–McKee algorithm, we find the graph of figure 19.7.
with a Cholesky factor

$$L = \begin{pmatrix} x & & & & & \\ x & x & & & & \\ & x & x & & & \\ & x & \bullet & x & & \\ & x & \bullet & \bullet & x & \\ & x & \bullet & \bullet & \bullet & x \end{pmatrix}.$$

We have 6 fill–ins with this ordering. If we reverse the ordering, we obtain figure 19.8.

$$L = \begin{pmatrix} x & & & & & \\ & x & & & & \\ & & x & & & \\ & & & x & & \\ x & x & x & x & x & \\ & & & & x & x \end{pmatrix}.$$

There is no fill–in at all. We are going to show that in terms of number of fill–ins, RCM is always as good as CM. So, there is no reason to use CM.

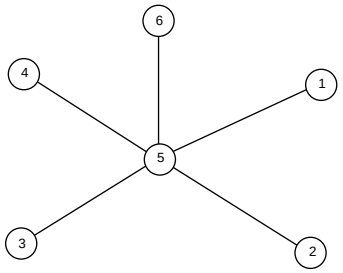

fig 19.8

Theorem 19.2. *Let A be an irreducible matrix and A_{CM} be the matrix corresponding to reordering (the graph of) A by the Cuthill–McKee scheme. Then,*

$$\forall i, j,\ i \leqslant j,\ f_i \leqslant f_j.$$

Moreover, $f_i < i$ if $i > 1$.

Proof. Suppose that the conclusion does not hold. Then, there exists a column k and rows p, l, m, $p < l < m$ such that

$$\begin{aligned} f_p &\leqslant k, \\ f_l &> k, \\ f_m &\leqslant k. \end{aligned}$$

This means that

$$\begin{aligned} a_{p,k} &\neq 0 \Longrightarrow x_p \in Adj(x_k), \\ a_{m,k} &\neq 0 \Longrightarrow x_m \in Adj(x_k), \\ a_{l,k} &= 0 \Longrightarrow x_l \notin Adj(x_k). \end{aligned}$$

But this is impossible as the Cuthill–McKee algorithm has numbered successively all nodes in $Adj(x_k)$. □

We introduce a new definition.

• $Tenv(A) = \{(i, j)|\ j \leqslant i,\ \exists k \geqslant i,\ a_{k,j} \neq 0\}$.

$Tenv(A)$ is the "transpose envelope" of A. Let us consider an example on figure 19.9. If we use the reverse Cuthill–McKee algorithm, we have to reverse the ordering. Thus, if for instance, we have the previous matrix, we obtain figure 19.10. The rows of A_{RCM} are the columns of A_{CM}.

Lemma 19.1.

$$|Env(A_{RCM})| = |Tenv(A_{CM})|.$$

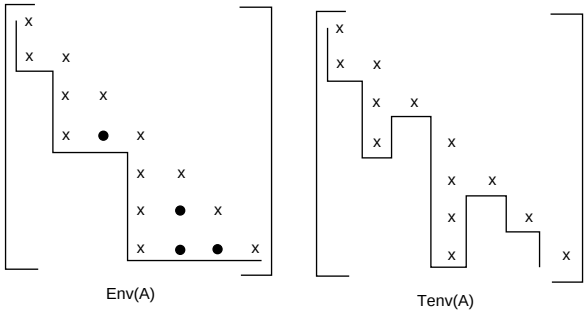

fig 19.9

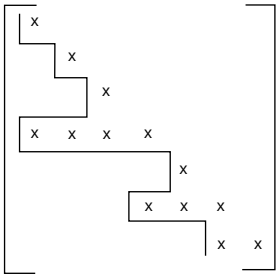

fig 19.10

Proof. Straightforward. □

Theorem 19.3.

$$Tenv(A_{CM}) \subseteq Env(A_{CM}).$$

Proof. Looking at matrix pictures, the result is obvious. However, let us formalize the proof. Suppose we have $(i,j) \in Tenv(A_{CM})$ and $(i,j) \notin Env(A_{CM})$. Then, $\exists k \geqslant i$ such that $a_{k,j} \neq 0$. Either
1) $a_{i,j} \neq 0 \Longrightarrow (i,j) \in Env(A_{CM})$
2) $a_{i,j} = 0$. If $(i,j) \notin Env(A_{CM}) \Longrightarrow \forall l \leqslant j$, $a_{i,l} = 0 \Longrightarrow f_i > j$.
On the other hand, we have $f_k \leqslant j$. This implies $f_k < f_i$ which is impossible as $k \geqslant i$ by Theorem 19.2 □

Obviously, we have

$$|Env(A_{RCM})| \leqslant |Env(A_{CM})|.$$

and George proved the following result,

Lemma 19.2. *If $\forall i > 1$, $f_i < i$, the envelope $Env(A)$ fills completely.*

This implies

$$|Fill(A_{RCM})| \leqslant |Fill(A_{CM})|.$$

There are cases for which equality holds. Notice that the previous results are true for every ordering such that $k \geqslant i \Longrightarrow f_k \geqslant f_i$. If we look back at the example given after Theorem 18.1, as a function of the initial ordering, we have if we choose 2 as the starting node (notice that 2 is a peripheral node),

	1	2	3	4	5	6	7
CM	2	1	3	4	6	7	5
RCM	6	7	5	4	2	1	3

With the initial ordering we get six fill–ins. With CM we get three fills and zero with RCM.

It has been shown that RCM can be implemented to run in $O(|E|)$ time. For a regular $N \times N$ grid and P_1 triangular finite elements, the storage for RCM varies as $O(N^3)$, ($\approx 0.7N^3$) that is $O(n^{\frac{3}{2}})$.

Several other algorithms have been proposed to reduce the profile of a symmetric matrix, for example the King algorithm, see George and Liu [1981].

20. The minimum degree ordering for symmetric matrices

This ordering scheme was introduced in Tinney and Walker [1967]. It is one of the ordering schemes that are most often used today. It is a local minimization algorithm.

The minimum degree ordering works with the elimination graphs $G^{(i)} = (X_i, E_i)$. The i–th step of the algorithm is

Algorithm MD

1) in $G^{(i)}$, find a node x_i such that

$$deg(x_i) = \min_{y \in X_i} \{deg(y)\},$$

2) form $G^{(i+1)}$ by eliminating x_i

3) if $i + 1 < n$ go to 1) with $i \leftarrow i + 1$.

So, to use this algorithm we must have a way to represent the elimination graphs and to transform them. Of course, the degree of some nodes change after the deletion of edges incident to x_i and the addition of new edges.

The biggest problem arising with the minimum degree algorithm is that quite often, there may be several nodes of minimum degree. This must be resolved using a tie breaking strategy. Unfortunately, the final number of fill–ins is quite sensitive to the tie breaking strategy, see George and Liu [1989a].

As being a local minimization algorithm, the minimum degree does not always give a minimum fill–in ordering. There are cases, like trees, for which it gives no fill–in at all, but there are examples for which it generates fill–in that is more than a constant time greater than the minimum fill–in, see (Berman and Schnitger [1990]) .

If we look again at the example after Theorem 18.1, we get the following ordering,

	1	2	3	4	5	6	7
MD	4	1	3	5	2	6	7

and there is no fill–in.

During the years many improvements have been suggested to the basic algorithm, mainly to shorten the computer time needed to run the algorithm rather than to improve the ordering. A summary of these results can be found in George and Liu [1989a]. The main points are the following,

–*Mass elimination*

When x_i is eliminated, often there are nodes in $Adj_{G^{(i)}}(x_i)$ that can be eliminated immediately. This is because, when x_i is eliminated, only the degrees of nodes in $Adj_{G^{(i)}}(x_i)$ change and some of them can be $deg(x_i) - 1$. For instance, if a node in a clique is eliminated, then the degree of all the other nodes in the clique decreases by 1. Therefore, all these nodes can be eliminated at once, before the degrees are updated. This leads to the concept of indistinguishable nodes.

• Two nodes u and v are indistinguishable in G if

$$Adj_G(u) \cup \{u\} = Adj_G(v) \cup \{v\}.$$

George and Liu [1989a] proved the following result.

Theorem 20.1. *Let $z \in Adj_{G^{(i)}}(x_i)$, then $deg_{G^{(i+1)}}(z) = deg_{G^{(i)}}(x_i) - 1$ if and only if*

$$Adj_{G^{(i)}}(z) \cup \{z\} = Adj_{G^{(i)}}(x_i) \cup \{x_i\}.$$

By merging indistinguishable nodes together, we need only to update the degrees of the representatives of these nodes.

–*Incomplete degree update*

• v is said to be outmatched by u in G if

$$Adj_G(u) \cup \{u\} \subseteq Adj_G(v) \cup \{v\}.$$

Theorem 20.2, (George and Liu [1989a] **).** *If v is outmatched by u in $G^{(i)}$, it is also outmatched by u in $G^{(i+1)}$.*

If v becomes outmatched by u, it is not necessary to update the degree of v until u is eliminated.

–*Multiple elimination*

This slight variation of the basic scheme Liu [1985] proposed is when x_i has been chosen, to select a node with the same degree than x_i in $G^{(i)}/(Adj_{G^{(i)}}(x_i) \cup \{x_i\})$. This process is repeated until there are no nodes of the same degree and then the degrees are updated.

Therefore, at each step, an independent set of minimum degree nodes is selected. Notice that the ordering that is produced is not the same as for the basic algorithm. However, it is generally as good as the "true" ordering.

–*Early stop*

In many implementations of Gaussian elimination for sparse matrices, a switch is done to dense matrices when the percentage of non zeroes in the remaining matrix is large enough. Of course, the ordering can be stopped at that stage as later stages are meaningless.

–*Tie breaking*

An issue that is really important is the choice of a tie breaking strategy. Unfortunately, not much is known about how to decide which nodes to choose at a given stage. Some experiments (George and Liu [1989a]) show that there can be large differences in the number of non zeroes and factorization times when several random tie breakers are chosen.

Most often, the initial ordering determines the way ties are broken. It has been suggested to use another ordering scheme like the Reverse Cuthill–McKee algorithm before running the minimum degree.

–*Approximate minimum degree*

In Amestoy, Davis and Duff [1994] it was proposed to use some bounds on the degree of nodes instead of the real degree. This allows a faster update of the information when nodes are eliminated. Techniques based on the quotient graph are used to obtain these bounds. The quality of the orderings that are obtained are comparable to the ones from the genuine minimum degree algorithm although the algorithm is much faster, see the performances in Amestoy, Davis and Duff [1994].

21. The nested dissection ordering for symmetric matrices

Introduction

This algorithm has been introduced by George [1973] for finite element problems and then generalized to general sparse matrices. It is very close to an old idea used in Mechanics called substructuring and also to what is now called domain decomposition.

The idea is based on Theorem 18.2 that essentially says that there won't be any fill–in between x_i and x_j if, on every path from x_i to x_j in G, there is a node with a number greater than x_i and x_j.

Consider the graph of figure 21.1 (arising, for instance, from a finite element matrix) and its partitioning given in the right hand side.

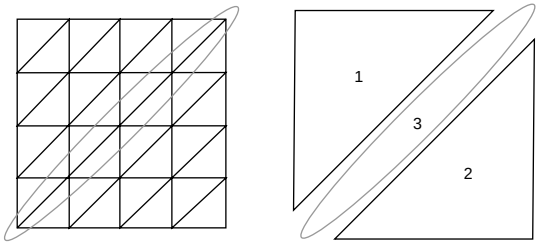

fig 21.1

The main diagonal separates the graph into three pieces. The diagonal 3 is called a separator. From Theorem 18.2, it is clear that, if we first number the nodes in part 1, then the nodes in part 2 and finally the nodes of the separator 3, there won't be any fill–in between sets of nodes 1 and 2. With this ordering and obvious notations, the matrix has the following block structure

$$A = \begin{pmatrix} A_1 & 0 & A_{3,1}^T \\ 0 & A_2 & A_{3,2}^T \\ A_{3,1} & A_{3,2} & A_3 \end{pmatrix}.$$

It is obvious that the Cholesky factor L has the following block structure

$$L = \begin{pmatrix} L_1 & & \\ 0 & L_2 & \\ L_{3,1} & L_{3,2} & L_3 \end{pmatrix},$$

this means that blocks A_1 and A_2 can be factored independently.

The basis of the nested dissection algorithm is to apply this idea recursively to sets 1 and 2. If we look at a mesh graph like the one in figure 21.2, there are two basic ways to partition it. The first one is to partition the graph into vertical (or horizontal) stripes, see figure 21.3.

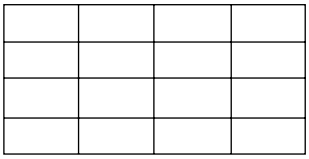

fig 21.2

This is called one–way dissection. The other way is to alternate between vertical and horizontal partitioning, see figure 21.4. This is called nested dissection. Of course, one–way dissection is a little simpler to implement.

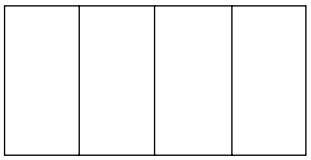

fig 21.3

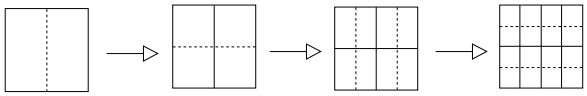

fig 21.4

One–way dissection

George [1980] considered a mesh graph consisting of an m by l grid. It is partitioned by σ vertical grid lines.

George showed that the required storage for storing the LU factors using the one–way dissection ordering is (if $m \leqslant l$)

$$S(\sigma) = \frac{ml^2}{\sigma} + \frac{3\sigma m^2}{2}.$$

This is approximately minimized (as a function of σ) by

$$\sigma = l\left(\frac{2}{3m}\right)^{\frac{1}{2}},$$

giving $S_{\text{opt}} = \sqrt{6}\, m^{\frac{3}{2}} l + O(ml)$. For comparison, numbering the graph by columns would yield a storage of $m^2 l + O(ml)$.

The operation count for the factorization is approximately

$$\theta = \frac{ml^3}{2\sigma^2} + \frac{7\sigma m^3}{6} + \frac{2m^2 l^2}{\sigma}.$$

This is approximately minimized by

$$\sigma = l\left(\frac{12}{7m}\right)^{\frac{1}{2}},$$

yielding $\theta_{\text{opt}} = \left(\frac{28}{3}\right)^{\frac{1}{2}} m^{\frac{5}{2}} l + O(m^2 l)$.

One–way dissection can be somehow generalized to general sparse matrices by using level structures of the graph, see George and Liu [1978b].

Nested dissection of a mesh graph

Consider a square mesh. A partition function Π is defined for integers i from 0 to N $(= 2^l)$ as

$$\begin{aligned}\Pi(0) &= 1\\ \Pi(N) &= 1\\ \Pi(i) &= p+1, \text{ if } i = 2^p(2q+1)\end{aligned}$$

For example, for $N = 16$, we obtain

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\Pi(i)$	1	1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	1

For $k = 1, \ldots, l$ we define sets P_k of mesh nodes (i, j) as

$$P_k = \{(i, j) | \max(\Pi(i), \Pi(j)) = k\}.$$

For a 17×17 mesh we have figure 21.5 where the numbers denote the set P_k to which the nodes belong and the lines separate these sets.

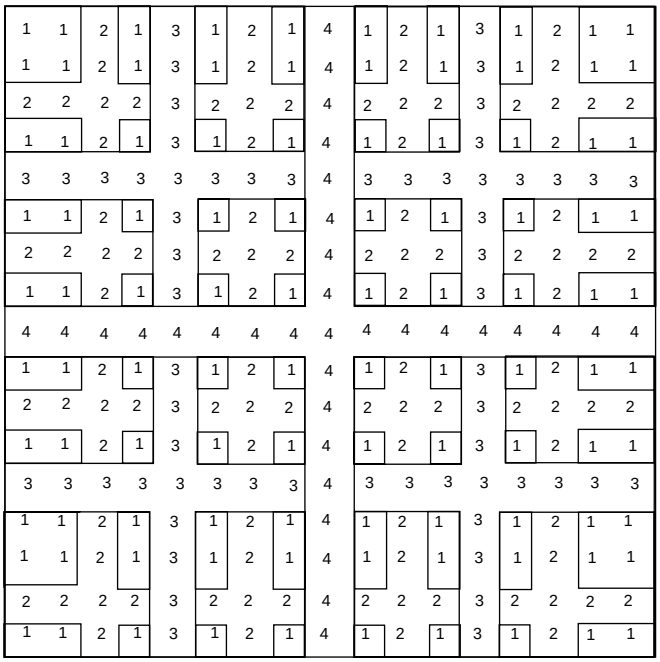

fig 21.5

Nodes in P_1 are numbered first, then nodes in P_2, etc... up to nodes in P_l. George [1973] has shown that

$$S = O(N^2 \log_2 N),$$
$$\theta = O(N^3).$$

Duff, Erisman and Reid [1976] generalized the partition function when $N \neq 2^l$ as
1) l is defined as the smallest integer such that $2^l \geqslant N$,
2) $\tilde{\Pi}(0) = \tilde{\Pi}(N) = 1$,
3) $\tilde{\Pi}(i) = 0,\ i = 1, \ldots, N-1$,
4) For $m = l, l-1, \ldots, 1$ we look for the mid point x of groups of adjacent nodes such that $\tilde{\Pi}(i) = 0$ and we set $\tilde{\Pi}(x) = m$.

Example: $N = 11 \Longrightarrow l = 4$,

Step	0	1	2	3	4	5	6	7	8	9	10	11
1	1					4						1
2	1			3		4			3			1
3	1		2	3	2	4	2		3	2		1
4	1	1	2	3	2	4	2	1	3	2	1	1

The partition function is applied to each side of the rectangular mesh.

It must be noticed that the storage schemes that we have described at the beginning of this Chapter are not well suited for nested dissection orderings. In nested dissection for mesh problems, there is a natural block structure that arises. Each block corresponds to subsets of each P_k. Diagonal blocks are stored by rows in a one dimensional array together with an integer pointer that gives the position of the diagonal element.

Non diagonal blocks are stored in a one dimensional array. It is necessary to know the beginning of each block, see figure 21.6.

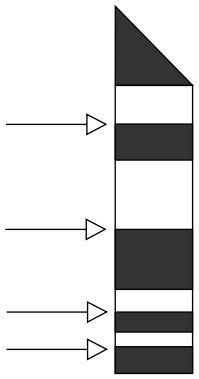

fig 21.6

For an $N \times N$ 2D grid, the storage for nested dissection is smaller than the one for RCM for $N > 37$.

Nested dissection for a general matrix

For a general sparse matrix, we have to work directly on the graph of the matrix and the problem is to find a small separator that partitions the graph in two or more components of almost an equal number of nodes. There are many ways to handle this problem.

Geometric (or greedy) algorithms can be used, see Ciarlet and Lamour [1994], Farhat [1988] or spectral bisection techniques, Simon [1991] relying on computation of the smallest non zero eigenvalue of the Laplacian matrix of the graph. The Laplacian $\mathcal{L}$ of the graph is constructed as follows: for the row i $\mathcal{L}_{i,j} = -1$ if node j is a neighbor of node i in the graph and the diagonal term is minus the sum of the other entries. From the corresponding eigenvector a partition is obtained by considering the components of the eigenvector larger or smaller than the median value.

Earlier, George and Liu [1981] proposed an algorithm based on the level structure. Consider the graph $G = (X, E)$,

1) Let $\mathcal{P} = \emptyset,\ R = X,\ N = |X|$,
2) If $R = \emptyset$ we stop. Otherwise, we choose a pseudo peripheral node y
3) We build the level structure $\mathcal{L}(y) = \{L_0, \ldots, L_l\}$ and let $j = (l+1)/2$,
4) Find a separator $S \subseteq L_j$,

$$S = \{y \in L_j | Adj(y) \cap L_{j+1} \neq \emptyset\},$$

5) Nodes in S are numbered from $N - |S| + 1$ to N using, for instance, RCM. Set $N = N - |S|$.
6) Set $R = R - S,\ \mathcal{P} = \mathcal{P} \cup \{S\}$. Go to 2)

In this algorithm, there can be some ties and the result depends on how they are broken. For instance, if we consider the graph of figure 21.7.

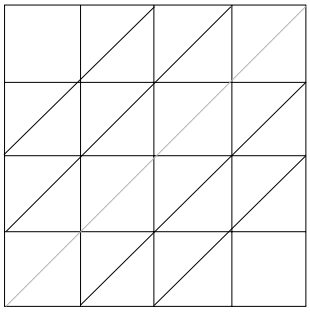

fig 21.7

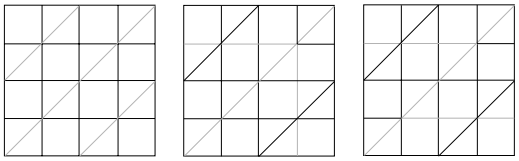

fig 21.8

The first separator is the diagonal. Then, we have several different choices given in figure 21.8, giving rise to different orderings.

Considering again the same small example, we can choose the set $\{1, 4\}$ as a separator and the following ordering,

	1	2	3	4	5	6	7
ND	6	3	2	7	1	4	5

With this ordering, there is no fill–in.

The storage scheme used for mesh problems has to be generalized to cope with the ordering scheme, see George and Liu [1981].

General theorems have been proved using graph theory about the existence of good separators, see Lipton, Rose and Tarjan [1979], Roman [1985], Charrier and Roman [1989]. Without going into too much details, let us summarize the kind of results that have been established.

• Let $\mathcal{S}$ be a family of graphs. There is an $f(n)$–separator theorem for $\mathcal{S}$ if there exists α and β positive real numbers such that $\forall\ G \in \mathcal{S}$, the nodes of G can be partitioned into three subsets A, B, C with the following properties

–there is no edges between nodes of A and nodes of B,

–$card(A) \leqslant \alpha n,\ card(B) \leqslant \alpha n$,

–$card(C) \leqslant \beta f(n)$,

where $card(A)$ is the number of elements in the subset A. C is called a separator of G.

Lipton and Tarjan [1980] proved that $f(n) = \sqrt{n}$ for planar graphs and 2D finite element graphs.

The elements of the separator C are numbered last and the same partition algorithm is applied recursively to A and B. It has been shown in Roman [1985] that

Theorem 21.1. *Let G be a graph satisfying the previous definition with $f(n) = \sqrt{n}$ with n nodes and m edges,*

–the time to construct the partitions is $O(n + m)$,

–the number of non zeroes in L is $O(n \log_2 n)$,

–the number of floating point operations for the factorization is $O(n\sqrt{n})$.

Therefore, we have the same results as for a mesh grid. These results can be slightly generalized (Roman [1985]).

22. The multifrontal method

Introduction

The multifrontal method has been introduced by Duff and Reid [1984] as a generalization of the frontal method developed by Irons [1970] for finite element problems.

The essence of the frontal method was that in finite element problems the two phases, assembly of the matrix (from integral computations) and factorization of the matrix can be mixed together. However, a variable can be eliminated only when it has been fully assembled.

The main goal of the multifrontal method is to be able to use dense matrix technology for sparse matrices. A possible drawback of the method is that technical details are quite complex and many refinements are necessary to make the method efficient. A nice exposition of the principles of the method has been given in Liu [1992]. We will follow his lines.

We consider a symmetric matrix A. The basis of the method is the block outer product Cholesky factorization,

$$A = \begin{pmatrix} D & B^{\mathrm{T}} \\ B & C \end{pmatrix} = \begin{pmatrix} L_D & 0 \\ BL_D^{-\mathrm{T}} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & C - BD^{-1}B^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} L_D^{\mathrm{T}} & L_D^{-1}B^{\mathrm{T}} \\ 0 & I \end{pmatrix},$$

$$D = L_D L_D^{\mathrm{T}}.$$

The Schur complement $C - BD^{-1}B^{\mathrm{T}}$ is the next matrix to be factored. Let D be of order $j-1$,

$$BD^{-1}B^{\mathrm{T}} = (BL_D^{-\mathrm{T}})(L_D^{-1}B^{\mathrm{T}}) = \sum_{k=1}^{j-1} \begin{pmatrix} l_{j,k} \\ \vdots \\ l_{n,k} \end{pmatrix} (\, l_{j,k} \quad \ldots \quad l_{n,k} \,),$$

$l_{i,k}$ being the elements of the Cholesky factors.

The multifrontal method uses the definition of an elimination tree. We have the following result that we state without proof,

Theorem 22.1. *If node k is a descendant of node j in $T(A)$, then the nonzero structure of $(l_{j,k}, \ldots, l_{n,k})^{\mathrm{T}}$ is contained in the structure of $(l_{j,j}, \ldots, l_{n,j})^{\mathrm{T}}$. If $l_{j,k} \neq 0$, $k < j$, node k is a descendant of node j in $T(A)$.*

As in Liu [1992], let us consider a small example,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & & & x & x & x \\ & x & & & x & \\ & & x & & x & x \\ x & & & x & & x \\ x & x & x & & x & x \\ x & & x & x & x & x \end{pmatrix} \end{array}.$$

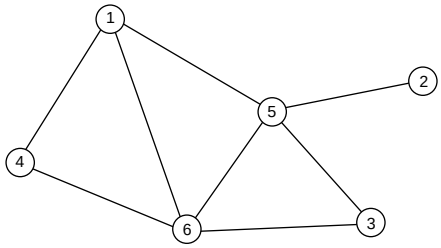

fig 22.1

The graph of the matrix A is given on figure 22.1.

Doing Gaussian elimination, we get

$$L = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & & & & & \\ & x & & & & \\ & & x & & & \\ x & & & x & & \\ x & x & x & \bullet & x & x \\ x & & x & x & x & x \end{pmatrix} \end{array}.$$

The elimination tree $T(A)$ is shown on figure 22.2.

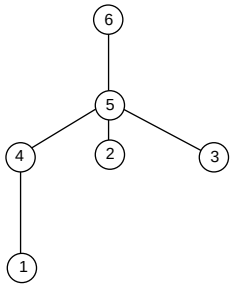

fig 22.2

It is clear that we can eliminate $1, 2$ and 3 independently. If we consider 1, we can restrict ourselves to the following matrix (rows and columns where there are non zeroes in the first row and first column,

$$F_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{4,1} & & & \\ a_{5,1} & & & \\ a_{6,1} & & & \end{pmatrix} \end{array}.$$

Eliminating 1 will create contributions in a reduced matrix $\bar{U}_4$,

$$\bar{U}_4 = \begin{array}{c} \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & \bullet & x \\ \bullet & x & x \\ x & x & x \end{pmatrix} \end{array},$$

where, as before, the $\bullet$ represents a fill–in. In parallel, we can eliminate 2, defining

$$F_2 = \begin{array}{c} \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 5 \end{array} \\ \begin{pmatrix} a_{2,2} & a_{2,5} \\ a_{5,2} & \end{pmatrix} \end{array}.$$

Elimination of 2 will create a contribution to the $(5,5)$ term,

$$\bar{U}_5^2 = \begin{array}{c} \\ 5 \end{array} \begin{array}{c} \begin{array}{c} 5 \end{array} \\ \begin{pmatrix} x \end{pmatrix} \end{array}.$$

We can also eliminate 3,

$$F_3 = \begin{array}{c} \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{3,3} & a_{3,5} & a_{3,6} \\ a_{5,3} & & \\ a_{6,3} & & \end{pmatrix} \end{array}.$$

Elimination of 3 will create contributions,

$$\bar{U}_5^3 = \begin{array}{c} \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cc} 5 & 6 \end{array} \\ \begin{pmatrix} x & x \\ x & x \end{pmatrix} \end{array}.$$

Then, we eliminate 4. For this, we have to consider the matrix resulting from the elimination of 1, that is

$$F_4 = \begin{array}{c} \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{4,4} & 0 & a_{4,6} \\ 0 & & \\ a_{6,4} & & \end{pmatrix} \end{array} + \bar{U}_4.$$

Elimination of 4 creates contributions,

$$\bar{U}_5^4 = \begin{array}{c} \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cc} 5 & 6 \end{array} \\ \begin{pmatrix} x & x \\ x & x \end{pmatrix} \end{array}.$$

Now, before eliminating node 5, we must sum up the contributions from the original matrix and what we get from the eliminations of nodes $2, 3$ and 4. To do this, we must extend $\bar{U}_5^2$ to the proper set of indices, i.e. $5, 6$. We do this as in Liu [1992] by considering an operator that we denote by $\circ$.

For two matrices A and B, $A \circ B$ takes as the set of indices of the result, the union of the sets of indices of A and B and whenever they coincide, the result is the sum of the entries. Let

$$\bar{U}_5 = \bar{U}_5^2 \circ \bar{U}_5^3 \circ \bar{U}_5^4,$$

$$F_5 = \begin{pmatrix} a_{5,5} & a_{5,6} \\ a_{6,5} & 0 \end{pmatrix} + \bar{U}_5.$$

Elimination of 5 gives a matrix of order 1 that is added to $a_{6,6}$ to give the last term of the factorization.

On this example, we have seen that all the elimination steps can be carried out by working on small dense matrices of different orders, extending and summing up these matrices by looking at the elimination tree.

Description of the method

Following Liu [1992], let us formalize the process of the multifrontal method. Giving the elimination tree $T(A)$, we define the subtree update matrix for column j as

$$\bar{U}_j = - \sum_{k \in T[j] - \{j\}} \begin{pmatrix} l_{j,k} \\ l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{pmatrix} \begin{pmatrix} l_{j,k} & l_{i_1,k} & \dots & l_{i_r,k} \end{pmatrix},$$

where, $i_0 = j, i_1, \dots, i_r$ are the row indices of the non zeroes in column j of L, $(L_{*,j})$.

$\bar{U}_j$ contains the contributions for the columns preceding j which are proper descendants of j in the tree. The frontal matrix F_j is defined as

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \dots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & & \\ a_{i_r,j} & & & \end{pmatrix} + \bar{U}_j.$$

The first column of F_j contains all the non zero updates entries to column j. Then, we perform one step of elimination on F_j,

$$F_j = \begin{pmatrix} l_{j,j} & 0 \\ l_{i_1,j} & \\ \vdots & I \\ l_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & U_j \end{pmatrix} \begin{pmatrix} l_{j,j} & l_{i_1,j} & \dots & l_{i_r,j} \\ 0 & & I & \end{pmatrix}.$$

U_j is called the update matrix. It is a dense matrix. It is proved in Liu [1992] that

$$U_j = - \sum_{k \in T[j]} \begin{pmatrix} l_{i_1,k} \\ \vdots \\ l_{i_r,k} \end{pmatrix} (\, l_{i_1,k} \quad \ldots \quad l_{i_r,k} \,) \, .$$

If $c_1, \ldots, c_s$ are the children of j in $T(A)$, then

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \ldots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & & \\ a_{i_r,j} & & & \end{pmatrix} \circ U_{c_1} \circ \ldots \circ U_{c_s} .$$

Then, the multifrontal method is defined as

`for j=1:n`
1) Form the update matrix $U_{c_1} \circ \ldots \circ U_{c_s}$
2) Form the frontal matrix F_j
3) Factorize F_j
`end`

A lot of issues have to be considered to derive an efficient multifrontal matrix code. Let us consider what the main problems are.
1) Storage of the frontal and update matrices

Update matrices must be stored and easily retrieved when they are needed in the algorithm to contribute to a frontal matrix. A nice way to do this is to use a topological ordering of $T(A)$ and to number the nodes in every subtree consecutively (this is called a postordering). Then, the update matrices can be stored in a stack using a last–in first–out algorithm. Using this scheme, the update matrices appear at the top of the stack in the order they are needed.

To manage the storage working space, Duff proposed to use a buddy system. In a buddy system, each block of storage has a buddy with which it can be combined to form a larger block. In a binary buddy system, the sizes of the block are $c2^i$. Each block keeps some associated information: a flag to indicate if the block is free or not and the logarithm of the size of the block. Free blocks are linked through a doubly linked list. There is also a free list for each block size.

When a working area of memory of size m is needed, the system allocates a block of list i, where $c2^i \geqslant m$. If there is no block on the i–th free list, a level $i+1$ block must be split in two. A part is used to serve the request, the other part is put on the i–th free list. The algorithm is applied recursively.

When a block is deallocated, the system checks if the block's buddy is free and of the correct size. If the answer is positive, the two blocks are combined and put on the $i+1$–rst free list.

Liu proposed a postordering of $T(A)$ that minimizes the working storage. Generally, reduction in the working storage can be obtained by tree restructuring. In particular, tree rotations can be used, see Liu [1988].

2) Supernode methods

Some nodes in $T(A)$ can be grouped together and treated as a single computational unit. Formally (Liu [1992]), a supernode is a set of contiguous nodes

$$\{j, j+1, \ldots, j+s\},$$

such that

$$Adj_G(T[j]) = \{j+1, \ldots, j+s\} \cup Adj_G(T[j+s]).$$

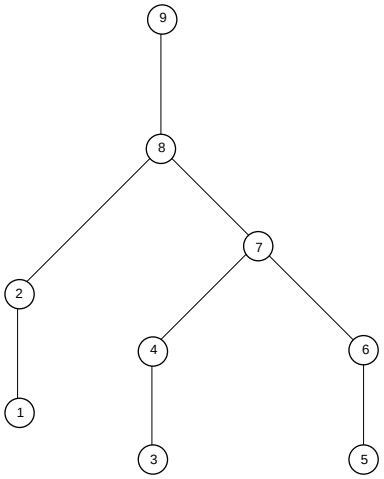

fig 22.3

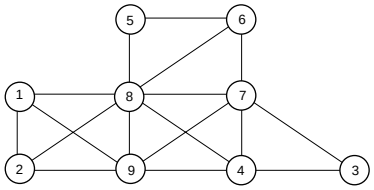

fig 22.4

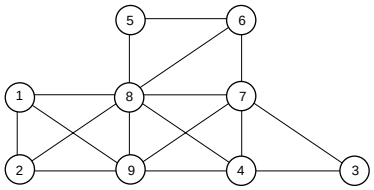

fig 22.5

Consider the tree of figure 22.3, corresponding to the graph of figure 22.4. The supernodal elimination tree is given in figure 22.5. As an example, consider

$$Adj_G(T[7]) = \{8,9\} = \{8,9\} \cup Adj_G(T[9]) = \{8,9\} \cup \emptyset.$$

Therefore, $\{7,8,9\}$ is a supernode.

Then, all the nodes in a supernode are eliminated in the same step of the algorithm. The advantages of using supernodes are that the supernodal tree is smaller and the frontal matrices have a larger order giving better performances on modern computers.

3) Dense techniques

An advantage of the multifrontal method is to allow to use dense matrix techniques in the sparse case. Particularly, dense algorithms based on the use of Level 3 BLAS can be used when factoring the frontal matrices. Furthermore, there is much less indirect addressing.

4) Node amalgamation

There are some performance advantages to have large supernodes. Therefore, it has been suggested (Duff and Reid [1984]) to amalgamate some nodes, treating some zeroes as non zeroes to obtain larger supernodes. For a detailed study of this point, see Amestoy [1990].

Software packages are available that implement the multifrontal method. See various programs in the Harwell Scientific Library, or the MUPS code developed by Amestoy and Duff.

23. Non symmetric sparse matrices

Introduction

There is an additional difficulty in Gaussian elimination for non symmetric sparse problems, namely the need of pivoting to improve numerical stability. When dealing with sparse symmetric positive definite systems, the ordering of the unknowns can be chosen only for the purpose of maintaining sparsity as much as possible during the elimination. This is not true anymore for non symmetric problems.

There are still a lot of active research going on to try to make the algorithms for non symmetric systems as efficient and robust as those we have seen in the previous sections. We are going to briefly review these efforts.

The Markowitz criterion

If we choose the pivots as it is done for dense systems (for instance, partial pivoting), there is no room for preserving sparsity. Therefore, for sparse matrices, we have to relax the constraints for choosing the pivot. The usual strategy is to consider candidate pivots satisfying the inequality,

$$|a_{i,j}^{(k)}| \geqslant u \max_l |a_{l,j}^{(k)}|,$$

where u is a user defined parameter such that $0 < u \leqslant 1$. This will limit the overall growth as

$$\max_i |a_{i,j}^{(k)}| \leqslant \left(1 + \frac{1}{u}\right)^{p_j} \max_i |a_{i,j}|,$$

where p_j is the number of off diagonal entries in column j of U, see Duff, Erisman and Reid [1986].

From these candidates, one is selected that minimizes

$$(r_i^{(k)} - 1)(c_j^{(k)} - 1),$$

where $r_i^{(k)}$ is the number of non zero entries in row i of the remaining $(n-k)\times(n-k)$ matrix in A_k. Similarly, $c_j^{(k)}$ is the number of non zeroes in column j. This criterion is due to Markowitz [1957].

This method modifies the least entries in the remaining submatrix. Notice that if A is symmetric, this is exactly the minimum degree algorithm that was introduced historically after the Markowitz criterion. Many variations of the Markowitz criterion have been studied. For a summary, see Duff, Erisman and Reid [1986]. However, most of these other methods are generally not as efficient as the Markowitz criterion.

One possibility is to choose the entry (which is not too small) that introduces the least amount of fill–in at step k. Unfortunately, this is much more expensive than the Markowitz criterion. Moreover, having a local minimum fill–in does not always gives a globally optimal fill–in count. There are even some examples where the Markowitz criterion does a better job at globally reducing the fill–in.

As for the minimum degree algorithm, the tie breaking strategy is quite important for the result of the Markowitz algorithm. Details of the implementation of the Markowitz algorithm are discussed in Duff, Erisman and Reid [1986]. A switch to dense matrix techniques is done when the non zero density is large enough.

Elimination structures for non symmetric matrices

Gilbert and Liu [1993] have introduced some definitions to characterize the structures of the triangular factors of a non symmetric matrix (without pivoting).

This starts by considering a triangular matrix L and its directed graph $G(L)$ which is acyclic (there is no directed cycles). An acyclic directed graph is called a dag. Let $w = (w_1, \ldots, w_n)^{\mathrm{T}}$, then we define

$$Struct(w) = \{i \in \{1, \ldots, n\} | w_i \neq 0\}.$$

Theorem 23.1. *If $Lx = b$ then $Struct(x)$ is given by the set of vertices reachable from vertices of $Struct(b)$ in the dag $G(L^{\mathrm{T}})$.*

An economical way to represent the information contained in a dag G is to consider its transitive reduction G^0. Then, in Theorem 23.1, we can replace $G(L^{\mathrm{T}})$ by $G^0(L^{\mathrm{T}})$. The transitive closure G^* of a directed graph G is a graph that has an edge (u, v) whenever G has a directed path from u to v.

Let A be factored as $A = LU$ without pivoting. $G^0(L)$ and $G^0(U)$ are called the lower and upper elimination dags (edags) of A. For a symmetric matrix, $G^0(L)$ and $G^0(U)$ are both equal to the elimination tree.

If B and C are two matrices with non zero diagonal elements then $G(B) + G(C)$ is the union of the graphs of B and C i.e. the graph whose edge set is the union of

those of $G(B)$ and $G(C)$. $G(B) \cdot G(C)$ is the graph with an edge (i,j) if (i,j) is an edge of $G(B)$ or (i,j) is an edge of $G(C)$ or there is a k such that (i,k) is an edge of $G(B)$ and (k,j) is an edge of $G(C)$.

Gilbert and Liu [1993] proved the following result.

Theorem 23.2. *If $A = LU$ and there is a path in $G(A)$ from i to j, then there exists a $k, 1 \leqslant k \leqslant n$ such that $G^0(U)$ has a path from i to k and $G^0(L)$ has a path from k to j. That is*

$$G^*(A) \subseteq G^{0^*}(U) \cdot G^{0^*}(L).$$

If there is no cancellation in the factorization $A = LU$, then

$$G(L) \cdot G(U) = G(L) + G(U).$$

From these results, the row and column structures of L and U can be derived.

Theorem 23.3. *If $l_{i,j} \neq 0$, then there exists a path from i to j in $G^0(L)$.*

Let $i > j$, $l_{i,j} \neq 0$ if and only if there exists $k \leqslant j$ such that $a_{i,k} \neq 0$ and there is a directed path in $G^0(U)$ from k to j.

$$Struct(L_{*,j}) = Struct(A_{*,j}) \cup \bigcup \{Struct(L_{*,k}) | k < j, u_{k,j} \neq 0\} - \{1, \ldots, j-1\}.$$

In the last statement, $k < j, u_{k,j} \neq 0$ can be replaced by (k,j) is an edge of $G^0(U)$. Similarly the structure of U can be characterized.

From these results, an algorithm can be derived for the symbolic fill computation when there is no pivoting, see Gilbert and Liu [1993]. When pivoting is required for stability, edags can also be useful, this time to symbolically compute the fill at each stage of Gaussian elimination.

The multifrontal method

The multifrontal algorithm we have studied for symmetric matrices previously has been applied to non symmetric matrices by Duff and Reid [1984]. The idea was to consider the sparsity pattern of $A + A^{\mathrm{T}}$ to construct the elimination tree. Numerical pivoting takes place within the frontal matrices. This works well if the pattern of A is nearly symmetric. However, the results are poor if the pattern of A is far from being symmetric.

Recently Davis and Duff have introduced an extension of the multifrontal algorithm to non symmetric matrices in a series of papers: Davis and Duff [1993], Davis [1992], Davis [1993], Hadfield and Davis [1992], [1994a], [1994b]). This results in a package called UMFPACK, see Davis [1993].

In the non symmetric case, the frontal matrices are rectangular. The elimination tree is replaced by a directed acyclic graph (DAG) and update matrices are no longer assembled by a single parent.

We briefly summarize the work of Davis and Duff [1993]. A frontal matrix F_k is a dense $r_i^{(k)} \times c_j^{(k)}$ submatrix that corresponds to the choice of $a_{i,j}^{(k)}$ as a pivot. The columns in F_k are defined by the set of indices $\mathcal{U}_k$ and the rows by the set $\mathcal{L}_k$. As in the symmetric case, amalgamation can take place if some other pivots have the same row and column patterns. Let g_k be the number of such pivots. Then

$$\mathcal{L}_k = \mathcal{L}'_k \cup \mathcal{L}''_k, \quad \mathcal{U}_k = \mathcal{U}'_k \cup \mathcal{U}''_k,$$

where $\mathcal{L}'_k$ and $\mathcal{U}'_k$ correspond to the g_k pivots. We write

$$F_k = \begin{pmatrix} E_k & B_k \\ C_k & D_k \end{pmatrix},$$

where E_k is of order g_k. A numerical factorization $E_k = L'_k U'_k$ is computed as well as the block column L''_k of L and the block row U''_k of U and the Schur complement

$$D'_k = D_k - L''_k U''_k.$$

In the factorization any entry of E_k can be selected as a pivot. A bipartite graph $\mathcal{A}^k = (\mathcal{A}^k_\nu, \mathcal{A}^k_\epsilon)$ represents the active submatrix,

$$\mathcal{A}^k_\nu = \mathcal{A}^k_{\mathrm{R}} \cup \mathcal{A}^k_{\mathrm{C}},$$

where $\mathcal{A}^k_{\mathrm{R}}$ (resp. $\mathcal{A}^k_{\mathrm{C}}$) is the index set of the rows (resp. columns) and

$$\mathcal{A}^k_\epsilon = \{(i,j) | i \in \mathcal{A}^k_{\mathrm{R}},\, j \in \mathcal{A}^k_{\mathrm{C}},\, a^{(k)}_{i,j} \neq 0\}.$$

The factorized frontal matrices are described by a directed acyclic graph

$$G^k = (\nu^k, \epsilon^k),$$

$$\nu^k = \{t | F_t \text{ is a frontal matrix created before step } k\},$$

$$\nu^k \subseteq \{1, \ldots, k-1\},$$

$$\epsilon^k = \epsilon^k_L \cup \epsilon^k_U,$$

$$\epsilon^k_L = \{(i,j) | i < j < k,\, i \in \nu^k,\, j \in \nu^k, \text{one or more entire rows of } F_i^j \text{ are assembled in } F_j\},$$

$$\epsilon^k_U = \{(i,j) | i < j < k,\, i \in \nu^k,\, j \in \nu^k, \text{one or more entire columns of } F_i^j \text{ are assembled in } F_j\},$$

F_i^j is a submatrix of F_i formed by rows and columns that are non pivotal and not yet assembled into subsequent frontal matrices before step j.

Edges in ϵ^k are called inactive edges. Node i is an LU child of node j if F_i^j is assembled into F_i, i.e. $(i,j) \in \epsilon^k_L \cap \epsilon^k_U$. If $(i,j) \in \epsilon^k_L$, i is an L–child of j. If $(i,j) \in \epsilon^k_U$, i is an U–child of j.

An active edge connects a frontal matrix in G with a row or column in $\mathcal{A}$ for which it has an unassembled contribution. They can be partitioned into active L–edges and active U-edges.

The actual algorithm is quite complex. Therefore, we refer the reader to Davis and Duff [1993] and we consider just the beginning of a small example.

Before the start, the DAG G^1 is empty and $\mathcal{A}^1$ is the bipartite graph of the original matrix. A pivot is chosen (based on the Markowitz strategy) and permuted to the first row and column. Consecutive pivots with the same row and column patterns are also included in the first frontal matrix F_1 which is then factorized.

The first node in G refers to F_1. $\mathcal{L}_1$ (resp. $\mathcal{U}_1$) is given by the set of edges in $\mathcal{A}^1$ incident to columns (resp. rows) nodes 1 through g_1. Pivot row and column nodes 1 through g_1 and edges in $\mathcal{A}^1$ incident to these nodes are removed from $\mathcal{A}^1$. $\mathcal{L}_1''$ (resp. $\mathcal{U}_1''$) defines new active L (resp. U) edges.

The example taken from Davis and Duff [1993] is the following,

$$A = \begin{pmatrix} a_1 & & & x & x & & \\ x & a_2 & x & & x & & x \\ x & x & a_3 & & & & x \\ x & & & a_4 & x & & \\ & x & x & & x & x & \\ & & & & & x & \\ x & x & & & x & & x \end{pmatrix}.$$

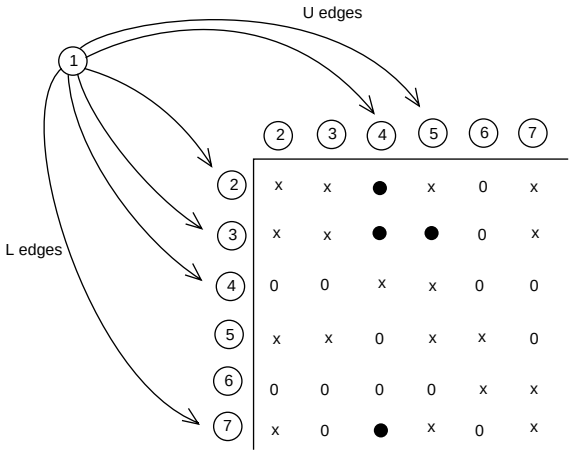

fig 23.1

Node 1 is an L–child of node 2. Two pivots are considered in the second step as rows and columns 2 and 3 have the same pattern in L and U. The graphs are given in figure 23.2.

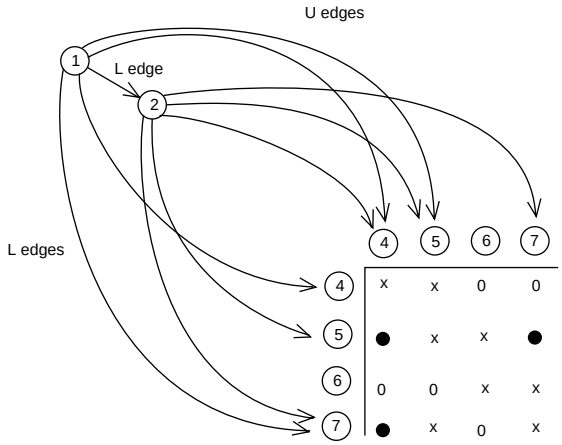

fig 23.2

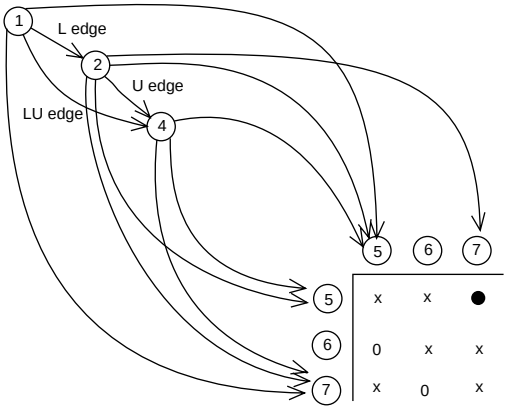

fig 23.3

Then, node 4 is eliminated, $u_{2,4}$ and $u_{3,4}$ are non zero as well as $l_{4,1}$ and $l_{1,4}$. This gives figure 23.3. and the algorithm goes on...

Some edge reduction operations are applied during the factorization, see Davis and Duff [1993]. With no additional edge reductions, a frontal matrix persists until the latest possible step of the elimination when it is completely assembled into other frontal matrices. The edge reduction discussed in Davis and Duff tries to assemble these contributions as soon as possible.

Some node amalgamation can also be introduced to give larger frontal matrices at the price of having a little more fill–in. Additionally, to speed up the pivot search,

only upper and lower bounds of the degree of each row and column are computed. Moreover, only the first few columns with minimum upper bounds are scanned.

The DAG (the edge set) is constructed for being reused if some matrix with the same pattern is to be factored. Numerical experiments and comparisons with other methods are given in Davis and Duff [1993].

SuperLU

In 1995, J. Demmel, S. Eisenstat, J. Gilbert, X. Li and J.W. Liu described and made available a code named SuperLU that combines several interesting techniques.

1) The first thing is to extend supernodes to non symmetric matrices. Several possibilities were studied but the one retained in SuperLU is the following. A supernode is a range of columns in L with a full triangular diagonal block and the same structure below the diagonal block. This will allow a supernode to column update using BLAS2 routines. All the updates to a column from a supernode are done together.

In SuperLU (written in C), the matrix is stored by columns as well as L and U but the diagonal blocks in the supernodes are stored in L (as if they were full). This allows to treat supernodes as two dimensional arrays.

As many supernodes are not large enough, some columns are merged into artificial supernodes even if they don't have the same structure. Also several consecutive columns can be factored at the same time (this set is called a panel).

2) The column elimination tree is defined as the elimination tree of $A^{\mathrm{T}}A$. The column elimination tree T gives the dependencies among columns in the LU factorization. In particular, if $l_{i,j} \neq 0$ then, i is an ancestor of j in T, if $u_{i,j} \neq 0$ then, j is an ancestor of i in T.

Before factoring the matrix, the columns are permuted according to a postordering of the column elimination tree.

3) Before each supernode–panel update, a symbolic factorization is done using a depth first traversal of the graph (this is in fact done on a reduced graph).

Careful analysis and experiments were done by Demmel and al. showing that this code is one of the most efficient one for non symmetric matrices.

24. Numerical stability for sparse matrices

The study of componentwise error analysis for sparse systems has been considered in Arioli, Demmel and Duff [1989]. In this paper, they show how to compute estimates of the backward error. The perturbation f of the right hand side is chosen in an a posteriori way and is not equal to $|b|$ to keep the perturbation on A sparse and the iterative refinement algorithm convergent (see Arioli, Demmel and Duff [1989]).

Let $w = |A|\ |y| + |b|$, y being the computed solution. A threshold τ_i is chosen for each w_i such that if $w_i > \tau_i$, then $f_i = |b_i|$. Otherwise if $w_i \leqslant \tau_i$, f_i is chosen larger. The value of τ_i suggested in Arioli, Demmel and Duff [1989] is $\tau_i = 1000\ n\ u(\|A_{i,*}\|_\infty \|y\|_\infty + |b_i|)$ where $A_{i,*}$ is the ith row of A.

Let $f^{(2)}$ be the components of f for which $w_i \leqslant \tau_i$, then $f^{(2)}$ is defined as $f^{(2)} = \|b\|_\infty e$ where e is the column vector of all ones. With this choice, we can

compute an estimate of the backward error

$$\frac{|b - Ay|_i}{(|A||y| + f)_i}.$$

Remember that the condition number is

$$K_{\mathrm{BS}}(A, b) = \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty}.$$

But we may just want to estimate $\| \, |A^{-1}| \, |A| \, \|_\infty = \| \, |A^{-1}| \, |A|e\|_\infty$. This can be estimated by an algorithm due to Hager [1984] that uses multiplications by the matrix and its transpose. This is obtained by forward and backward solves using the LU factorization of A.

Numerical experiments in Arioli, Demmel and Duff [1989] using iterative refinement show that it is possible to guarantee solutions of sparse linear systems that are exact solutions of a nearby system with a matrix of the same structure. Estimates of the condition number and of the backward error can be obtained easily using the previous strategies giving estimates of the error.

5. Parallel algorithms for sparse matrices

25. Introduction

As for dense matrices, it is important to be able to solve efficiently sparse linear systems on parallel architectures. From one side, it is easier to consider sparse matrices rather than dense ones as, in the sparse case, there is more natural parallelism. Data dependencies are weaker in the sparse case as, in the factorization process, some columns are independent of each other. On the other side, it is more difficult to obtain significant performances as the granularity of independent tasks is often quite small and indirect addressing could lead to a lack of data locality.

Nevertheless, a lot of research is going on to obtain efficient algorithms. As for dense matrices, we will only consider algorithms for distributed memory architectures. A good review has been done in Heath, Ng and Peyton [1991].

26. Symmetric positive definite systems

As we have seen, for symmetric positive definite matrices, Gaussian elimination proceeds in three phases: ordering, symbolic factorization and numerical factorization. The problem we are faced with is to have parallel implementations of these three phases. Therefore, things are complicated as for instance for the first phase, not only we have to find an ordering that reduces the fill–in and gives a good degree of parallelism during the solution phase, but also ideally, we need to be able to compute this ordering in parallel.

Orderings

As we have seen, the most commonly used heuristic algorithm to reduce fill–in is the minimum degree or one of its variants. Although there is not much theory about it (except negative results), it works quite well in practice.

Unfortunately, the basic minimum degree algorithm is quite sequential by nature. Modifications have to be made to generate parallelism and to be able to run the algorithm on parallel computers. Several attempts have been made or are under consideration by now.

One idea is to look for multiple elimination of independent nodes of minimum degree, see Liu [1985]. Another idea is to relax a bit the constraint of finding a node of minimum degree and just to look at nodes whose degree are within a constant factor of the minimum degree.

However, the implementation of the algorithm on sequential computers is now really efficient but quite complex and it is not that easy to port it to a parallel computer.

An ordering that is much more promising for parallelism is the nested dissection algorithm. As it is a divide and conquer algorithm, it is well suited to parallel computers. However, as we have seen, the algorithm is not easy to implement for general sparse matrices. For this, we need an efficient algorithm for partitioning a graph satisfying certain constraints on the separators. Research is underway to

reach these goals, Simon [1991], Ciarlet and Lamour [1994], Farhat [1988], even though these algorithms are not parallel themselves. Some research is under way in this field. Combinations of the minimum degree and nested dissection have been proposed by Liu [1989b]. Therefore, the problem of finding in parallel a good ordering for parallel computation cannot be considered to be completely solved by 1996.

The current approach of the ordering problem is to separate the two (conflicting) goals that we have. First of all, an ordering for (approximately) minimizing the fill–in is chosen. Then, the ordering is modified by restructuring the elimination tree in order to introduce parallelism.

This approach has been described by Jess and Kees [1982]. The method starts by looking at PAP^{T}, where the permutation P was chosen to preserve sparsity. Then, the natural ordering is a perfect elimination one for $F = L + L^{\mathrm{T}}$. Now, the goal is to find a permutation matrix Q that gives also a perfect elimination but with more parallelism.

A node in G_F whose adjacency set is a clique is called simplicial. Such a node can be eliminated without causing any fill–in. Two nodes are independent if they are not adjacent in G_F. The Jess and Kees algorithm is the following,

• Until all nodes are eliminated, choose a maximum set of independent simplicial nodes, number them consecutively and eliminate these nodes.

It has been shown, Liu [1988], that the Jess and Kees method gives an ordering that has the shortest elimination tree over all orderings that do a perfect elimination of F.

The problem now is to implement this algorithm. This question was not really addressed by Jess and Kees. A proposal using clique trees was described in Lewis, Peyton and Pothen [1989]. We will return to this method later. Another implementation was proposed in Liu and Mirzaian [1989].

Heuristically, it can be seen that larger elimination trees (having more leaf nodes) introduce more parallelism. The number of nodes being fixed, larger trees mean shorter trees. Therefore, it seems that finding an ordering that gives a shorter tree would increase the level of parallelism.

Liu [1988] has proposed to use tree rotations to reach this goal. The purpose of this algorithm is to find a reordering by working on the structure of PAP^{T}, namely the elimination tree.

A node x in a tree $T(B)$ is eligible for rotation if

$$Adj_{G(B)}(T[x]) \neq Anc(x),$$

where $Anc(x)$ is the set of ancestors of x in T and

$$Adj_{G(B)}(T[v]) = Anc(v), \ \forall v \text{ ancestor of } x.$$

A tree rotation at x is a reordering of $G(B)$ such that the nodes in $Adj_{G(B)}(T[x])$ are labeled last while keeping the relative order of the nodes.

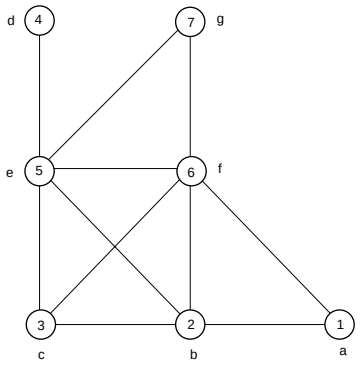

fig 26.1

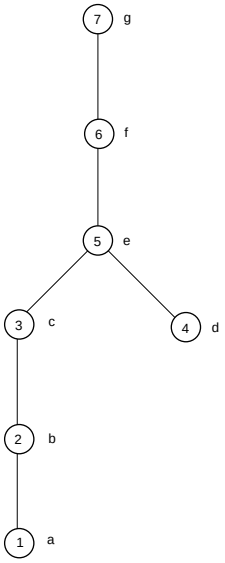

fig 26.2

Consider a small example similar to the one in Liu [1988], see figures 26.1 and 26.2.

$$Adj(T[c]) = \{e, f\} \neq \{e, f, g\} = Anc(c),$$
$$Adj(T[d]) = \{e\} \neq \{e, f, g\} = Anc(d).$$

Thus, c and d are eligible for tree rotations. A rotation at c gives what is seen in figures 26.3 and 26.4. A rotation at d gives figures 26.5 and 26.6.

Let $h_{\mathrm{T}}(v)$ be the height of $T[v]$ and $\bar{h}_{\mathrm{T}}$ be defined as -1 if every subtree of T

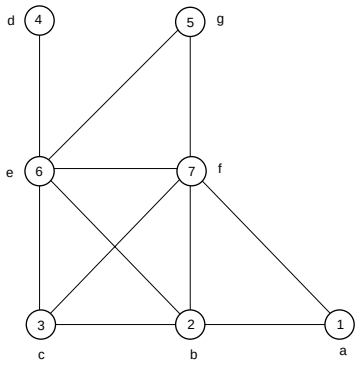

fig 26.3

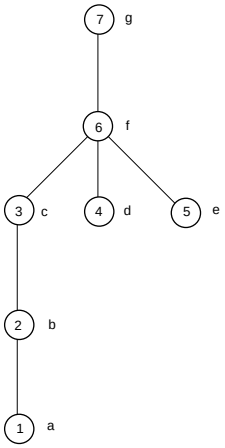

fig 26.4

intersects $T[v]$ or otherwise,

$$\bar{h}_{\mathrm{T}} = \max\{h_{\mathrm{T}}(w) | T[w] \cap T[v] = \emptyset\}.$$

In the original tree,

$$h_{\mathrm{T}}(a) = 0, h_{\mathrm{T}}(b) = 1, h_{\mathrm{T}}(c) = 2, h_{\mathrm{T}}(d) = 0, h_{\mathrm{T}}(e) = 3, \mathrm{etc}\ldots$$

$$\bar{h}_{\mathrm{T}}(a) = 0, \bar{h}_{\mathrm{T}}(b) = 0, \bar{h}_{\mathrm{T}}(c) = 0, \bar{h}_{\mathrm{T}}(d) = 2, \bar{h}_{\mathrm{T}}(e) = -1, \mathrm{etc}\ldots.$$

The algorithm proposed by Liu [1988] is the following,

• If x is an eligible node with $\bar{h}_{\mathrm{T}}(x) < h_T(x)$, then apply a tree rotation at x relabeling the nodes.

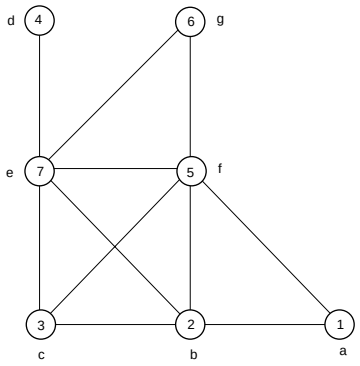

fig 26.5

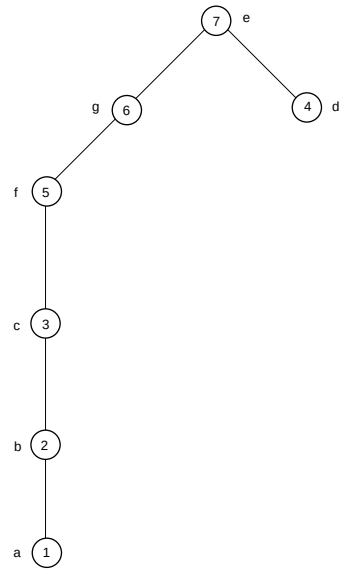

fig 26.6

In the above example, only c can be chosen by the algorithm. Results are proven in Liu [1988] that supports this choice. The implementation details and experimental results are given in Liu [1988]. However, tree rotations do not always give a tree of minimum height.

More than the height of the elimination tree, it will be better to minimize the parallel completion time. This was defined by Liu as,

• Let $time[v]$ be the execution time for the node v in T (for instance, a constant times the number of operations).

Then,

$$level[v] = \begin{cases} time[v] & \text{if } v \text{ is the root} \\ time[v] + level[parent\ of\ v] & \text{otherwise.} \end{cases}$$

The parallel completion time is $\max_{v \in T} level[v]$.

Liu and Mirzaian [1989] proposed an implementation of the Jees and Kees algorithm. In their method, the cost of detecting simplicial nodes is $O(n\nu(F))$ where $\nu(F)$ is the number of off diagonal elements in $L + L^{\mathrm{T}}$. The tree rotations heuristic is faster than that.

In Lewis, Peyton and Pothen [1989], an implementation of the Jees and Kees algorithm was proposed using clique trees. They gave a characterization of simplicial nodes,

Theorem 26.1. *A node is simplicial if and only if it is contained in only one maximal clique.*

Proof. See Lewis, Peyton and Pothen [1989]. □

Let $Madj(v) = \{u | u \in Adj(v),\ u > v\}$, then

Theorem 26.2. *If K is a maximal clique in G_F and v is the lowest numbered node in K, then*

$$K = \{v\} \cup Madj(v).$$

Proof. See Lewis, Peyton and Pothen [1989]. □

The clique $K(v)$ is represented by such a v which is called a representative node. Lewis, Peyton and Pothen proposed a way to find the representative nodes.

Theorem 26.3. *A node v is not representative for any maximal clique if and only if there exists a representative node z, $z < v$, such that*

$$\{v\} \cup Madj(v) \subset \{z\} \cup Madj(z).$$

Proof. See Lewis, Peyton and Pothen [1989]. □

An algorithm based on these results can be constructed to find the representative nodes, hence the maximal cliques. Considering the details of the algorithm will take us too far as it is rather technical. However, during the marking procedure nodes are partitioned into sets $new(K(v)), anc(K(v))$.

$New(K(v))$ consists of v together with the nodes marked as non representative while considering $K(v)$. $Anc(K(v))$ is the ancestor set of $K(v)$. Rather than looking

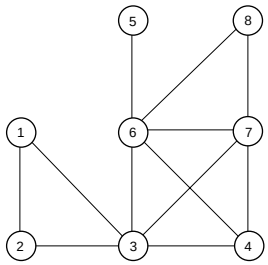

fig 26.7

at the details, let us consider the example given in Lewis, Peyton and Pothen [1989] which comes from Liu, see figure 26.7.

$$K(1) = \{1,2,3\},\ K(3) = \{3,4,6,7\},\ K(5) = \{5,6\},\ K(6) = \{6,7,8\},$$

$$newK(1) = \{1,2\},\ newK(3) = \{3,4\},\ newK(5) = \{5\},\ newK(6) = \{6,7,8\},$$

$$ancK(1) = \{3\},\ ancK(3) = \{6,7\},\ ancK(5) = \{6\},\ newK(6) = \emptyset.$$

The number of nodes of the clique tree is the number of maximal cliques and

$$parent(K) = \{K' | first\ anc(K) \in new(K')\},$$

where $first\ anc(K)$ is the smallest element of $anc(K)$,

$$parent[newK(1)] = newK(3),$$
$$parent[newK(3)] = newK(6),$$
$$parent[newK(5)] = newK(6),$$

and the clique tree is the one in figure 26.8.

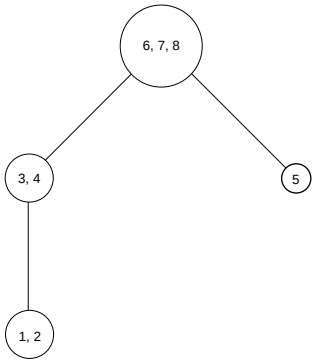

fig 26.8

Lewis, Peyton and Pothen [1989] described how to update the clique tree during elimination using the Jees and Kees algorithm.

Let $R(K)$ be defined as

$$R(K) = \begin{cases} K & \text{if } K \text{ contains no simplicial nodes,} \\ K - \{u\} & \text{where } u \text{ is the eliminated simplicial node of } K. \end{cases}$$

Given the clique tree, when a simplicial node u contained in K is eliminated, the new clique tree is constructed in the following way,

– if $R(K)$ is maximal, make no changes
– otherwise,
  – if K is a leaf, delete K,
  – if K is an ancestor, let C be a child with the largest ancestor set
    – assign all nodes in $new(K)$ to $new(C)$, removing them from $anc(C)$,
    – attach C as a child of $parent(K)$ with $first\ anc(C) = first\ anc(K)$,
    – attach children of K other than C as children of C,
    – delete K.

Proofs are given in Lewis, Peyton and Pothen [1989]. Experimental results in Lewis, Peyton and Pothen show that this implementation is about as fast as the tree rotation heuristic and faster than the Liu and Mirzaian [1989] Jees and Kees implementation.

Symbolic factorization

There is not much parallelism to be found in the symbolic factorization phase. The natural structure for increasing the level of parallelism is the elimination tree. But, then we are faced with a bootstrapping problem as we have also to compute the elimination tree in parallel.

George, Heath, Liu and Ng [1988] have proposed a column oriented parallel symbolic factorization algorithm. Experimental results showed only modest speed ups. Further research is needed in this area.

Numerical factorization

The first algorithms that were studied were column oriented. Traditionally, the two main operations of Gaussian elimination are denoted:

- $cmod(j, k)$: modification of column j by column k, $k < j$
- $cdiv(j)$: division of column j by a scalar (the pivot).

In the fan–out and fan–in algorithms, data distribution is such that columns are assigned to processors. As in the dense case, column k is stored on processor $p = map(k)$. Leaf nodes of the elimination tree are independent of each other and can be processed first. Let $mycols(p)$ be the set of columns owned by processor p and

$$procs(L_{*,k}) = \{map(j) | j \in Struct(L_{*,k})\}.$$

The fan–out algorithm is the following,

```
For j ∈ mycols(p)
  if j is a leaf node
    cdiv(j)
    send L_{*,j} to p' ∈ procs(L_{*,j})
    mycols(p)=mycols(p)-{ j}
  end
  while mycols(p) ≠ ∅
    receive column L_{*,k}
    for j ∈ Struct(L_{*,k})∩ mycols(p)
      cmod(j,k)
      if all cmods are done for column j
        cdiv(j)
        send L_{*,j} to p' ∈ procs(L_{*,j})
        mycols(p)=mycols(p)-{ j}
      end
    end
  end
end
```

When columns are sent to other processors, it is also necessary to send their structures to be able to complete the cmods operations. There are too many communications in this algorithm. This has been improved in the fan–in algorithm Ashcraft, Eisenstat and Liu [1990],

• Processing column j, processor p computes the modification $u(j,k)$ for $k \in mycols(p) \cap Struct(L_{j,*})$. If p does not own column j, it sends $u(j,k)$ to processor $map(j)$. If p owns column j, it receives and processes the aggregated modifications and then completes the $cdiv(j)$ operation.

When processing column j, if there are no modifications arrived yet for this column, processor p can proceed to compute some columns $i > j$ by aggregating modifications already received or receiving updates. In practice, this look ahead is limited to column i in the same supernode as column j.

An important issue in these column oriented algorithms is the mapping of columns to the processors. Currently, most implementations use a static mapping of computational tasks to processors. This can lead to load balancing problems. In the fan–out or fan–in algorithms, the assignment of columns to processors is guided by the elimination tree. The goals are good load balancing and few processor communications.

The first implementations were based on wrap mapping of the levels of the elimination tree starting from the bottom up. This gives good load balancing properties but too many communications.

Another technique that has been much used is the subtree to subcube mapping. This was specifically designed for hypercube architectures but can be easily generalized to other distributed memory architectures.

This is illustrated on the following example, see figure 26.9, distributing the tree on 4 processors.

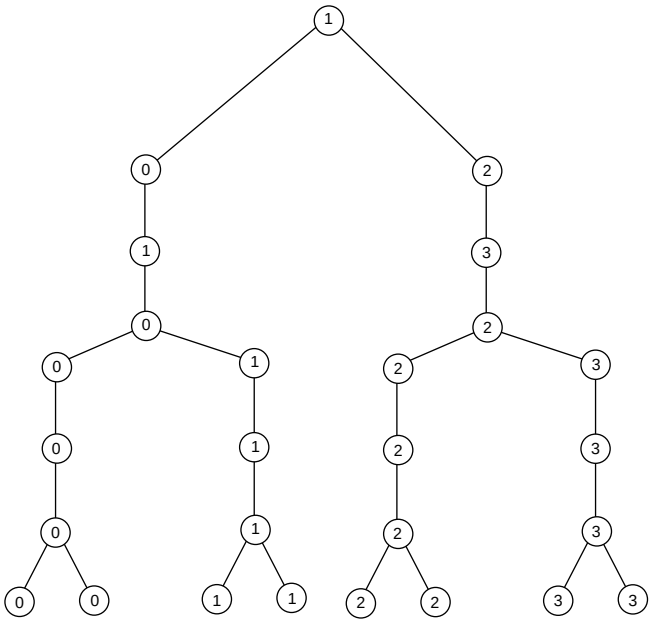

fig 26.9

In Karypis and Kumar [1994], a subforest to subcube mapping is introduced which seems to improve on the subtree to subcube mapping.

Another algorithm that is in favor is the multifrontal algorithm. We have already seen that there is a natural parallelism in the early (bottom) stages of the multifrontal method. Dense frontal matrices are assigned to one processor. The problem is that when moving to the root of the tree, there is less and less parallelism of this kind. However, frontal matrices are getting larger and larger and dense techniques used to handle these matrices can be distributed on several processors (using level 3 BLAS primitives).

In Kapyris and Kumar [1994], the multifrontal method is implemented using the subforest to subcube mapping and very good efficiencies are obtained. In this scheme, many subtrees of the elimination tree are assigned to each subcube. They are chosen in order to balance the work. Algorithms are given in Kapyris and Kumar [1994] to obtain this partitioning.

All these mappings are based on column distribution of the matrix to the processors. Rothberg and Gupta [1994] proposed to use a block oriented approach of sparse Gaussian elimination. These partitionings have good communication performances.

27. Non symmetric systems

As for dense systems, the problem here is complicated by the issue of pivoting.

There are not many efficient implementations of Gaussian elimination on parallel architectures for general sparse matrices.

The best algorithms are based on the multifrontal method, either the methods based on the structure of $A + A^{\mathrm{T}}$ or the truly non symmetric ones.

The non symmetric version of the multifrontal method has been studied by Hadfield and Davis [1994b] for parallel computing. As for the symmetric case, different levels of parallelism must be used. Experimental results are given in Hadfield and Davis [1994b].

Davis and Yew [1990] have proposed another nondeterministic algorithm called D2. This algorithm is based on selecting an independent set of m pivots whose updates can be computed in parallel

$$S = \{a_{i,j}^{(k-1)} | a_{i,j}^{(k-1)} = a_{j,i}^{(k-1)} = 0, \text{ for } k \leqslant j \leqslant m+k-1, k \leqslant i \leqslant m+k-1, \text{ and } i \neq j\}.$$

Then, a sparse rank–m update is done. This algorithm gives good results on matrices that have a very non symmetric pattern.

References

AASEN,J.O., (1971), On the reduction of a symmetric matrix to tridiagonal form, *BIT 11*, 233–242

AHAC,A.A., BUONI,J.J. and OLESKY,D.D., (1988), Stable LU factorization of H–matrices, *Linear Algebra and its Appl. 99*, 97–110

AHAC,A.A. and OLESKY,D.D., (1986), A stable method for the LU factorization of M–matrices, *SIAM J. Alg. Disc. Meth. 7 no 3*, 368–378

ALAGHBAND,G., (1989), Parallel pivoting combined with parallel reduction and fill–in control, *Parallel Comp. 11*, 201–221

ALVARADO,F.L. and SCHREIBER,R., (1993), Optimal parallel solution of sparse triangular systems, *SIAM J. Sci. Stat. Comput. 14 no 2*, 446–460

AMESTOY,P.R., (1990), Factorisation de grandes matrices creuses non symétriques basée sur une méthode multifrontale dans un environnement multiprocesseur, *Ph.D. Thesis, CERFACS report TH/PA/91/2*

AMESTOY,P., DAVIS,D.A. and DUFF,I.S., (1994), An approximate minimum degree ordering algorithm, *Report TR–94–039, University of Florida*

AMESTOY,P.R. and DUFF,I.S., (1989), Vectorization of a multiprocessor multifrontal code, *Int. J. Supercomputer Appl. 3 no 3*, 41–59

AMODO,P., BRUGNANO,L. and POLITI,T., (1993), Parallel factorizations for tridiagonal matrices, *SIAM J. Numer. Anal. 30 no 3*, 813–823

ANAND,I.M., (1980), Numerical stability of nested dissection orderings, *Math. Comp. 35 no 152*, 1235–1249

ANDERSON,E., BAI,Z., BISCHOF,C., DEMMEL,J., DONGARRA,J., DU CROZ,J., GREENBAUM,A., HAMMARLING,S., McKENNEY,A., OSTROUCHOV,S. and SORENSEN,D., (1992), *LAPACK User's guide*, (SIAM)

ANDERSON,E. and SAAD,Y., (1989), Solving sparse triangular linear systems on parallel computers, *Int. J. High Speed Comp. 1 no 1*, 73–95

ANDERSON,D.V., FRY,A.R., GRUBER,R. and ROY,A., (1987), Gigaflop speed algorithm for the direct solution of large block tridiagonal systems in 3D physics applications, *Report UCRL–96034, Lawrence Livermore Nat. Lab.*

ANGELACCIO,M. and CLAJANNI,M., (1994), The row/column pivoting strategy on multicomputers, *Parallel Comp. 20 no 2*, 197–214

ANGELACCIO,M. and CLAJANNI,M., (1994), Subcube matrix decomposition, a unifying view for LU factorization on multicomputers, *Parallel Comp. 20 no 2*, 257–270

ARIOLI,M., DEMMEL,J.W. and DUFF,I.S., (1989), Solving sparse linear systems with backward error, *SIAM J. Matrix Anal. Appl. 10*, 165–190

ASHCRAFT,C., (1991), A taxonomy of distributed dense LU factorization methods, *Report ECA–TR–161, Boeing Computer Services*

ASHCRAFT,C., EISENSTAT,S.C. and LIU,J.W–H., (1990), A fan–in algorithm for distributed sparse numerical factorization, *SIAM J. Sci. Stat. Comput. 11 n,o 3*, 593–599

ASHCRAFT,C. and GRIMES,R.G., (1989), The influence of relaxed supernode partitions on the multifrontal method, *ACM Trans. Math. Soft. 15*, 291–309

ASHCRAFT,C., GRIMES,R.G., LEWIS,J.G., PEYTON,B.W. and SIMON,H.D., (1987), Progress in sparse matrix methods for large linear systems on vector supercomputers, *Int. J. Supercomputer Appl. 1 no 4*, 10–30

AXELSSON,O. and BARKER,V.A., (1984) *Finite element solution of boundary value problems*, (Academic Press)

BABUŠKA,I. and ELMAN,H.C., (1989), Some aspects of parallel implementation of the finite element method on message passing architectures, *J. Comp. Appl. Math. 27*, 157–187

BANK,R.E. and ROSE,D.J., (1990), On the complexity of sparse Gaussian elimination via bordering, *SIAM J. Sci. Stat. Comput. 11 no 1*, 145–160

BANK,R.E. and SMITH,R.K., (1987), General sparse elimination requires no permanent integer storage, *SIAM J. Sci. Stat. Comput. 8 no 4* 574–584

BARLOW,J.L., (1986), A note on monitoring the stability of triangular decomposition of sparse matrices, *SIAM J. Sci. Stat. Comput. 7 no 1*, 166–168

BARNARD,S.T., POTHEN,A. and SIMON,H.D., (1993), A spectral algorithm for envelope reduction of sparse matrices, *Tech Rep, Univ. of Waterloo, CS–93–49*

BARWELL,V. and GEORGE,A., (1976), A comparison of algorithms for solving symmetric indefinite systems of linear equation, *ACM Trans. Math. Soft. 2 no 3*, 242–251

BAUER,F.L., (1974), Computational graphs and rounding error, *SIAM J. Numer. Anal. 11*, 87–96

BENNETT,J.M., (1965), Triangular factors of modified matrices, *Numer. Math. 7*, 217–221

BERMAN,A. and PLEMMONS,R.J., (1979) *Nonnegative matrices in the mathematical sciences*, (Academic Press)

BERMAN,P. and SCHNITGER,G., (1990), On the performance of the minimum degree ordering for Gaussian elimination, *SIAM J. Matrix Anal. Appl. 11 no 1*, 83–89

BISCHOF,C.H., (1990), Incremental condition estimation, *SIAM J. Matrix Anal. Appl. 11*, 312–322

BISCHOF,C.H., LEWIS,J.G. and PIERCE,D.J., (1990), Incremental condition estimation for sparse matrices, *SIAM J. Matrix Anal. Appl. 11 no 4*, 644–662

BLAIR,J.R. and PEYTON,B., 1993, An introduction to chordal graphs and clique trees, *in [167]*, 1–29

BODEWIG,E., (1959) *Matrix calculus*, second edition, (North–Holland)

BOISVERT,R.F., (1991), Algorithms for special tridiagonal systems, *SIAM J. Sci. Stat. Comput. 12 no 2*, 423–442

BOTHE,Z., (1975), Bounds for rounding errors in the Gaussian elimination for band systems, *J. Inst. Maths. Applics. 16*, 133–142

BRAYTON,R.K., GUSTAVSON,F.G. and WILLOUGHBY,R.A., (1970), Some results on sparse matrices, *Math. Comp. 24 no 112* 937–954

BROYDEN,C.G., (1973), Some condition number bounds for the Gaussian elimination process, *J. Inst. Maths. Applics. 12*, 273–286

BUNCH,J.R., (1971), Analysis of the diagonal pivoting method, *SIAM J. Numer. Anal. 8 no 4*, 656–680

BUNCH,J.R., (1973), Complexity of sparse elimination, *in [348]*, 197–220

BUNCH,J.R., (1974), Partial pivoting strategies for symmetric matrices, *SIAM J. Numer. Anal. 11 no 3*, 521–528

BUNCH,J.R., (1974), Analysis of sparse elimination, *SIAM J. Numer. Anal. 11 no 5*, 847–873

BUNCH,J.R., (1987), The weak and strong stability of algorithms in numerical linear algebra, *Linear Algebra and its Appl. 88*, 49–66

BUNCH,J.R., DEMMEL,J.W. and VAN LOAN,C., (1989), The strong stability of algorithms for solving symmetric linear systems, *SIAM J. Matrix Anal. Appl. 10 no 4*, 494–499

BUNCH,J.R., DONGARRA,J.J., MOLER,C. and STEWART,G.W., (1979), *Linpack User's guide*, (SIAM)

BUNCH,J.R. and HOPCROFT,J.,E., (1974), Triangular factorization and inversion by fast matrix multiplication, *Math. Comp. 28 no 125*, 231–236

BUNCH,J.R. and KAUFMAN,L., (1977), Some stable methods for calculating inertia and solving symmetric linear systems, *MC 31*, 162–179

BUNCH,J.R. and PARLETT,B.N., (1971), Direct methods for solving symmetric indefinite systems of linear equations, *SIAM J. Numer. Anal. 8 no 4*, 639–655

BUNCH,J.R. and ROSE,D.J. Eds, (1976) *Sparse matrices computations*, (Academic Press)

BURGESS,I.W. and LAI,P.K.F., (1986), A new node renumbering algorithm for bandwith reduction, *Int. J. Num. Meth. Eng. 23*, 1693–1704

CARNEVALI,P., RADICATI,G., ROBERT,Y. and SGUAZZERO,P., (1987), Efficient Fortran implementation of the Gaussian elimination and Householder reduction algorithms on the

IBM 3090 vector multiprocessor, *Report ICE–0012 IBM European Center for Scientific and Engineering Computing*

CARTER,R., (1991), Y–MP floating point and Cholesky factorization, *Int. J. High Speed Comp. 3 no 3*, 215–222

CHAN,T., (1984), On the existence and computation of LU factorizations with small pivots, *Math. Comp. 42 no 166*, 535–547

CHAN,T.F. and FOULSER,D.E., (1988), Effectively well conditioned linear systems, *SIAM J. Sci. Stat. Comput. 9*, 963–969

CHAN.,T. and RESASCO,D.C., (1986), Generalized deflated block elimination, *SIAM J. Numer. Anal. 23 no 5*, 913–924

CHAN,W.M. and GEORGE,A., (1980) A linear time implementation of the reverse Cuthill–McKee algorithm, *BIT 20*, 8–14

CHANDRASEKARAN,S. and IPSEN,I., (1995), On the sensitivity of solution components in linear systems of equations, *SIAM J. Matrix Anal. Appl. 16 no 1* 93–112

CHARRIER,P. and ROMAN,J., (1989), Algorithmique et calculs de complexité pour un solveur de type dissection emboîtée, *Numer. Math. 55*, 463–476

CHARRIER,P. and ROMAN,J., (1991), Analysis of refined partitions for a parallel implementation of nested dissection, *Report LABRI 91–38, Université de Bordeaux I*

CHATELIN,F. and FRAYSSE,V., (1993), Qualitative computing, elements of a theory of finite precision computation, *CERFACS Report*

CHEN,S.C., KUCK,D.J. and SAMEH,A.H., (1978), Practical parallel band triangular system solvers, *ACM Trans. Math. Soft. 1 no 3*, 270–277

CHOI,J., DONGARRA,J., POZO,R. and WALKER,D.W., (1994), ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers, *in Proceedings of Fourth Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia)* (IEEE Computer Society Press)

CHU,E. and GEORGE,A., (1987), Gaussian elimination with partial pivoting and load balancing on a multiprocessor, *Parallel Comp. 5*, 65–74

CIARLET,P. Jr. and LAMOUR,F., (1994), On the validity of a front oriented approach to partitioning large sparse graphs with a connectivity constraint, *Report CAM 94–37, UCLA*, submitted to Numerical Algorithms

CLEARY,A.J., (1990), Parallelism and fill–in in the Cholesky factorization of reordered banded matrices, *Report SAND90–2757, Sandia Nat. Lab., Albuquerque*

CLINE,A.K., CONN,A.R. and VAN LOAN,C.F., (1982), Generalizing the LINPACK condition estimator, *Lecture Notes in Mathematics 909*, (Springer), 73–83

CLINE,A.K., MOLER,C.B., STEWART,G.W. and WILKINSON,J.H., (1979), An estimate of the condition number of a matrix, *SIAM J. Numer. Anal. 16*, 368–375

CONCUS,P., GOLUB,G.H. and MEURANT,G. (1985), Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Stat. Comput. 6 no 1*, 220–252

COSNARD,M., ROBERT,Y., QUINTON,P. and TCHUENTE,M. Eds, (1986) *Parallel algorithms and architectures*, (North–Holland)

COSNARD,M., ROBERT,Y. and TRYSTRAM,D., (1986), Résolution parallèle de systèmes linéaires denses par diagonalisation, *EDF, Bulletin DER, série C no 2*, 67–88

CRYER,C.W., (1968), Pivot growth in Gaussian elimination, *Numer. Math. 12*, 335-345

CSANKY,L., (1976), Fast parallel matrix inversion algorithms, *SIAM J. Comp. 5 no 4*, 618–623

CURTIS,A.R. and REID,J.K., (1971), The solution of large sparse unsymmetric systems of linear equations, *J. Inst. Maths. Applics. 8*, 344–353

CURTIS,A.R. and REID,J.K., (1972), On the automatic scaling of matrices for Gaussian elimination, *J. Inst. Maths. Applics. 10*, 118–124

CUTHILL,E., (1972), Several strategies for reducing the bandwith of matrices, *in [317]*

DAHLQUIST,G. and BJÖRCK,Å, (1974) *Numerical methods*, (Prentice–Hall)

DAHLQUIST,G., EISENSTAT,S.C. and GOLUB,G.H., (1972), Bounds for the error of linear systems of equations using the theory of moments, *J. of Math. Anal. and Appl. 37 no 1*, 151–166

DAVE,A.K. and DUFF,I.S., Sparse matrix calculations on the Cray 2, *Parallel Comp. 5*, 55–64

DAVIS,T.A., (1992), Performance of an unsymmetric pattern multifrontal method for sparse LU factorization, *Report TR–92–014, University of Florida*

DAVIS,T.A., (1993), Users' guide for the unsymmetric pattern multifrontal package, *Report TR–93–020, University of Florida*

DAVIS,T.A., (1994), A combined unifrontal/multifrontal method for unsymmetric sparse matrices, *Report TR–94–005, University of Florida*

DAVIS,T.A. and DUFF,I.S., (1993), An unsymmetric pattern multifrontal method for sparse LU factorization, *Report TR–93–018, University of Florida*

DAVIS,T.A. and YEW,P.C., (1990), A nondeterministic parallel algorithm for unsymmetric sparse LU factorization, *SIAM J. Matrix Anal. Appl. 11*, 383–402

DAYDE,M.J. and DUFF,I.S., (1989), Use of Level 3 BLAS in LU factorization on the Cray 2, the ETA–10P and the IBM 3090/VF, *Int. J. Supercomputer Appl. 3 no 2*, 40–70

DAYDE,M.J., DUFF,I.S. and PETITET,A., (1993), A parallel block implementation of level 3 BLAS for MIMD vector processors, *Report RAL–93–037, Rutherford Appleton Lab.*

DEMMEL,J.W., (1984), Underflow and the reliability of numerical software, *SIAM J. Sci. Stat. Comput. 5*, 887–919

DEMMEL,J.W., (1987), On condition numbers and the distance to the nearest ill–posed problem, *Numer. Math. 51 no 3*, 251–289

DEMMEL,J.W., (1989), On floating point errors in Cholesky, *Lapack Working note 14, University of Tennessee*

DEMMEL,J.W., (1992), Open problems in numerical linear algebra, *Report CS–92–164 Univ. of Tennessee*

DEMMEL,J.W., (1992), Trading off parallelism and numerical stability, *Report CS–92–179 Univ. of Tennessee*

DEMMEL,J.W., (1992), The componentwise distance to the nearest singular matrix, *SIAM J. Matrix Anal. Appl. 13 no 1*, 10–19

DEMMEL,J.W., HEATH,M.T. and VAN DER VORST,H.A. (1993), Parallel Numerical linear algebra, *Acta Numerica*, 111–197

DEMMEL,J.W., HIGHAM,N.J. and SCHREIBER,R.S., (1995), Stability of block LU factorization, *Numer. Lin. Alg. with Appl. 2 no 2*

DESPREZ,F., TOURANCHEAU,B. and DONGARRA,J.J., (1994), Performance complexity of LU factorization with efficient pipelining and overlap on a multiprocessor; *Report Univ of Tennessee 1994*

DONGARRA,J. and DEMMEL,J.W., (1991), LAPACK; a portable high–performance numerical library for linear algebra, *Supercomputer 46*, 33–38

DONGARRA,J., DU CROZ,J., HAMMARLING,S. and DUFF,I.S., (1990), A set of level 3 basic linear algebra subprograms, *ACM Trans. Math. Soft. 16 no 1*, 1–17

DONGARRA,J., DU CROZ,J., HAMMARLING,S. and HANSON,R., (1988), An extended set of Fortran basic linear algebra subprograms, *ACM Trans. Math. Soft. 14 no 1*, 1–17

DONGARRA,J., DUFF,I.S., GAFFNEY,P. and McKEE,S. Eds, (1989) *Vector and parallel computing, issues in applied research and development*, (Ellis Horwood)

DONGARRA,J., DUFF,I.S., SORENSEN,D.C. and VAN DER VORST,H.A., (1991) *Solving linear systems on vector and shared memory computers*, (SIAM)

DONGARRA,J., GUSTAVSON,F.G. and KARP,A., (1984), Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, *SIAM Rev. 26*, 91–112

DONGARRA,J. and HEWITT,T., (1986), Implementing dense linear algebra algorithms using multitasking on the Cray X–MP–4 (or approaching the Gigaflop), *SIAM J. Sci. Stat. Comput. 7 no 1*, 347–305

DONGARRA,J. and SAMEH,A.H., (1984), On some parallel banded system solvers, *Parallel Comp. 1*, 223–235

DONGARRA,J. and WALKER,D., (1993), The design of linear algebra libraries for high performance computers, *Report University of Tennessee*

DRMAC,Z., OMLADIC,M. and VESELIC,K., (1994), On the perturbation of the Cholesky factorization, *SIAM J. Matrix Anal. Appl. 15 no 4*, 1319–1332

DU CROZ,J.J. and HIGHAM,N.J., (1992), Stability of methods for matrix inversion, *J. Inst. Maths. Applics. 12*, 1–19

DUFF,I.S., (1974), On the number of nonzeros added when Gaussian elimination is performed on sparse random matrices, *Math. Comp. 28 no 125*, 219–230

DUFF,I.S., (1977), A survey of sparse matrix research, *Proc. of the IEEE 65 no 4*, 500–535

DUFF,I.S. Ed, (1981) *Sparse matrices and their use*, (Academic Press)

DUFF,I.S., (1981) MA32, a package for solving sparse unsymmetric systems using the frontal method, *Report AERE R10079, Harwell Lab.*

DUFF,I.S., (1983), The solution of sparse linear equations on the Cray 1, *Report CSS125, Harwell Lab.*

DUFF,I.S., (1984), Design features of a frontal code for solving sparse unsymmetric linear systems out–of–core, *SIAM J. Sci. Stat. Comput. 5*, 270–280

DUFF,I.S., (1984), Direct methods for solving sparse systems of linear equations, *SIAM J. Sci. Stat. Comput. 5 no 3*, 605–619

DUFF,I.S., (1984) Data structures, algorithms and software for sparse matrices, *Report CSS 158, Harwell Lab.*

DUFF,I.S., (1986), Parallel implementation of multifrontal schemes, *Parallel Comp. 3*, 193–204

DUFF,I.S., (1986), The use of vector and parallel computers in the solution of large sparse linear equations, *Report AERE R12393, Harwell Lab.*

DUFF,I.S., (1986), The influence of vector and parallel processors on numerical analysis, *Report AERE R12329, Harwell Lab.*

DUFF,I.S., (1988) Multiprocessing a sparse matrix code on the Alliant FX/8, *Report CSS 210, Harwell Lab.*

DUFF,I.S., (1993), The solution of augmented systems, *Report RAL–93–084 Rutherford Appleton Lab.*

DUFF,I.S., ERISMAN,A.M. and REID,J.K., (1976), On George's nested dissection method, *SIAM J. Numer. Anal. 13 no 5*, 686–695

DUFF,I.S., ERISMAN,A.M. and REID,J.K., (1986), *Direct methods for sparse matrices*, (Oxford University Press)

DUFF,I.S. and JOHNSON,S.L., (1989), Node ordering and concurrency in structurally symmetric sparse problems, *in Parallel supercomputing: methods, algorithms and applications, G. Carey Ed*

DUFF,I.S., GOULD,N.I., LESCRENIER,M. and REID,J.K., (1987), The multifrontal method in a parallel environment, *Report CSS 211, Harwell Lab.*

DUFF,I.S., GOULD,N.I., REID,J.K., SCOTT,J.A. and TURNER,K., (1990), The factorization of sparse symmetric indefinite matrices, *Report RAL–90–084 Rutherford Appleton Lab.*

DUFF,I.S., LAMINIE,J., LICHNEVSKY,A. and THOMASSET,F., (1987), An experiment with arithmetic precision in linear algebra computations, *Int. J. Numer. Meth. in Fluids 7*, 1077–1092

DUFF,I.S. and REID,J.K., (1974), A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination, *J. Inst. Maths. Applics. 14*, 281–291

DUFF,I.S. and REID,J.K., (1979), Some design features of a sparse matrix code, *ACM Trans. Math. Soft. 5 no 1*, 18–35

DUFF,I.S. and REID,J.K., (1983), The multifrontal solution of indefinite sparse symmetric linear systems, *ACM Trans. Math. Soft. 9*, 302–325

DUFF,I.S. and REID,J.K., (1984), The multifrontal solution of unsymmetric sets of linear systems, *SIAM J. Sci. Stat. Comput. 5 no 3*, 633–641

DUFF,I.S. and REID,J.K., (1993), MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations, *RAL–93–072 Rutherford Appleton Lab.*

DUFF,I.S. and REID,J.K., (1995), MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems, *RAL–95–001 Rutherford Appleton Lab.*

DUFF,I.S., REID,J.K., MUNSKGAARD,N. and NIELSEN,B., (1979), Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite, *J. Inst. Maths. Applics. 23*, 235–250

DUFF,I.S. and SCOTT,J.A., (1993), MA42– A new frontal code for solving sparse unsymmetric systems, *Report RAL–93–064, Rutherford Appleton Lab*

DUFF,I.S. and SCOTT,J.A., (1994),The use of multiple fronts in Gaussian elimination, *Report RAL–94–040, Rutherford Appleton Lab*

DUFF,I.S. and STEWART,G.W. Eds, (1979) *Sparse matrix proceedings 1978*, (SIAM)

ELDEN,L. and SVENSSON,G., (1991), Matrix computations on an SIMD parallel computer, *Report LiTH–MAT–R–1990–19, Linköping University*

EDELMAN,A., (1993), Large dense numerical linear algebra in 1993, the parallel computing influence, *Int. J. Supercomputer Appl. 7 no 2* 113–128

EDELMAN,A. and OHLROCH,M., (1991), Editor's Note in *SIAM J. Matrix Anal. Appl. 12*

EISENSTAT,S.C., HEATH,M.T., HENKEL,C.S. and ROMINE,C.H., (1988), Modified cyclic algorithms for solving triangular systems on distributed memory multiprocessors, *SIAM J. Sci. Stat. Comput. 9 no 3*, 589–600

EISENTAT,S.C., SCHULTZ,M.H. and SHERMAN,A.H., (1975), Efficient implementation of sparse symmetric Gaussian elimination, *in Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky Ed.*, 33–39

EISENTAT,S.C., SCHULTZ,M.H. and SHERMAN,A.H., (1975), Application of sparse matrix methods to partial differential equations, *in Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky Ed.*, 40–45

EISENTAT,S.C., SCHULTZ,M.H. and SHERMAN,A.H., (1981), Algorithms and data structures for sparse symmetric Gaussian elimination, *SIAM J. Sci. Stat. Comput. 2 no 2*, 225–237

ERISMAN,A.M., GRIMES,R.G., LEWIS,J.G., POOLE,W.G. and SIMON,H.D., (1987), Evaluation of orderings for unsymmetric sparse matrices, *SIAM J. Sci. Stat. Comput. 8 no 4*, 600–624

EVANS,D.J. Ed, (1985) *Sparsity and its applications*, (Cambridge University Press)

FADDEEV,D.K. and FADDEEVA, V.N., (1963) *Computational methods of linear algebra*, (W.H. Freeman and company)

FARHAT,C., (1988), A simple and efficient automatic FEM domain decomposer, *Computers and structures 28 no 5*, 579–602

FARHAT,C., (1990), Redesigning the skyline solver for parallel/vector supercomputers, *Int. J. High Speed Comp. 2 no 3*, 223–238

FIEDLER,M., (1986) *Special matrices and their applications in numerical mathematics*, (Martinus Nijhoff)

FIEDLER,M. and PTAK,V., (1962), On matrices with non–positive off–diagonal elements and positive principal minors, *Czechoslovak Math. J. 12*, 123–128

FONG,K. and JORDAN,T.L., (1977), Some linear algebraic algorithms and their performance on CRAY–1, *Report LA–6774, Los Alamos Scientific Laboratory*

FORSYTHE,G.E., (1953), Tentative classification of methods and bibliography on solving systems of linear equations, *Nat. Bur. Stand. Appl. Math. 29*, 1–28

FORSYTHE,G.E. and MOLER,C.B., *Computer solution of linear algebraic systems*, (Prentice–Hall)

FOSTER,L.V., (1994), Gaussian elimination with partial pivoting can fail in practice, *SIAM J. Matrix Anal. Appl. 15 no 4*, 1354–1362

FOX,L., HUSKEY,H.D. and WILKINSON,J.H., (1948), Notes on the solution of algebraic linear simultaneous equations, *Quart. J. Appl. Math. 1*, 149–173

FUNDERLIC,R.E., NEUMANN,M. and PLEMMONS,R.J., (1982), LU decompositions of generalized diagonally dominant matrices, *Numer. Math. 40*, 57–69

FUNDERLIC, R.E. and PLEMMONS,R.J, (1981), LU decomposition of M–matrices by elimination without pivoting, *Linear Algebra and its Appl. 41*, 41–99

GALLIVAN,K.A., HEATH,M.T., NG,E., ORTEGA,J.M., PEYTON,B.W., PLEMMONS, R.J., ROMINE,C.H., SAMEH,A.H. and VOIGT,R.C., (1990), *Parallel algorithms for matrix computations*, (SIAM)

GALLIVAN,K.A., PLEMMONS,R.J. and SAMEH,A.H., (1990), Parallel algorithms for dense linear algebra computations, *SIAM Rev. 32 no 1*, 54–135

GANTMACHER,F.R., (1959), *Matrix theory, vol I*, (Chelsea)

GEIST,G.A., (1985) Efficient parallel LU factorization with pivoting on a hypercube multiprocessor, *Report ORNL–6211 Oak Ridge Nat. Lab.*

GEIST,G.A. and NG,E., (1989), Task scheduling for parallel sparse Cholesky factorization, *Int. J. Parallel Prog. 18 no 4*, 291–314

GEIST,G.A. and ROMINE,C.H., (1988), LU factorization algorithms on distributed memory multiprocessor architectures, *SIAM J. Sci. Stat. Comput. 9 no 4*, 639–649

GEORGE,A., (1973), Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal. 10 no 2*, 345–363

GEORGE,A., (1974), On block elimination for sparse linear systems, *SIAM J. Numer. Anal. 11 no 3*, 585–603

GEORGE,A., (1977), Numerical experiments using dissection methods to solve n by n grid problems, *SIAM J. Numer. Anal. 14 no 2*, 161–179

GEORGE,A., (1980), An automatic one way dissection algorithm for irregular finite element problems, *SIAM J. Numer. Anal. 17 no 6*, 740–751

GEORGE,A., GILBERT,J.R. and LIU,J.W–H. Eds, (1993), *Graph theory and sparse matrix computation*, (Springer Verlag)

GEORGE,A., HEATH,M. and LIU,J.W–H., (1986), Parallel Cholesky factorization on a shared memory multiprocessor, *Linear Algebra and its Appl. 77*, 165–187

GEORGE,A., HEATH,M.T., LIU,J.W–H. and NG,E., (1988), Sparse Cholesky factorization on a local memory multiprocessor, *SIAM J. Sci. Stat. Comput. 9 no 2*, 327–340

GEORGE,A. and LIU,J.W–H., (1975), A note on fill for sparse matrices, *SIAM J. Numer. Anal. 12 no 3*, 452–455

GEORGE,A. and LIU,J.W–H., (1978), Algorithms for matrix partitioning and the numerical solution of finite element systems, *SIAM J. Numer. Anal. 15 no 2*, 297–327

GEORGE,A. and LIU,J.W–H., (1978), An automatic nested dissection algorithm for irregular finite element problems, *SIAM J. Numer. Anal. 15 no 5*, 1053–1069

GEORGE,A. and LIU, J.W–H., (1979), An implementation of a pseudoperipheral node finder, *ACM Trans. Math. Soft. 5 no 3*, 284–295

GEORGE,A. and LIU,J.W–H., (1979), The design of a user interface for a sparse matrix package, *ACM Trans. Math. Soft. 5 no 2*, 139–162

GEORGE,A. and LIU,J.W–H., (1980), A minimal storage implementation of the minimum degree algorithm, *SIAM J. Numer. Anal. 17 no 2*, 282–299

GEORGE,A. and LIU,J.W–H., (1980), An optimal algorithm for symbolic factorization of symmetric matrices, *SIAM J. Comp. 9 no 3*, 583–593

GEORGE,A. and LIU,J.W–H., (1980), A fast implementation of the minimum degree algorithm using quotient graphs, *ACM Trans. Math. Soft. 6 no 3*, 337–358

GEORGE,A. and LIU,J.W–H., (1981), *Computer solution of large sparse positive definite systems*, (Prentice–Hall)

GEORGE,A. and LIU,J.W–H., (1989), The evolution of the minimum degree ordering algorithm, *SIAM Rev. 31 no 1*, 1–19

GEORGE,A., LIU,J.W–H. and NG,E., (1988), A data structure for sparse QR and LU factorizations, *SIAM J. Sci. Stat. Comput. 9 no 1*, 100–121

GEORGE,A., LIU,J.W–H. and NG,E., (1989), Communication results for parallel sparse Cholesky factorization on a hypercube, *Parallel Comp. 10 no 3*, 287–298

GEORGE,A. and McINTYRE,D.R., (1978), On the application of the minimum degree algorithm to finite element systems, *SIAM J. Numer. Anal. 15 no 1*, 90–112

GEORGE,A. and NG,E., (1985), An implementation of Gaussian elimination with partial pivoting for sparse systems, *SIAM J. Sci. Stat. Comput. 6 no 2*, 390–409

GEORGE,A. and NG,E., (1987), Symbolic factorization for sparse Gaussian elimination with partial pivoting, *SIAM J. Sci. Stat. Comput. 8 no 6*, 877–898

GEORGE,A. and NG,E., (1988), On the complexity of sparse QR and LU factorization for finite element matrices, *SIAM J. Sci. Stat. Comput. 9 no 5*, 849–861

GEORGE,A., POOLE,W.G. and VOIGT,R.G., (1978), Incomplete nested dissection for solving n by n grid problems, *SIAM J. Numer. Anal. 15 no 4*, 662–673

GEORGE,A. and RASHWAN,H., (1980), On symbolic factorization of partitioned sparse symmetric matrices, *Linear Algebra and its Appl. 34*, 145–157

GIBBS,N.E., POOLE,W.G. and STOCKMEYER,P.K., (1976), An algorithm for reducing the bandwith and profile of a sparse matrix, *SIAM J. Numer. Anal. 13 no 2*, 236–250

GILBERT,J.R., (1994), Predicting structure in sparse matrix computations, *SIAM J. Matrix Anal. Appl. 15 no 1*, 62–79

GILBERT,J.R. and HAFSTEINSSON,H., (1990), Parallel symbolic factorization of sparse linear systems, *Parallel Comp. 14*, 151–162

GILBERT,J.R. and LIU,J.W–H., (1993), Elimination structures for unsymmetric sparse LU factors, *SIAM J. Matrix Anal. Appl. 14 no 2*, 334–352

GILBERT,J.R., NG,E.G. and PEYTON,B.W., (1994), An efficient algorithm to compute row and column counts for sparse Cholesky factorization, *SIAM J. Matrix Anal. Appl. 15 no 4*, 1075–1091

GILBERT,J.R. and PEIERLS,T., (1988), Sparse partial pivoting in time proportional to arithmetic operations, *SIAM J. Sci. Stat. Comput. 9 no 5*, 862–874

GILBERT,J.R. and SCHREIBER,R., (1992), Highly parallel sparse Cholesky factorization, *SIAM J. Sci. Stat. Comput. 13 no 5*, 1151–1172

GILBERT,J.R. and ZMIJEWSKI,E., (1990), A parallel graph partitioning algorithm for a message passing multiprocessor, *Int. J. Parallel Prog. 16*, 427–449

GILL,P.E., GOLUB,G.H., MURRAY,W. and SAUNDERS,M.A., (1974), Methods for modifying matrix factorizations, *Math. Comp. 28 no 126*, 505–535

GOLBERG,D., (1991), What every computer scientist should know about floating point arithmetic, *ACM Computing Surveys 23 no 1*, 5–48

GOLDSTINE,H.H., (1977), *A history of numerical analysis from the 16th through the 19th century*, (Springer)

GOLUB,G.H. Ed, (1984), Studies in numerical analysis, vol 24 (The Mathematical Association of America)

GOLUB,G.H. and MEURANT,G.A., (1983), *Résolution numérique des grands systèmes linéaires*, (Eyrolles)

GOLUB,G.H. and MEURANT,G.A., (1994), Matrices, moments and quadrature, *in ([207])*, 105–156

GOLUB,G.H. and ORTEGA,J.M., (1992), *Scientific computing and differential equations, an introduction to numerical methods*, (Academic Press)

GOLUB,G.H. and VAN LOAN,C., (1979), Unsymmetric positive definite linear systems, *Linear Algebra and its Appl. 28*, 85–97

GOLUB,G.H. and VAN LOAN,C., (1989), *Matrix computations, second edition*, (Johns Hopkins University Press)

GOULD,N., (1991), On growth in Gaussian elimination with complete pivoting, *SIAM J. Matrix Anal. Appl. 12 no 2*, 354–361

GRIFFITHS,D.F. and WATSON,G.A., (1989), *Numerical Analysis 1989, vol 228, Pitman research notes in mathematics series*, (Longman Scientific and Technical)

GRIFFITHS,D.F. and WATSON,G.A., (1994), *Numerical Analysis 1994, vol 303, Pitman research notes in mathematics series*, (Longman Scientific and Technical)

GRIMES,R.G., PIERCE,D.J. and SIMON,H.D., (1990), A new algorithm for finding a pseudoperipheral node in a graph, *SIAM J. Matrix Anal. Appl. 11 no 2*, 323–334

HADFIELD,S.M. and DAVIS,T.A., (1992), Analysis of potential parallel implementations of the unsymmetric pattern multifrontal method for sparse LU factorization, *Report TR–92–017, University of Florida*

HADFIELD,S.M. and DAVIS,T.A., (1994), A parallel unsymmetric pattern multifrontal method, *Report TR–94–??, University of Florida*

HADFIELD,S.M. and DAVIS,T.A., (1994), Potential and achievable parallelism in the unsymmetric pattern LU factorization method for sparse matrices, *Report TR–94–006, University of Florida*

HADFIELD,S.M. and DAVIS,T.A., (1994), Potential and achievable parallelism in the unsymmetric pattern LU factorization, *Report TR–94–027, University of Florida*

HAGER,W.W., (1984), Condition estimates, (1984), *SIAM J. Sci. Stat. Comput. 5 no 2*, 311–316

HARARY,F., (1971), Sparse matrices and graph theory, *in [306]*, 139–150

HARROD,W.J., (1986), LU decompositions of tridiagonal irreducible H–matrices, *SIAM J. Alg. Disc. Meth. 7 no 2*, 180–187

HEATH,M.T., NG,E. and PEYTON,B.W., (1991), Parallel algorithms for sparse linear systems, *SIAM Rev. 33 no 3*, 420–460

HEATH,M.T. and RAGHAVAN,P., (1993), Distributed solution of sparse linear systems, *Report UT CS–93–201, University of Tennessee*

HEATH,M.T. and RAGHAVAN,P., (1993), A cartesian parallel nested dissection algorithm, *SIAM J. Matrix Anal. Appl. 16 no 1*, 235–253

HEATH,M.T. and ROMINE,C.H., (1988), Parallel solution of triangular systems on distributed memory multiprocessors, *SIAM J. Sci. Stat. Comput. 9 no 3*, 558–588

HEGLAND,M., (1990), On the parallel solution of tridiagonal systems by wraparound partitioning and incomplete LU factorization, *Report 90–14, IPS, ETH*

HELLER,D., (1978), A survey of parallel algorithms in numerical linear algebra, *SIAM Rev. 20 no 4*, 740–777

HENDRICKSON,B.A. and WOMBLE,D.E., (1994), The torus–wrap mapping for dense matrix calculations on massively parallel computers, *SIAM J. Sci. Stat. Comput. 15 no 5*, 1201–1226

HENRICI,P., (1964), *Elements of numerical analysis*, (John Wiley)

HIGHAM,N.J., (1986), Efficient algorithms for computing the condition number of a tridiagonal matrix, *SIAM J. Sci. Stat. Comput. 7 no 1*, 150–165

HIGHAM,N.J., (1987), A survey of condition number estimation for triangular matrices, *SIAM Rev. 29 no 4*, 575–596

HIGHAM,N.J., (1989), The accuracy of solutions to triangular systems, *SIAM J. Numer. Anal. 26 no 5*, 1252–1265

HIGHAM,N.J., (1989), How accurate is Gaussian elimination?, *in ([206])*

HIGHAM,N.J., (1990), Bounding the error in Gaussian elimination for tridiagonal systems, *SIAM J. Matrix Anal. Appl. 11 no 4*, 521–530

HIGHAM,N.J. and HIGHAM,D.J., (1989), Large growth factors in Gaussian elimination with pivoting, *SIAM J. Matrix Anal. Appl. 10 no 2*, 155–164

HIGHAM,N.J. and HIGHAM,D.J., (1992), Backward error and condition of structured linear systems, *SIAM J. Matrix Anal. Appl. 13 no 1*, 162–175

HIGHAM,N.J. and POTHEN,A., (1994), Stability of the partitioned inverse method for parallel solution of sparse triangular systems, *SIAM J. Sci. Stat. Comput. 15 no 1*, 139–148

HOFFMAN,A.J., MARTIN,M.S. and ROSE.D.J., (1973), Complexity bounds for regular finite difference and finite element grids, *SIAM J. Numer. Anal. 10 no 2*, 364–369

HOOD,P., (1976), Frontal solution program for unsymmetric matrices, *Int. J. Num. Meth. Eng. 10*, 379–400

HOUSEHOLDER,A.S., (1953), *Principles of numerical analysis*, (McGraw–Hill)

HULBERT,L. and ZMIJEWSKI,E., (1991), Limiting communication in parallel sparse Cholesky factorization, *SIAM J. Sci. Stat. Comput. 12 no 5*, 1184–1197

IFRAH,G., (1994), *Histoire universelle des chiffres*, (Robert Laffont)

IPSEN,I.C., SAAD,Y. and SCHULTZ,M.H., (1986), Complexity of dense linear system solution on a multiprocessor ring, *Linear Algebra and its Appl. 77*, 205–239

IRONS,B.M., (1970), A frontal solution program for finite element analysis, *Int. J. Num. Meth. Eng. 2*, 5–32

JANKOWSKI,M. and WOZNIAKOWSKI,H., (1977), Iterative refinement implies numerical stability, *BIT 17*, 303–311

JENNINGS,A., (1977), *Matrix computation for engineers and scientists*, (John Wiley)

JENNINGS,A. and TUFF,A.D., (1971), A direct method for the solution of large sparse symmetric simultaneous equations, *in 306*, 97–104

JESS,J. and KEES,H., (1982), A data structure for parallel LU decomposition, *IEEE Trans. Comput. C–31*, 231–239

JOHNSSON,L., (1984), Odd–even cyclic reduction on ensemble architectures and the solution of tridiagonal systems of equations, *Report CSD/RR–339 Yale University*

JOHNSON,L., (1987), Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Stat. Comput. 8 no 3*, 354–392

JOHNSON,L. and MATHUR,K.K., (1990), Data structures and algorithms for the finite element method on a data parallel supercomputer, *Int. J. Num. Meth. Eng. 29*, 881–908

JONES,M.T. and PATRICK,M.L., (1994), Factoring symmetric indefinite matrices on high–performance architectures, *SIAM J. Matrix Anal. Appl. 15 no 1*, 273–283

JORDAN,T.L., (1974), Gaussian elimination for dense systems on STAR and a new parallel algorithm for diagonally dominant tridiagonal systems, *Report LA–5803, Los Alamos Scientific Laboratory*

KAHANER,D., MOLER,C.B. and NASH,S., (1988), *Numerical methods and software*, (Prentice–Hall)

KARYPIS,G. and KUMAR,V., (1994), A high performance sparse Cholesky factorization algorithm for scalable parallel computers, *Report 94–41, University of Minnesota*

KOWALIK,J.S. Ed, (1984), *High speed computation*, NATO ASI Series vol 7, (Springer)

LASCAUX,P. and THEODOR,R., (1993), *Analyse numérique matricielle appliquée à l'art de l'ingénieur, tome 1, second edition*, (Masson)

LAWSON,C.L., HANSON,R.J., KINCAID,D.R. and KROGH,F.T., (1979), Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Soft. 5 no 3*, 308–323

LEUZE,M.R., (1989), Independent set orderings for parallel matrix factorization by Gaussian elimination, *Parallel Comp. 10*, 177–191

LEWIS,J.G. and GRIMES,R.G., (1981), Condition number estimation for sparse matrices, *SIAM J. Sci. Stat. Comput. 2*, 384–388

LEWIS,J.G., PEYTON,B.W. and POTHEN,A., (1989), A fast algorithm for reordering sparse matrices for parallel factorization, *SIAM J. Sci. Stat. Comput. 10 no 6*, 1146–1173

LEWIS,J.G. and SIMON,H.D., (1988), The impact of hardware gather/scatter on sparse Gaussian elimination, *SIAM J. Sci. Stat. Comput. 9 no 2*, 304–311

LI,G. and COLEMAN,T.F., (1989), A new method for solving triangular systems on distributed memory message passing multiprocessors, *SIAM J. Sci. Stat. Comput. 10 no 2*, 382–396

LICHTENSTEIN,W. and JOHNSON,L., (1993) Block–cyclic dense linear algebra, *SIAM J. Sci. Stat. Comput. 14 no 6*, 1259–1288

LIPTON,R.J., ROSE,D.J. and TARJAN,R.E., (1979), Generalized nested dissection, *SIAM J. Numer. Anal. 16 no 2*, 346–358

LIPTON,R.J. and TARJAN,R.E., (1980), Applications of a planar separator theorem, *SIAM J. Comp. 9 no 3*, 615–627

LIU,J.W–H., (1985), Modification of the minimum degree algorithm by multiple elimination, *ACM Trans. Math. Soft. 11*, 141–153

LIU,J.W–H., (1987), An adaptive general sparse out–of–core Cholesky factorization scheme, *SIAM J. Sci. Stat. Comput. 9 no 4*, 585–599

LIU,J.W–H., (1987), A note on sparse factorization in a paging environment, *SIAM J. Sci. Stat. Comput. 8 no 6*, 1085–1088

LIU,J.W–H., (1988), Equivalent sparse matrix reordering by elimination tree rotations, *SIAM J. Sci. Stat. Comput. 9 no 3*, 424–444

LIU,J.W–H., (1989), Reordering sparse matrices for parallel elimination, *Parallel Comp. 11*, 73–91

LIU,J.W–H., (1989), The minimum degree ordering with constraints, *SIAM J. Sci. Stat. Comput. 10 no 6*, 1136–1145

LIU,J.W–H., (1990), The role of elimination trees in sparse factorization, *SIAM J. Matrix Anal. Appl. 11*, 134–172

LIU,J.W–H., (1992), The multifrontal method for sparse matrix solution: theory and practice, *SIAM Rev. 34 no 1*, 82–109

LIU,J.W–H. and MIRZAIAN,A., (1989), A linear reordering algorithm for parallel pivoting of chordal graphs, *SIAM J. Alg. Disc. Meth. 2*, 100–107

LIU,J.W–H.,NG,E.G. and PEYTON,B.W., (1993), On finding supernodes for sparse matrix computations, *SIAM J. Matrix Anal. Appl. 14 no 1*, 242–252

LIU,J.W–H. and SHERMAN,A.H., (1976), Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices, *SIAM J. Numer. Anal. 13 no 2*, 198–213

LORD,R.E., KOWALIK,J.S. and KUMAR,S.P., (1983), Solving linear algebraic equations on an MIMD computer, *J. ACM 30 no 1*, 103–117

MARKOWITZ,H.M., (1957) The elimination form of the inverse and its application to linear programming, *Manag. Sci. 3*, 255–269

MARRAKCHI,M. and ROBERT,Y., (1989), Optimal algorithms for Gaussian elimination on an MIMD computer, *Parallel Comp. 12 no 2*, 183–194

MARTIN,R.S. and WILKINSON,J.H., (1965), Symmetric decomposition of positive definite band matrices, *Numer. Math. 7*, 355–361

MARTIN,R.S. and WILKINSON,J.H., (1967), Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices, *Numer. Math. 9*, 279–301

MEIER,U., (1985), A parallel partition method for solving banded systems of linear equations, *Parallel Comp. 2*, 33–43

MELHEM,R.G., (1988), A modified frontal technique suitable for parallel systems, *SIAM J. Sci. Stat. Comput. 9 no 2*, 289–303

MEURANT,G., (1992), A review on the inverse of symmetric tridiagonal and block tridiagonal matrices, *SIAM J. Matrix Anal. Appl. 13 no 3*, 707–728

MICHIELSE,P.H. and VAN DER VORST,H.A., (1988), Data transport in Wang's partition method, *Parallel Comp. 7*, 87–95

MITCHISON,G. and DURBIN,R., (1986), Optimal numberings of an $N \times N$ array, *SIAM J. Alg. Disc. Meth. 7 no 4*, 571–582

MUNKSGAARD,N., (1979), New factorization codes for sparse, symmetric and positive definite matrices, *BIT 19*, 43–52

NASH,S.G. Ed, (1990), *A history of scientific computing*, (ACM Press)

NEAL,L. and POOLE,G., (1992), A geometric analysis of Gaussian elimination, II, *Linear Algebra and its Appl. 173*, 239–264

NG,E., (1993), Supernodal symbolic Cholesky factorization on a local memory multiprocessor, *Parallel Comp. 19*, 153–162

NG,E. and PEYTON,B.W., (1993), A supernodal Cholesky factorization algorithm for shared memory multiprocessors, *SIAM J. Sci. Stat. Comput. 14 no 4*, 761–769

NG,E. and PEYTON,B.W., (1993), Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Stat. Comput. 14 no 5*, 1034–1056

OETTLI,W. and PRAGER,W., (1964), Compatibility of approximate solution of linear equations with given error bounds for coefficients and right hand sides, *Numer. Math. 6*, 405–409

OGIELSKI,A.T. and AIELLO,W., (1993), Sparse matrix computations on parallel processor arrays, *SIAM J. Sci. Stat. Comput. 14 no 3*, 519–530

O'LEARY,D. and STEWART,G., (1986), Assignment and scheduling in parallel matrix factorization, *Linear Algebra and its Appl. 77*, 275–299

OLVER,F.W. and WILKINSON,J.H., (1982), A posteriori error bounds for Gaussian elimination, *J. Inst. Maths. Applics. 2*, 377–406

ORTEGA,J., (1988), *Introduction to parallel and vector solution of linear systems*, (Plenum Press)

ORTEGA,J.M., (1988), The ijk forms of factorization methods I. Vector computers, *Parallel Comp. 7 no 2*, 135–148

ORTEGA,J.M. and ROMINE,C.H., (1988), The ijk forms of factorization methods II. Parallel systems, *Parallel Comp. 7 no 2*, 149–162

ORTEGA,J., VOIGT,R.G. and ROMINE,C.H., (1988), A bibliography on parallel and vector numerical algorithms, *ICASE Report NASA 181764*

ØSTERBY,O. and ZLATEV,Z., (1983) *Direct methods for sparse matrices*, Lecture Notes in Computer Science 157, (Springer)

PARTER,S.V., (1961) The use of linear graphs in Gauss elimination, *SIAM Rev. 3 no 2*, 119–130

PEYTON,B., (1986), Some applications of clique trees to the solution of sparse linear systems, *Ph.D. thesis Clemson University*

PISSANETZKY,S., (1984) *Sparse matrix technology*, (Academic Press)

POOLE,G. and NEAL,L., (1991), A geometric analysis of Gaussian elimination, I, *Linear Algebra and its Appl. 149*, 249–272

POOLE,G. and NEAL,L., (1992), Gaussian elimination: when is scaling beneficial?, *Linear Algebra and its Appl. 162–164*, 309–324

POTHEN,A. and ALVARADO,F.L., (1992), A fast reordering algorithm for parallel sparse triangular solution, *SIAM J. Sci. Stat. Comput. 13 no 2*, 645–653

POTHEN,A., SIMON,H.D. and LIOU,K.P., (1990), Partitioning sparse matrices with eigenvectors of graphs, *SIAM J. Matrix Anal. Appl. 11 no 3*, 430–452

POTHEN,A. and SUN,C., (1993), A mapping algorithm for parallel sparse Cholesky factorization, *SIAM J. Sci. Stat. Comput. 14 no 5*, 1253–1257

RAGHAVAN,P., (1993), Distributed sparse Gaussian elimination and orthogonal factorization, *Report UT CS–93–203, University of Tennessee*

REID,J.K. Ed, (1971) *Large sparse sets of linear equations*, (Academic Press)

REID,J.K., (1971), A note on the stability of Gaussian elimination, *J. Inst. Maths. Applics. 8*, 374–375

REID,J.K., (1986), Sparse matrices, *Report CSS 201, Harwell Lab.*

RIGAL,J.L. and GACHES,J., (1967), On the compatibility of a given solution with the data of a linear system, *J. ACM 14*, 543–548

RODRIGUE,G. Ed, (1982) *Parallel Computations*, (Academic Press)

RODRIGUE,G. Ed, (1989) *Parallel processing for scientific computing*, (SIAM)

ROHN,J., (1990), Nonsingularity under data rounding, *Linear Algebra and its Appl. 139*, 171–174

ROMAN,J., (1985), Calcul de complexité relatifs à une méthode de dissection emboîtée, *Numer. Math. 47*, 175–190

ROSE,D.J., (1970), Triangulated graphs and the elimination process, *J. Math. Anal. Appl. 32*, 597–609

ROSE,D.J. and TARJAN,R.E., (1978), Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comp. 5 no 2*, 266–283

ROSE,D.J., TARJAN,R.E. and LUEKER,G.S., (1976), Algorithmic aspects of vertex elimination on directed graphs, *SIAM J. Appl. Math. 34 no 1*, 176–197

ROSE,D.J. and WILLOUGHBY,R.A. Eds, (1972) *Sparse matrices and their applications*, (Plenum Press)

ROTHBERG,E., (1993), Exploiting the memory hierarchy in sequential and parallel sparse Cholesky factorization, *Ph.D. thesis, Stanford University*

ROTHBERG,E. and GUPTA,A., (1994), An efficient block–oriented approach to parallel sparse Cholesky factorization, *SIAM J. Sci. Stat. Comput. 15 no 6*, 1413–1439

SAAD,Y., (1986), Communication complexity of the Gaussian elimination algorithm on multiprocessors, *Linear Algebra and its Appl. 77*, 315–340

SAAD,Y. and SCHULTZ,M.H., (1987), Parallel direct methods for solving banded linear systems, *Linear Algebra and its Appl. 88*, 623–650

SAAD,Y. and SCHULTZ,M.H., (1989), Data communication in parallel architectures, *Parallel Comp. 11 no 2*, 131–150

SAMEH,A.H. and KUCK,D.J., (1978), On stable parallel linear system solvers, *J. ACM 25 no 1*, 81–91

SAMEH,A.H. and BRENT,R.P., (1977), Solving triangular systems on a parallel computer, *SIAM J. Numer. Anal. 14 no 6*, 1101–1113

SCHREIBER,R.S., (1982), A new implementation of sparse Gaussian elimination, *ACM Trans. Math. Soft. 8*, 256–276

SHAPIRO A., (1985), Optimal block diagonal l_2–scaling of matrices, *SIAM J. Numer. Anal. 22 no 1*, 81–94

SHERMAN,A.H., (1978), Algorithms for sparse Gaussian elimination with partial pivoting, *ACM Trans. Math. Soft. 4 no 4*, 330–338

SIMON,H.D., (1991), Partitioning of unstructured problems for parallel processing, *Report RNR–91–008 NASA Ames*

SKEEL,R.D., (1979), Scaling for numerical stability in Gaussian elimination, *J. ACM 26 no 3*, 494–526

SKEEL,R.D., (1980), Iterative refinement implies numerical stability for Gaussian elimination, *Math. Comp. 35 no 151*, 817–832

STARK,S. and BERIS,A.N., (1992), LU decomposition optimized for a parallel computer with a hierarchical memory, *Parallel Comp. 18 no 9*, 959–972

STEWART,G.W., (1973), *Introduction to matrix computations*, (Academic Press)

STEWART,G.W., (1993), On the perturbation of LU, Cholesky and QR factorizations, *SIAM J. Matrix Anal. Appl. 14 no 4*, 1141–1145

STEWART,G.W., (1990), Communication and matrix computations on large message passing systems, *Parallel Comp. 16 no 1*, 27–40

STEWART,G.W., (1991), Maybe we should call it "Lagrangian elimination", *Na-net message of Friday 21 June 91*

STEWART,G.W., (1995), The triangular matrices of Gaussian elimination and related decompositions, *Report TR–95–91, University of Maryland*

STEWART,G.W., (1995), On the perturbation of LU and Cholesky factors, *Report TR–95–93, University of Maryland*

STEWART,G.W. and SUN,J–G., (1990), *Matrix perturbation theory*, (Academic Press)

STRANG,G., (1976), *Linear algebra and its applications*, (Academic Press)

STRASSEN,V., (1969), Gaussian elimination is not optimal, *Numer. Math. 13*, 354–356

STONE,H.S., (1975), Parallel tridiagonal equation solvers, *ACM Trans. Math. Soft. 1 no 4*, 289–307

TEWARSON,R.P., The product form of the inverse of sparse matrices and graph theory, *SIAM Rev. 9 no 1*, 91–99

TEWARSON,R.P., (1970), Computations with sparse matrices, *SIAM Rev. 12 no 4*, 527–543

TEWARSON,R.P., (1973), *Sparse matrices*, (Academic Press)

TINNEY,W.F. and WALKER,J.W., (1967), Direct solutions of sparse network equations by optimally ordered triangular factorization, *Proc. IEEE 55*, 1801–1809

TINNEY,W.F. and MEYER,W.S., (1973), Solution of large sparse systems by ordered triangular factorization, *IEEE Trans. Aut. Control AC–18 no 4*, 333–346

TISMENETSKY,M., (1986), A direct method for solving linear systems, *Technical report 88.179, IBM Israel*

TRAUB,J.F., (1973), *Complexity of sequential and parallel numerical algorithms*, (Academic Press)

TREFETHEN,L.N., (1985), Three mysteries of Gaussian elimination, *SIGNUM Newsletter 20 no 4*, 2–5

TREFETHEN,L.N. and SCHREIBER,R.S., (1990), Average–case stability of Gaussian elimination, *SIAM J. Matrix Anal. Appl. 11 no 3*, 335–360

VARGA,R.S. and CAI,D–Y., (1981), On the LU factorization of M–matrices, *Numer. Math. 38*, 179–192

VAN DER SLUIS,A., (1969), Condition numbers and equilibration of matrices, *Numer. Math. 14*, 14–23

VAN DER VORST,H.A., (1986), Analysis of a parallel solution method for tridiagonal systems, *Report 86–06, Delft University of Technology*

VAN DER VORST,H.A., (1988), Practical aspects of parallel scientific computing, *Fut. Gen. Comp. Sys. 4*, 285–291

VON NEUMANN,J. and GOLDSTINE,H.H., (1947), Numerical inverting of matrices of high order, *Bull. of the AMS 53 no 11*, 1021–1099

WILKINSON,J.H., (1961), Error analysis of direct methods of matrix inversion, *J. ACM 10*, 281–330

WILKINSON,J.H., (1965), *The algebraic eigenvalue problem*, (Oxford University Press)

WILKINSON,J.H., (1971), Modern error analysis, *SIAM Rev. 13*, 548–568

WRIGHT,S.J., (1991), Parallel algorithms for banded linear systems, *SIAM J. Sci. Stat. Comput. 12 no 4*, 824–842

WRIGHT,S.J., (1993), A collection of problems for which Gaussian elimination with partial pivoting is unstable, *SIAM J. Sci. Stat. Comput. 14 no 1*, 231–238

YANG,W.H., (1977), A method for updating Cholesky factorization of a band matrix, *Comp. Meth. Appl. Mech. Eng. 12*, 281–288

YANNAKAKIS,M., (1981), Computing the minimum fill–in is NP–complete, *SIAM J. Alg. Disc. Meth. 2 no 1*, 77–79

ZHANG,G. and ELMAN,H.C., (1992), Parallel sparse Cholesky factorization on a shared memory multiprocessor, *Parallel Comp. 18 no 9*, 1009–1022

ZLATEV,Z., (1980) On some pivotal strategies in Gaussian elimination by sparse technique, *SIAM J. Numer. Anal. 17 no 1*, 18–30

ZMIJEWSKI,E. and GILBERT,J.R., (1988), A parallel algorithm for sparse symbolic Cholesky factorization on a multiprocessor, *Parallel Comp. 7*, 199–210