Contents

Chapter 7

Direct Methods for Solving Linear System

The problem treated in this chapter is the numerical solution of a system $Ax = b$ of m linear equations in n variables. Systems of linear equations enters at some stage in almost every scientific computing problem. Often their solution is the dominating part of the work to solve the problem. Also the solution of a *nonlinear* problem is usually accomplished by solving a *sequence* of linear systems obtained, e.g., by Newton's method. Since algorithms for solving linear systems are perhaps the most widely used in scientific computing, it is of great importance that they are efficient and reliable.

The linear system $Ax = b$ has a unique solution for all vectors b only if the matrix A has full row and column rank. If $\text{rank}\,(A) < n$ the system either has many solutions (is underdetermined) or no solution (is overdetermined). Note that due to inaccuracy of the elements of A the rank may not be well defined. Under- and overdetermined systems will be treated in Chapter 8.

Two quite different classes of methods for solving systems of linear equations are of interest: **direct** methods and **iterative** methods. In a direct method the system is transformed by a sequence of elementary transformed into a system of simpler form, e.g., triangular or diagonal form, which can be solved in an elementary way. The most important direct method is Gaussian elimination, which is the method of choice when the matrix A is of full rank and has no special structure.

Disregarding rounding errors, direct methods give the exact solution after a finite number of arithmetic operations. Iterative methods, on the other hand, compute a sequence of approximate solutions, which (assuming exact arithmetic) in the limit converges to the exact solution x. Iterative methods have the advantage that in general they only require a subroutine for computing the matrix-vector product Ax for any given vector x. Hence they may be much more efficient than direct methods when the matrix A is large and matrix-vector multiplication cheap. The distinction is not sharp since iterative methods are usually applied to a so called preconditioned version of the system that may involve the solution of a sequence of simpler auxiliary systems by a direct method. Iterative methods and preconditioning techniques are treated in Chapter 11.

Many applications give rise to linear systems where the matrix has some special property that can be used to achieve savings in work and storage. An important case is when A is symmetric positive definite, when about half the work and storage can be saved; see Section 7.3. If only a small fraction of the elements in A are nonzero the linear system $Ax = b$ is called **sparse**. The simplest case is when A has a banded structure, but also more general sparsity patterns can be taken advantage of; see Section 7.8. Indeed, without the exploitation of sparsity many important problems would be intractable!

There are also some classes of structured matrices, which although not sparse, have a structure, which can be used to develop fast solution methods. One example is Vandermonde matrices, which are related to polynomial interpolation. Other important examples of structured matrices are Toeplitz and Hankel matrices. In all these instances the n^2 elements in the matrix are derived from only $(n-1)$ quantities.

Numerical methods for linear systems are a good illustration of the difference between classical mathematics and practical numerical analysis. Even though the mathematical theory is simple and the algorithms have been known for centuries, decisive progress in the development of algorithms has been made during the last few decades. It is important to note that methods, which are perfectly acceptable for theoretical use, may be useless for the numerical solution. For example, the explicit determinant formula (Cramer's rule) for the inverse matrix and for the solution of linear systems of equations is extremely uneconomical except for matrices of order two or three, and matrices of very special structure.

Since critical details in the algorithms can influence the efficiency and accuracy in a way the beginner can hardly expect the reader is strongly advised to use the efficient and well-tested software available in the public domain; see Section 7.7.3.

The emphasis in this chapter will be on algorithms for *real* linear systems, since (with the exception for Hermitian systems) these occur most commonly in applications. However, all algorithms given can readily be generalized to the complex case.

7.1 Linear Algebra and Matrix Analysis

7.1.1 Linear Vector Spaces

We denote the field of real numbers by $\mathbf{R}$ and $\mathbf{R}^n$ is the vector space of n-tuples of real numbers. The operation addition and scalar multiplication are defined for all $v \in \mathbf{R}^n$, and have the following properties:

1. the following distributive properties hold:

$$\alpha(v+w) = \alpha v + \alpha w, \qquad (\alpha+\beta)v = \alpha v + \beta v,$$

   for all $\alpha, \beta \in \mathbf{K}$ and v, $w \in \mathbf{W}$.

2. there is an element $0 \in \mathbf{W}$ called the **null vector** such that $v + 0 = v$ for all $v \in \mathbf{R}^n$;

3. for each vector v there exists a vector $-v$ such that $v + (-v) = 0$;

4. $0 \cdot v = 0$ and $1 \cdot v = v$ where 0 and 1 are the zero and unity in $\mathbf{K}$.

Similar properties hold for the vector space $\mathbf{C}^n$ of n-tuples of elements of the field of complex numbers by $\mathbf{C}$.

If $\mathbf{W} \subset \mathbf{V}$ is a vector space then $\mathbf{W}$ is called a **vector subspace** of $\mathbf{V}$. The set of all linear combinations of $v_1, \ldots, v_k \in \mathbf{V}$ form a vector subspace denoted by

$$\text{span}\,\{v_1, \ldots, v_k\} = \sum_{i=1}^{k} \alpha_i v_i, \quad \alpha_i \in \mathbf{K}, \quad i = 1 : k.$$

If $\mathbf{S_1}, \ldots, \mathbf{S_k}$ are vector subspaces of $\mathbf{V}$ then their sum defined by

$$S = \{v_1 + \cdots + v_k |\ v_i \in \mathbf{S_i},\ i = 1 : k\}$$

is also a vector subspace. The intersection T of a set of vector subspaces is also a subspace,

$$T = \mathbf{S_1} \cap \mathbf{S_2} \cdots \cap \mathbf{S_k}.$$

(The union of vector spaces is generally no vector space.) If the intersection of the subspaces are empty, $\mathbf{S_i} \cap \mathbf{S_j} = 0$, $i \neq j$, then the sum of the subspaces is called their **direct sum** and denoted by

$$\mathbf{S} = \mathbf{S_1} \oplus \mathbf{S_2} \cdots \oplus \mathbf{S_k}.$$

A set of vectors $\{v_1, v_2, \ldots, v_k\}$ in $\mathbf{V}$ is said to be **linearly independent** if

$$\sum_{i=1}^{k} c_i v_i = 0, \quad \Rightarrow \quad c_1 = c_2 = \cdots = c_k = 0.$$

Otherwise, if a nontrivial linear combination of $v_1, \ldots, v_k$ is zero, the vectors are said to be linearly dependent. Then at least one vector v_i will be a linear combination of the rest.

A **basis** in $\mathbf{V}$ is any set of linearly independent vectors $v_1, v_2, \ldots, v_n \in \mathbf{V}$ such that all vectors $v \in \mathbf{V}$ can be uniquely decomposed as

$$v = \sum_{i=1}^{n} \xi_i v_i.$$

The scalars ξ_i are called the components or coordinates of v with respect to the basis $\{v_i\}$.

If the vector space $\mathbf{V}$ has a basis of k vectors, then every system of linearly independent vectors of $\mathbf{V}$ has at most k elements and any other basis of $\mathbf{V}$ has the same number k of elements. The number k is called the **dimension** of $\mathbf{V}$ and denoted by $\dim(\mathbf{V})$.

The **standard basis** for $\mathbf{C}^n$ is the set of unit vectors $e_1, e_2, ..., e_n$, where the jth component of e_i equals 1 if $j = i$, and 0 otherwise. We shall use the same name for a vector as for its coordinate representation by a column vector, with respect to the standard basis. For the the vector space $\mathcal{P}_n$ of polynomials of degree less than n monomials $1, x, \ldots, x^{n-1}$ form a basis.

7.1.2 Matrix and Vector Algebra

A **matrix** A is a collection of $m \times n$ numbers ordered in m rows and n columns

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

We write $A \in \mathbf{R}^{m\times n}$, where $\mathbf{R}^{m\times n}$ denotes the set of all real $m \times n$ matrices. If $m = n$, then the matrix A is said to be square and of order n. If $m \neq n$, then A is said to be rectangular. The empty matrix is a matrix of dimension 0×0 with no columns and no rows. Empty matrices are convenient to use as place holders.

A **column vector** is a matrix consisting of just one column and we write $x \in \mathbf{R}^m$ instead of $x \in \mathbf{R}^{m\times 1}$. Similarly a **row vector** is a matrix consisting of just one row.

A **linear map** from the vector space $\mathbf{C}^n$ to $\mathbf{C}^m$ is a function f such that

$$f(\alpha v + \beta w) = \alpha f(u) + \beta f(v)$$

for all $\alpha, \beta \in \mathbf{K}$ and $u, v \in \mathbf{C}^n$. Let x and y be the column vectors representing the vectors v and $f(v)$, respectively , using the standard basis of the two spaces. Then there is a unique matrix $A \in \mathbf{C}^{m\times n}$ representing this map such that

$$y = Ax.$$

This gives a link between linear maps and matrices.

We will follow a convention introduced by Householder[1] and use capital letters (e.g. A, B) to denote matrices. The corresponding lower case letters with subscripts ij then refer to the (i, j) component of the matrix (e.g. a_{ij}, b_{ij}). Greek letters $\alpha, \beta, \dots$ are usually used to denote scalars. Column vectors are usually denoted by lower case letters (e.g. x, y).

Two matrices in $\mathbf{R}^{m\times n}$ are said to be **equal**, $A = B$, if

$$a_{ij} = b_{ij}, \quad i = 1 : m, \quad j = 1 : n.$$

The basic operations with matrices are defined as follows. The product of a matrix A with a scalar α is

$$B = \alpha A, \qquad b_{ij} = \alpha a_{ij}.$$

The **sum** of two matrices A and B in $\mathbf{R}^{m\times n}$ is

$$C = A + B, \qquad c_{ij} = a_{ij} + b_{ij}. \tag{7.1.1}$$

As a special case of the multiplication rule, if $A \in \mathbf{R}^{m\times n}$, $x \in \mathbf{R}^n$, then

$$y = Ax \in \mathbf{R}^m, \qquad y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1 : m.$$

[1] A. S. Householder 1904–1993, American mathematician at Oak Ridge National Laboratory and University of Tennessee. He pioneered the use of matrix factorization and orthogonal transformations in numerical linear algebra.

The **product** of two matrices A and B is defined if and only if the number of columns in A equals the number of rows in B. If $A \in \mathbf{R}^{m\times n}$ and $B \in \mathbf{R}^{n\times p}$ then

$$C = AB \in \mathbf{R}^{m\times p}, \qquad c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}, \tag{7.1.2}$$

and can be computed with mnp multiplications.

Matrix multiplication is associative and distributive,

$$A(BC) = (AB)C, \qquad A(B + C) = AB + AC,$$

but not *not commutative*. The product BA is not even defined unless $p = m$. Then the matrices $AB \in \mathbf{R}^{m\times m}$ and $BA \in \mathbf{R}^{n\times n}$ are both square, but if $m \neq n$ of different orders. In general, $AB \neq BA$ even when $m = n$. If $AB = BA$ the matrices are said to **commute**.

Example 7.1.1.

If $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$ and $C \in \mathbf{R}^{p\times q}$, then the product $M = ABC \in \mathbf{R}^{m\times q}$ is defined. computing M as $(AB)C$ requires $mp(n + q)$ operations, whereas using $A(BC)$ requires $nq(m + p)$ operations. These numbers can be very different! For example, if A and B are square $n \times n$ matrices and x a column vector of length n then computing the product ABx as $(AB)x$ requires $n^3 + n^2$ operations whereas $A(Bx)$ only requires $2n^2$ operations. When $n \gg 1$ this makes a great difference!

It is useful to define also **array operations**, which are carried out element-by-element on vectors and matrices. Following the convention in MATLAB we denote array multiplication and division by $.*$ and $./$, respectively. If A and B have the same dimensions $A\,.*\,B$ is the matrix with elements equal to $a_{ij} \cdot b_{ij}$ and $A./B$ has elements a_{ij}/b_{ij}. (Note that for $+, -$ array operations coincides with matrix operations so no distinction is necessary.)

The **transpose** A^T of a matrix $A = (a_{ij})$ is the matrix whose rows are the columns of A, i.e., if $C = A^T$ then $c_{ij} = a_{ji}$. For a complex matrix we denote by A^H the complex conjugate transpose of A

$$A = (a_{ij}), \qquad A^H = (\bar{a}_{ji}),$$

and it holds that $(AB)^H = B^H A^H$.

Row vectors are obtained by transposing column vectors (e.g. x^T, y^T). For the transpose of a product we have

$$(AB)^T = B^T A^T,$$

i.e., the product of the transposed matrices in *reverse order*.

We recall that $r = \operatorname{rank}(A)$ is the number of linearly independent columns which is the same as the number of linearly independent rows of A. A square matrix A of order n is said to **nonsingular** if $\operatorname{rank}(A) = n$. It is left as an exercise to show that the rank of a sum of two matrices satisfies

$$\operatorname{rank}(A + B) \leq \operatorname{rank}(A) + \operatorname{rank}(B), \tag{7.1.3}$$

and the rank of a product of two matrices satisfies

$$\operatorname{rank}(AB) \le \min\{\operatorname{rank}(A), \operatorname{rank}(B)\}. \tag{7.1.4}$$

If A is square and nonsingular there exists an **inverse matrix** denoted by A^{-1} with the property that

$$A^{-1}A = AA^{-1} = I.$$

By A^{-T} we will denote the matrix $(A^{-1})^T = (A^T)^{-1}$. For the inverse of a product of two matrices we have

$$(AB)^{-1} = B^{-1}A^{-1},$$

where the product of the inverse matrices are taken in reverse order.

The absolute value of a matrix A and vector b is defined by

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

We also introduce the partial ordering "$\le$" for matrices A, B and vectors x, y, which is to be interpreted component-wise[2]

$$A \le B \iff a_{ij} \le b_{ij}, \qquad x \le y \iff x_i \le y_i.$$

Further, it is easy to show that if $C = AB$, then

$$|c_{ij}| \le \sum_{k=1}^{n} |a_{ik}|\,|b_{kj}|,$$

and hence $|C| \le |A|\,|B|$. A similar rule holds for matrix-vector multiplication.

The Euclidean **inner product** of two vectors x and y in $\mathbf{R}^n$ is given by

$$x^T y = \sum_{i=1}^{n} x_i y_i = y^T x, \tag{7.1.5}$$

and the **Euclidian length** of the vector x is

$$\|x\|_2 = (x^T x)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}. \tag{7.1.6}$$

The **outer product** of $x \in \mathbf{R}^m$ and $y \in \mathbf{R}^n$ is the matrix

$$xy^T = \begin{pmatrix} x_1 y_1 & \dots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \dots & x_m y_n \end{pmatrix} \in \mathbf{R}^{m \times n}. \tag{7.1.7}$$

For some problems it is more relevant and convenient to work with complex vectors and matrices. We denote by $\mathbf{C}^{n \times m}$ the vector space of all complex $n \times m$

[2]Note that $A \le B$ in other contexts means that $B - A$ is positive semidefinite.

matrices whose components are complex numbers.[3] Most concepts introduced here carry over to complex matrices. Addition and multiplication of vectors and matrices follow the same rules as before. The **Hermitian** inner product of two vectors x and y in $\mathbf{C}^n$ is defined by

$$x^H y = \sum_{k=1}^{n} \bar{x}_k y_k, \tag{7.1.8}$$

where $x^H = (\bar{x}_1, \ldots, \bar{x}_n)$ and $\bar{x}_k$ denotes the complex conjugate of x_k. Hence $x^H y = \overline{y^H x}$ and $x^H x$ is a real number.

Any matrix D for which $d_{ij} = 0$ if $i \neq j$ is called a **diagonal matrix**. If $x \in \mathbf{R}^n$ is a vector then $D = \operatorname{diag}(x) \in \mathbf{R}^{n\times n}$ is the diagonal matrix formed by the elements of x. For a matrix $A \in \mathbf{R}^{n\times n}$ the elements a_{ii}, $i = 1 : n$, form the **main diagonal** of A, and we write

$$\operatorname{diag}(A) = \operatorname{diag}(a_{11}, a_{22}, \ldots, a_{nn}).$$

For $k = 1 : n-1$ the elements $a_{i,i+k}$ $(a_{i+k,i})$, $i = 1 : n-k$ form the kth **superdiagonal** (**subdiagonal**) of A. The elements $a_{i,n-i+1}$, $i = 1 : n$ form the (main) **antidiagonal** of A.

The **unit matrix** $I_n \in \mathbf{R}^{n\times n}$ is defined by

$$I_n = \operatorname{diag}(1, 1, \ldots, 1) = (e_1, e_2, \ldots, e_n),$$

and the k-th column of I_n is denoted by e_k. We have that $I_n = (\delta_{ij})$, where δ_{ij} is the **Kronecker symbol** $\delta_{ij} = 0, i \neq j$, and $\delta_{ij} = 1, i = j$. For all square matrices of order n it holds $AI_n = I_n A = A$. If the size of the unit matrix is obvious we delete the subscript and just write I.

Definition 7.1.1.

A matrix A is said to have **upper bandwidth** *r and* **lower bandwidth** *s if*

$$a_{ij} = 0, \quad j > i + r, \qquad a_{ij} = 0, \quad i > j + s,$$

respectively. This means that the number of non-zero diagonals above and below the main diagonal are r and s respectively. The maximum number of nonzero elements in any row is then $w = r + s + 1$, which is the **bandwidth** *of A.*

For a matrix $A \in \mathbf{R}^{m\times n}$ which is not square we define the bandwidth as

$$w = \max_{1\le i\le m} \{j - k + 1 \mid a_{ij} a_{ik} \neq 0\}.$$

Note that the bandwidth of a matrix depends on the ordering of its rows and columns. An important, but hard, problem is to find an optimal ordering of columns that minimize the bandwidth. However, there are good heuristic algorithms that can be used in practice and give almost optimal results; see Section 7.6.3.

[3] In MATLAB the only data type used is a matrix with either real or complex elements.

Several classes of band matrices that occur frequently have special names. Thus, a matrix for which $r = s = 1$ is called **tridiagonal**, if $r = 0$, $s = 1$ ($r = 1$, $s = 0$) it is called lower (upper) **bidiagonal** etc. A matrix with $s = 1$ ($r = 1$) is called an upper (lower) **Hessenberg** matrix.

A matrix A is called **symmetric** if its elements are symmetric about its main diagonal, i.e. $a_{ij} = a_{ji}$, $1 \le i < j \le n$, or equivalently $A^T = A$. A complex matrix A is called **Hermitian** if $A^H = A$ and **skew-Hermitian** if $A^H = -A$. The product of two Hermitian matrices is symmetric if and only if A and B commute, that is, $AB = BA$. If $A^T = -A$, then A is called **skew-symmetric**.

A square matrix A is called **persymmetric** if it is symmetric about its antidiagonal, i.e., $a_{ij} = a_{n-j+1,n-i+1}$.

7.1.3 Determinants and Permutations

The classical definition of the determinant requires some elementary facts about permutations, which we now state. Let $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a permutation of the integers $\{1, 2, \ldots, n\}$. The pair α_r, α_s, $r < s$ is said to form an inversion in the permutation if $\alpha_r > \alpha_s$. For example, in the permutation $\{2, \ldots, n, 1\}$ there are $(n-1)$ inversions $(2,1), (3,1), \ldots, (n,1)$. A permutation α is said to be even and $\operatorname{sign}(\alpha) = 1$ if it contains an even number of inversions; otherwise the permutation is odd and $\operatorname{sign}(\alpha) = -1$. The product of two permutations σ and τ is the composition $\sigma\tau$ defined by

$$\sigma\tau(i) = \sigma[\tau(i)], \quad i = 1 : n.$$

A **transposition** τ is a permutation which only interchanges two elements. Any permutation can be decomposed into a sequence of transpositions, but this decomposition is not unique.

Lemma 7.1.2.

A transposition τ of a permutation will change the number of inversions in the permutation by an odd number and thus $\operatorname{sign}(\tau) = -1$.

Proof. If τ interchanges two adjacent elements α_r and α_{r+1} in the permutation $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, this will not affect inversions in other elements. Hence the number of inversions increases by 1 if $\alpha_r < \alpha_{r+1}$ and decreases by 1 otherwise. Suppose now that τ interchanges α_r and α_{r+q}. This can be achieved by first successively interchanging α_r with α_{r+1}, then with α_{r+2}, and finally with α_{r+q}. This takes q steps. Next the element α_{r+q} is moved in $q-1$ steps to the position which α_r previously had. In all it takes an *odd number* $2q-1$ of transpositions of adjacent elements, in each of which the sign of the permutation changes. □

Definition 7.1.3.

The **determinant** *of a square matrix $A \in \mathbf{R}^{n\times n}$ is the scalar*

$$\det(A) = \sum_{\alpha \in S_n} \operatorname{sign}(\alpha)\, a_{1,\alpha_1} a_{2,\alpha_2} \cdots a_{n,\alpha_n}, \tag{7.1.9}$$

where the sum is over all permutations of the set $\{1,\dots,n\}$ *and* $\operatorname{sign}(\alpha) = \pm 1$ *according to whether* α *is an even or odd permutation.*

Note that there are $n!$ terms in (7.1.9) and each term contains exactly one factor from each row and each column in A. It follows easily that $\det(\alpha A) = \alpha^n \det(A)$ and $\det(A^T) = \det(A)$. The matrix A is nonsingular if and only if $\det(A) \neq 0$.

Theorem 7.1.4.

Let the matrix A *be nonsingular. Then the solution of the linear system* $Ax = b$ *can be expressed as*

$$x_i = \det(A_j)/\det(A), \quad i = 1:n, \tag{7.1.10}$$

where A_j *is the matrix* A *where the* j*th column has been replaced by the right hand side* b*.*

The expression (7.1.10) is known as **Cramer's rule**.[4] Although elegant, it is both computationally expensive and numerically instable. It should not be used for numerical computation except in very special cases.

The direct use of the definition (7.1.9) to evaluate $\det(A)$ would require about $nn!$ operations, which rapidly becomes infeasible as n increases. A much more efficient way to compute $\det(A)$ is by repeatedly using the following properties:

Theorem 7.1.5.

(i) *The value of the* $\det(A)$ *is unchanged if a row (column) in* A *multiplied by a scalar is added to another row (column).*

(ii) *The determinant of a triangular matrix equals the product of the elements in the main diagonal, i.e., if* U *is upper triangular*

$$\det(U) = u_{11}u_{22}\cdots u_{nn}.$$

(iii) *If two rows (columns) in* A *are interchanged the value of* $\det(A)$ *is multiplied by* (-1)*.*

(iv) *The product rule* $\det(AB) = \det(A)\det(B)$*.*

For a linear system $Ax = b$ there are three possibilities: it may have no solution, one unique solution, or an infinite set of solutions. If $b \in \mathcal{R}(A)$, or equivalently $\operatorname{rank}(A,\ b) = \operatorname{rank}(A)$, the system is said to be **consistent**. If $r = m$ then $\mathcal{R}(A)$ equals $\mathbf{R}^m$ and the system is consistent for all b. Clearly a consistent linear system always has *at least one solution* x.

The corresponding homogeneous linear system $Ax = 0$ is satisfied by any $x \in \mathcal{N}(A)$ and thus has $(n - r)$ linearly independent solutions. It follows that if a solution to an inhomogeneous system $Ax = b$ exists, it is unique only if $r = n$, whence $\mathcal{N}(A) = \{0\}$.

[4] Named after the Swiss mathematician Gabriel Cramer 1704–1752.

7.1.4 Partitioning and Block Matrices

A matrix formed by the elements at the intersection of a set of rows and columns of a matrix A is called a **submatrix**. For example, the matrices

$$\begin{pmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{pmatrix}, \qquad \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix},$$

are submatrices of A. The second submatrix is called a contiguous submatrix since it is formed by contiguous elements of A.

Definition 7.1.6.

A **submatrix** *of $A = (a_{ij}) \in \mathbf{R}^{m\times n}$, is a matrix $B \in \mathbf{R}^{p\times q}$ formed by selecting p rows and q columns of A,*

$$B = \begin{pmatrix} a_{i_1 j_1} & a_{i_1 j_2} & \cdots & a_{i_1 j_q} \\ a_{i_2 j_1} & a_{i_2 j_2} & \cdots & a_{i_2 j_q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_p j_1} & a_{i_p j_2} & \cdots & a_{i_p j_q} \end{pmatrix},$$

where

$$1 \le i_1 \le i_2 \le \cdots \le i_p \le m, \quad 1 \le j_1 \le j_2 \le \cdots \le j_q \le n.$$

If $p = q$ and $i_k = j_k$, $k = 1 : p$, then B is a **principal submatrix** *of A. If in addition, $i_k = j_k = k$, $k = 1 : p$, then B is a* **leading** *principal submatrix of A.*

It is often convenient to think of a matrix (vector) as being built up of contiguous submatrices (subvectors) of lower dimensions. This can be achieved by **partitioning** the matrix or vector into blocks. We write, e.g.,

$$A = \begin{array}{c} \\ p_1\{ \\ p_2\{ \\ \vdots \\ p_M\{ \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & \cdots & q_N \end{array} \\ \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MN} \end{pmatrix} \end{array}, \qquad x = \begin{array}{c} p_1\{ \\ p_2\{ \\ \vdots \\ p_M\{ \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \tag{7.1.11}$$

where A_{IJ} is a matrix of dimension $p_I \times q_J$. We call such a matrix a **block matrix**. The partitioning can be carried out in many ways, and is often suggested by the structure of the underlying problem. For square matrices the most important case is when $M = N$, and $p_I = q_I$, $I = 1 : N$. Then the diagonal blocks A_{II}, $I = 1 : N$, are square matrices.

The great convenience of block matrices lies in the fact that the operations of addition and multiplication can be performed by treating the blocks A_{IJ} as *non-commuting scalars* and applying the definitions (7.1.1) and (7.1.2). Therefore many algorithms defined for matrices with scalar elements have another simple generalization to partitioned matrices. Of course the dimensions of the blocks must correspond in such a way that the operations can be performed. When this is the case, the matrices are said to be partitioned **conformally**.

Let $A = (A_{IK})$ and $B = (B_{KJ})$ be two block matrices of block dimensions $M \times N$ and $N \times P$ respectively, where the partitioning corresponding to the index K is the same for each matrix. Then we have $C = AB = (C_{IJ})$, where

$$C_{IJ} = \sum_{K=1}^{N} A_{IK} B_{KJ}, \quad 1 \le I \le M, \quad 1 \le J \le P.$$

Often it is convenient to partition a matrix into rows or columns. Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$. Then the matrix product $C = AB \in \mathbf{R}^{m\times p}$ can be written

$$C = AB = (a_1\ a_2\ \cdots\ a_n) \begin{pmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_n^T \end{pmatrix} = \sum_{k=1}^{n} a_k b_k^T, \tag{7.1.12}$$

where $a_k \in \mathbf{R}^m$ are the columns of A and $b_k^T \in \mathbf{R}^p$ the rows in B. Note that each term in the sum of (7.1.12) is an *outer product*. The more common inner product formula is obtained from the partitioning

$$C = AB = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{pmatrix} (b_1\ b_2\ \cdots\ b_p) = (c_{ij}), \qquad c_{ij} = a_i^T b_j. \tag{7.1.13}$$

with $a_i, b_j \in \mathbf{R}^n$. Note that when the matrices A and B only have relatively few nonzero elements *the outer product formula* (7.1.12) *is a more efficient way to compute* AB! Further, if A and x are as in (7.1.11) then the product $z = Ax$ is a block vector with blocks

$$z_I = \sum_{K=1}^{N} A_{IK} x_K, \quad I = 1 : M.$$

Example 7.1.2.

Assume that the matrices A and B are conformally partitioned into 2×2 block form. Then

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}. \tag{7.1.14}$$

Be careful to note that since matrix multiplication is not commutative the *order* of the factors in the products cannot be changed! In the special case of block upper triangular matrices this reduces to

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} = \begin{pmatrix} R_{11}S_{11} & R_{11}S_{12} + R_{12}S_{22} \\ 0 & R_{22}S_{22} \end{pmatrix}.$$

Note that the product is again block upper triangular and its block diagonal simply equals the products of the diagonal blocks of the factors.

Let

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \quad U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}, \tag{7.1.15}$$

be 2×2 block lower and upper triangular matrices, respectively. For an upper block triangular matrix with square diagonal blocks $U_{II}, I = 1 : N$ we have

$$\det(U) = \det(U_{11}) \det(U_{22}) \cdots \det(U_{NN}), \tag{7.1.16}$$

Hence U is nonsingular if and only if all its diagonal blocks are nonsingular. Since $\det(L) = \det(L^T)$, a similar result holds for a lower block triangular matrix.

If L and U in (7.1.15) are nonsingular with square diagonal blocks, then their inverses are given by

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}, \qquad U^{-1} = \begin{pmatrix} U_{11}^{-1} & -U_{11}^{-1} U_{12} U_{22}^{-1} \\ 0 & U_{22}^{-1} \end{pmatrix}. \tag{7.1.17}$$

This can be verified by forming the products $L^{-1}L$ and $U^{-1}U$ using the rule for multiplying partitioned matrices.

7.1.5 Modified Linear Systems

Consider the block 2×2 linear system

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

where A and D are square matrices, and A nonsingular. We can eliminate the x variables by premultiplying the first block by CA^{-1} and subtracting from the second block equations, giving

$$Sy = c - CA^{-1}b, \quad S = D - CA^{-1}B. \tag{7.1.18}$$

The matrix S is called the **Schur complement**of A.[5]

The elimination can be expressed in matrix form as

$$\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & S \end{pmatrix}, \tag{7.1.19}$$

Inverting the block lower triangular matrix on the left hand side using (7.1.17) we obtain the block LU factorization

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & 0 \\ CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ 0 & S \end{pmatrix}. \tag{7.1.20}$$

[5]Issai Schur (1875–1941) was born in Russia but studied at the University of Berlin, where he became full professor in 1919. Schur is mainly known for his fundamental work on the theory of groups but he also worked in the field of matrices.

From $M^{-1} = (LU)^{-1} = U^{-1}L^{-1}$ using the formulas (7.1.17) for the inverses of 2×2 block triangular matrices we get the **Schur–Banachiewicz** inverse formula[6]

$$M^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}. \tag{7.1.21}$$

Similarly, assuming that D is nonsingular, we can factor M into a product of a block upper and a block lower triangular matrix

$$M = \begin{pmatrix} I & BD^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} T & 0 \\ C & D \end{pmatrix}, \qquad T = A - BD^{-1}C, \tag{7.1.22}$$

where T is the Schur complement of D in M. (This is equivalent to block Gaussian elimination in reverse order.) From this factorization an alternative expression of M^{-1} can be derived,

$$M^{-1} = \begin{pmatrix} T^{-1} & -T^{-1}BD^{-1} \\ -D^{-1}CT^{-1} & D^{-1} + D^{-1}CT^{-1}BD^{-1} \end{pmatrix}. \tag{7.1.23}$$

If A and D are nonsingular the two triangular factorizations (7.1.20) and (7.1.22) both exist. Then, using (7.1.16), it follows that

$$\det(M) = \det(A - BD^{-1}C)\det(D) = \det(A)\det(D - CA^{-1}B).$$

In the special case that $D^{-1} = \lambda$, $B = x$, and $B = y$, this gives

$$\det(A - \lambda xy^T) = \det(A)(1 - \lambda y^T A^{-1}x). \tag{7.1.24}$$

This shows that $\det(A - \lambda xy^T) = 0$ if $\lambda = 1/y^T A^{-1}x$, a fact which is useful for the solution of eigenvalue problems.

The following formula gives an expression for the inverse of a matrix A after it is modified by a matrix of rank p, and is very useful in situations where $p \ll n$.

Theorem 7.1.7. *[Max A. Woodbury [69]]*

Let A and D be square nonsingular matrices and let B and C be matrices of appropriate dimensions such that $(A - BD^{-1}C)$ exists and is nonsingular. Then

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}, \tag{7.1.25}$$

which is the **Woodbury formula**.

Proof. The result follows directly by equating the $(1,1)$ blocks in the inverse M^{-1} in (7.1.21) and (7.1.23). □

[6] Tadeusz Banachiewicz (1882–1954) Polish astronomer and mathematician. In 1919 he became the director Cracow Observatory. He developed in 1925 a special kind of matrix algebra for "cracovians", which brought him international recognition.

If we specialize the Woodbury formula to the case where D is a scalar we get the well known **Sherman–Morrison formula**

$$(A - \sigma bc^T)^{-1} = A^{-1} + \alpha A^{-1}bc^T A^{-1}, \quad \alpha = 1/(\sigma^{-1} - c^T A^{-1}b). \tag{7.1.26}$$

It follows that $(A - \sigma bc^T)$ is nonsingular if and only if $\sigma \neq 1/c^T A^{-1}b$. This formula can be used to cheaply compute the new inverse when a matrix A is modified by a matrix of rank one. Such formulas are called updating formulas and are widely used in many contexts.

Frequently it is required to solve a linear problem, where the matrix has been modified by a correction of low rank. Consider first a linear system $Ax = b$, where $A \in \mathbf{R}^{n\times n}$ is modified by a correction of rank one,

$$(A - \sigma uv^T)\hat{x} = b. \tag{7.1.27}$$

Using the Sherman–Morrison formula the solution can be written

$$(A - \sigma uv^T)^{-1}b = A^{-1}b + \alpha A^{-1}u(v^T A^{-1}b), \quad \alpha = 1/(\sigma^{-1} - v^T A^{-1}u), \tag{7.1.28}$$

Here $x = A^{-1}b$ is the solution to the original system and $v^T A^{-1}b = v^T x$ is a scalar. Hence

$$\hat{x} = x + \beta w, \qquad \beta = v^T x/(\sigma^{-1} - v^T w), \quad w = A^{-1}u, \tag{7.1.29}$$

which shows that the solution $\hat{x}$ can be obtained from x by solving the system $Aw = u$. Note that *computing A^{-1} can be avoided.*

More generally, consider a linear system where the matrix has been modified with a matrix of rank $p > 1$,

$$(A + U\Sigma V^T)\hat{x} = b, \quad U, V \in \mathbf{R}^{n\times p}, \tag{7.1.30}$$

with $\Sigma \in \mathbf{R}^{p\times p}$ nonsingular. Using now the Woodbury formula, we can write

$$(A - U\Sigma V^T)^{-1}b = x + A^{-1}U(\Sigma^{-1} - V^T A^{-1}U)^{-1}V^T x. \tag{7.1.31}$$

This formula first requires the solution of the linear systems $AW = U$ with p right hand sides. The correction is then obtained by solving the linear system of size $p \times p$

$$(\Sigma^{-1} - V^T W)z = V^T x,$$

and forming Wz. If $p \ll n$ and the solution $x = A^{-1}b$ has been computed by a direct method it is this scheme is very efficient.

We end this with a note of caution that the updating methods given here can not be expected to be numerically stable in all cases. In particular, problems will arise when the initial problem is more illconditioned than the modified one.

7.1.6 The Singular Value Decomposition

The **singular value decomposition** (SVD) of a matrix $A \in \mathbf{R}^{m\times n}$ is of great theoretical and practical importance. Although its history goes back more than a century its use in numerical computations is much more recent.

Theorem 7.1.8. (Singular Value Decomposition.)
Every matrix $A \in \mathbf{R}^{m\times n}$ of rank r can be written

$$A = U\Sigma V^T, \qquad \Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \in \mathbf{R}^{m\times n}, \tag{7.1.32}$$

where $U \in \mathbf{R}^{m\times m}$ and $V \in \mathbf{R}^{n\times n}$ are orthogonal, $\Sigma_1 = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, and

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0.$$

(Note that if $r = n$ and/or $r = m$, some of the zero submatrices in Σ disappear.) The σ_i are called the **singular values** *of A and if we write*

$$U = (u_1, \ldots, u_m), \qquad V = (v_1, \ldots, v_n),$$

the u_i, $i = 1, \ldots, m$, and v_i, $i = 1, \ldots, n$, are left and right **singular vectors** *, respectively.*

Proof. Let $f(x) = \|Ax\|_2 = (x^T A^T Ax)^{1/2}$, the Euclidian length of the vector $y = Ax$, and consider the problem

$$\sigma_1 := \max_x \{f(x) \mid x \in \mathbf{R}^n,\ \|x\|_2 \leq 1\}.$$

Here $f(x)$ is a convex function[7] defined on a convex, compact set. It is well known (see, e.g., Ciarlet [13, Sec. 7.4]) that the maximum σ_1 is then attained on an extreme point of the set. Let v_1 be such a point with $\sigma_1 = \|Av_1\|$, $\|v_1\|_2 = 1$. If $\sigma_1 = 0$ then $A = 0$, and (7.1.32) holds with $\Sigma = 0$, and U and V arbitrary orthogonal matrices. Therefore, assume that $\sigma_1 > 0$, and set $u_1 = (1/\sigma_1)Av_1 \in \mathbf{R}^m$, $\|u_1\|_2 = 1$. Let the matrices

$$V = (v_1, V_1) \in \mathbf{R}^{n\times n}, \qquad U = (u_1, U_1) \in \mathbf{R}^{m\times m}$$

be orthogonal. (Recall that it is always possible to extend an orthogonal set of vectors to an orthonormal basis for the whole space.) Since $U_1^T Av_1 = \sigma_1 U_1^T u_1 = 0$ it follows that $U^T AV$ has the following structure:

$$A_1 \equiv U^T AV = \begin{pmatrix} \sigma_1 & w^T \\ 0 & B \end{pmatrix},$$

where $w^T = u_1^T AV_1$ and $B = U_1^T AV_1 \in \mathbf{R}^{(m-1)\times(n-1)}$.

$$\left\| A_1 \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \sigma_1^2 + w^T w \\ Bw \end{pmatrix} \right\|_2 \geq \sigma_1^2 + w^T w.$$

We have $UA_1 y = AVy = Ax$ and, since U and V are orthogonal, it follows that

$$\sigma_1 = \max_{\|x\|_2=1} \|Ax\|_2 = \max_{\|y\|_2=1} \|A_1 y\|_2,$$

[7] A function $f(x)$ is convex on a convex set S if for any x_1 and x_2 in S and any λ with $0 < \lambda < 1$, we have $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$.

and hence,

$$\sigma_1(\sigma_1^2 + w^T w)^{1/2} \geq \left\| A_1 \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 .$$

Combining these two inequalities gives $\sigma_1 \geq (\sigma_1^2 + w^T w)^{1/2}$, and it follows that $w = 0$. The proof can now be completed by an induction argument on the smallest dimension $\min(m, n)$. □

The geometrical significance of this theorem is as follows. The rectangular matrix A represents a mapping from $\mathbf{R}^n$ to $\mathbf{R}^m$. The theorem shows that there is an orthogonal basis in each of these two spaces, with respect to which this mapping is represented by a generalized diagonal matrix Σ. We remark that a singular value decomposition $A = U\Sigma V^H$, with U and V unitary, and Σ real diagonal, holds for any **complex** matrix $A \in \mathbf{C}^{m\times n}$.

The singular values of A are uniquely determined. The singular vector v_j, $j \leq r$, is unique (up to a factor ± 1) if σ_j^2 is a *simple* eigenvalue of $A^T A$. For multiple singular values, the corresponding singular vectors can be chosen as any orthonormal basis for the unique subspace that they span. Once the singular vectors v_j, $1 \leq j \leq r$ have been chosen, the vectors u_j, $1 \leq j \leq r$ are uniquely determined, and vice versa, using

$$Av_j = \sigma_j u_j, \qquad A^T u_j = \sigma_j v_j, \qquad j = 1, \ldots, r. \tag{7.1.33}$$

If U and V are partitioned according to

$$U = (U_1,\ U_2), \quad U_1 \in \mathbf{R}^{m\times r}, \quad V = (V_1,\ V_2), \quad V_1 \in \mathbf{R}^{n\times r}. \tag{7.1.34}$$

then the SVD can be written in the compact form

$$A = U_1 \Sigma_1 V_1^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T. \tag{7.1.35}$$

The last expression expresses A as a sum of matrices of rank one.

From (6.2.20) it follows that

$$A^T A = V \Sigma^T \Sigma V^T, \qquad AA^T = U \Sigma \Sigma^T U^T.$$

Thus σ_j^2, $j = 1, \ldots, r$ are the nonzero eigenvalues of the symmetric and positive semidefinite matrices $A^T A$ and AA^T, and v_j and u_j are the corresponding eigenvectors. Hence in principle the SVD can be reduced to the eigenvalue problem for symmetric matrices. For a proof of the SVD using this relationship see Stewart [1973, p. 319]. *However, this does not lead to a numerically stable way to compute the SVD*, since the singular values are square roots of the eigenvalues.

Definition 7.1.9.

The **range** *of the matrix $A \in \mathbf{R}m \times n$, denoted by $\mathcal{R}(A)$, is the subspace of $\mathbf{R}^m$ of dimension $r = \operatorname{rank}(A)$*

$$\mathcal{R}(A) = \{y \in \mathbf{R}^m |\ y = Ax,\ x \in \mathbf{R}^n\}. \tag{7.1.36}$$

The **null space** $\mathcal{N}(A)$ *of* A *is a subspace of* $\mathbf{R}^n$ *of dimension* $n - r$*:*

$$\mathcal{N}(A) = \{x \in \mathbf{R}^n | \ Ax = 0\}. \tag{7.1.37}$$

The SVD gives complete information about the four fundamental subspaces associated with A and A^T. It is easy to verify that the range of A and nullspace of A^T are given by

$$\mathcal{R}(A) = \mathcal{R}(U_1) \qquad \mathcal{N}(A^T) = \mathcal{R}(U_2) \tag{7.1.38}$$

Since $A^T = V\Sigma^T U^T$ it follows that also

$$\mathcal{R}(A^T) = \mathcal{R}(V_1) \qquad \mathcal{N}(A) = \mathcal{R}(V_2). \tag{7.1.39}$$

We immediately find the well-known relations

$$\mathcal{R}(A)^\perp = \mathcal{N}(A^T), \qquad \mathcal{N}(A)^\perp = \mathcal{R}(A^T),$$

7.1.7 Norms of Vectors and Matrices

In perturbation theory as well as in the analysis of errors in matrix computation it is useful to have a measure of the size of a vector or a matrix. Such measures are provided by vector and matrix norms, which can be regarded as generalizations of the absolute value function on $\mathbf{R}$.

Definition 7.1.10.

A **norm** *on a vector space* $\mathbf{V} \in \mathbf{C}^n$ *is a function* $\mathbf{V} \to \mathbf{R}$ *denoted by* $\|\cdot\|$ *that satisfies the following three conditions:*

1. $\|x\| > 0, \quad \forall x \in \mathbf{V}, \quad x \neq 0$ *(definiteness)*
2. $\|\alpha x\| = |\alpha| \, \|x\|, \quad \forall \alpha \in \mathbf{C}, \quad x \in \mathbf{C}^n$ *(homogeneity)*
3. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbf{V}$ *(triangle inequality)*

The triangle inequality is often used in the form (see Problem 11)

$$\|x \pm y\| \geq \big| \, \|x\| - \|y\| \, \big|.$$

The most common vector norms are special cases of the family of **Hölder** norms or p-norms

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}, \qquad 1 \leq p < \infty. \tag{7.1.40}$$

The three most important particular cases are $p = 1, 2$ and the limit when $p \to \infty$:

$$\begin{aligned} \|x\|_1 &= |x_1| + \cdots + |x_n|, \\ \|x\|_2 &= (|x_1|^2 + \cdots + |x_n|^2)^{1/2} = (x^H x)^{1/2}, \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i|. \end{aligned} \tag{7.1.41}$$

A vector norm$\|\cdot\|$ is called **absolute** if $\|x\| = \|\,|x|\,\|$, and **monotone** if $|x| \le |y| \Rightarrow \|x\| \le \|y\|$. It can be shown that a vector norm is monotone if and only if it is absolute; see Stewart and Sun [61, Theorem II.1.3]. Clearly the vector p-norms are absolute for all $1 \le p < \infty$.

The vector 2-norm is also called the Euclidean norm. It is invariant under unitary (orthogonal) transformations since

$$\|Qx\|_2^2 = x^H Q^H Q x = x^H x = \|x\|_2^2$$

if Q is orthogonal.

The proof that the triangle inequality is satisfied for the p-norms depends on the following inequality. Let $p > 1$ and q satisfy $1/p + 1/q = 1$. Then it holds that

$$\alpha\beta \le \frac{\alpha^p}{p} + \frac{\beta^p}{q}.$$

Indeed, let x and y be any real number and λ satisfy $0 < \lambda < 1$. Then by the convexity of the exponential function it holds that

$$e^{\lambda x + (1-\lambda)y} \le \lambda e^x + (1-\lambda)e^y.$$

We obqatin the desired result by setting $\lambda = 1/p$, $x = p\log\alpha$ and $x = q\log\beta$.

Another important property of the p-norms is the **Hölder inequality**

$$|x^H y| \le \|x\|_p \|y\|_q, \quad \frac{1}{p} + \frac{1}{q} = 1, \quad p \ge 1. \tag{7.1.42}$$

For $p = q = 2$ this becomes the well known **Cauchy–Schwarz inequality**

$$|x^H y| \le \|x\|_2 \|y\|_2.$$

Another special case is $p = 1$ for which we have

$$|x^H y| = \left| \sum_{i=1}^n x_i^H y_i \right| \le \sum_{i=1}^n |x_i^H y_i| \le \max_i |y_i| \sum_{i=1}^n |x_i| = \|x\|_1 \|y\|_\infty. \tag{7.1.43}$$

Definition 7.1.11.

For any given vector norm $\|\cdot\|$ on $\mathbf{C}^n$ the **dual norm** *$\|\cdot\|_D$ is defined by*

$$\|x\|_D = \max_{y \ne 0} |x^H y| / \|y\|. \tag{7.1.44}$$

The vectors in the set

$$\{y \in \mathbf{C}^n \mid \|y\|_D \|x\| = y^H x = 1\}. \tag{7.1.45}$$

are said to be **dual vectors** *to x with respect to $\|\cdot\|$.*

It can be shown that the dual of the dual norm is the original norm (see [61, Theorem II.1.12]). It follows from the Hölder inequality that the dual of the p-norm is the q-norm, where

$$1/p + 1/q = 1.$$

The dual of the 2-norm can be seen to be itself. It can be shown to be the only norm with this property (see [42, Theorem 5.4.16]).

The vector 2-norm can be generalized by taking

$$\|x\|_{2,G}^2 = (x, x) = x^H G x, \tag{7.1.46}$$

where G is a Hermitian positive definite matrix. It can be shown that the unit ball $\{x : \|x\| \le 1\}$ corresponding to this norm is an ellipsoid, and hence they are also called **elliptic norms**. Other generalized norms are the **scaled** p-norms defined by

$$\|x\|_{p,D} = \|Dx\|_p, \quad D = \operatorname{diag}(d_1, \ldots, d_n), \quad d_i \neq 0, \quad i = 1 : n. \tag{7.1.47}$$

All norms on $\mathbf{C}^n$ are equivalent in the following sense: For each pair of norms $\|\cdot\|$ and $\|\cdot\|'$ there are positive constants c and c' such that

$$\frac{1}{c}\|x\|' \le \|x\| \le c'\|x\|', \quad \forall x \in \mathbf{C}^n. \tag{7.1.48}$$

In particular, it can be shown that for the p-norms we have

$$\|x\|_q \le \|x\|_p \le n^{(1/p-1/q)}\|x\|_q, \quad 1 \le p \le q \le \infty. \tag{7.1.49}$$

For example, setting $p = 2$ and $q = \infty$ we obtain

$$\|x\|_\infty \le \|x\|_2 \le \sqrt{n}\|x\|_\infty,$$

We now consider **matrix norms**. Given any vector norm, we can construct a matrix norm by defining

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|. \tag{7.1.50}$$

This norm is called the **operator norm**, or the matrix norm **subordinate** to the vector norm. From the definition it follows directly that

$$\|Ax\| \le \|A\|\ \|x\|, \quad \forall x \in \mathbf{C}^n. \tag{7.1.51}$$

Whenever this inequality holds, we say that the matrix norm is **consistent** with the vector norm.

It is an easy exercise to show that operator norms are **submultiplicative**, i.e., whenever the product AB is defined it satisfies the condition

4. $N(AB) \le N(A)N(B)$

Explicit expressions for the matrix norms subordinate to the vector p-norms are known only for $p = 1, 2, \infty$:

Theorem 7.1.12.
For $p = 1, 2\ \infty$ the matrix subordinate p-norm are given by

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|, \tag{7.1.52}$$

$$\|A\|_2 = \max_{\|x\|=1} (x^H A^H A x)^{1/2} = \sigma_1(A), \tag{7.1.53}$$

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|. \tag{7.1.54}$$

Proof. To prove the result for $p = 1$ we partition $A = (a_1, \dots, a_n)$ by columns For any $x = (x_1, \dots, x_n)^T \neq 0$ we have

$$\|Ax\|_1 = \left\| \sum_{j=1}^{n} x_j a_j \right\|_1 \le \sum_{j=1}^{n} |x_j| \|a_j\|_1 \le \max_{1\le j\le n} \|a_j\|_1 \|x\|_1.$$

It follows that $\|Ax\|_1 \le \max_{1\le j\le n} \|a_j\|_1 = \|a_k\|_1$, for some $1 \le k \le n$. But then

$$\|Ae_k\|_1 = \|a_k\|_1 = \max_{1\le j\le n} \|a_j\|_1,$$

and hence $\|A\|_1 \ge \max_{1\le j\le n} \|a_j\|_1$. This implies (7.1.52). The formula (7.1.54) for the matrix ∞-norm is proved in a similar fashion. The expression for the 2-norm follows from the extremal property

$$\max_{\|x\|=1} \|Ax\|_2 = \max_{\|x\|=1} \|U\Sigma V^T x\|_2 = \sigma_1(A)$$

of the singular vector $x = v_1$. □

For $p = 1$ and $p = \infty$ the matrix subordinate norms are easily computable. Note that the 1-norm is the maximal column sum and the ∞-norm is the maximal row sum of the magnitude of the elements. It follows that $\|A\|_1 = \|A^H\|_\infty$.

The 2-norm, also called the **spectral norm**, equals the largest singular value $\sigma_1(A)$ of A. It has the drawback that it is expensive to compute. but is a useful analytical tool. Since the nonzero eigenvalues of $A^H A$ and AA^H are the same it follows that $\|A\|_2 = \|A^H\|_2$. An upper bound for the matrix 2-norm is

$$\|A\|_2 \le (\|A\|_1 \|A\|_\infty)^{1/2}. \tag{7.1.55}$$

The proof of this bound is given as an exercise in Problem 16.

Another way to proceed in defining norms for matrices is to regard $\mathbf{C}^{m\times n}$ as an mn-dimensional vector space and apply a vector norm over that space.

Definition 7.1.13.
The **Frobenius norm**[8] *is derived from the vector* 2*-norm*

$$\|A\|_F = \Big(\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2\Big)^{1/2} \tag{7.1.56}$$

The Frobenius norm is submultiplicative, but is often larger than necessary, e.g., $\|I_n\|_F = n^{1/2}$. This tends to make bounds derived in terms of the Frobenius norm not as sharp as they might be. From (7.1.58) it follows that

$$\frac{1}{\sqrt{n}}\|A\|_F \le \|A\|_2 \le \|A\|_F, \quad k = \min(m,n). \tag{7.1.57}$$

Note that $\|A^H\|_F = \|A\|_F$. Useful alternative characterizations of the Frobenius norm are

$$\|A\|_F^2 = \operatorname{trace}(A^H A) = \sum_{i=1}^{k}\sigma_i^2(A), \quad k = \min(m,n), \tag{7.1.58}$$

where $\sigma_i(A)$ are the nonzero singular values of A. Of the matrix norms the 1-,∞- and the Frobenius norm are absolute, but for the 2-norm the best result is

$$\| \, |A| \, \|_2 \le n^{1/2}\|A\|_2.$$

Table 7.1.1. *Numbers* γ_{pq} *such that* $\|A\|_p \le \gamma_{pq}\|A\|_q$*, where* $A \in \mathbf{C}^{m\times n}$ *and* $\operatorname{rank}(A) = r$*.*

$p\backslash q$	1	2	∞	F
1	1	$\sqrt{m}$	m	$\sqrt{m}$
2	$\sqrt{n}$	1	$\sqrt{m}$	$\sqrt{mn}$
∞	n	$\sqrt{n}$	1	$\sqrt{n}$
F	$\sqrt{n}$	$\sqrt{r}$	$\sqrt{m}$	1

The Frobenius norm shares with the 2-norm the property of being invariant with respect to unitary (orthogonal) transformations

$$\|QAP\| = \|A\|. \tag{7.1.59}$$

Such norms are called **unitarily invariant** and have an interesting history; see Stewart and Sun [61, Sec. II.3].

[8] Ferdinand George Frobenius (1849–1917) German mathematician, professor at ETH Zürich (1875–1892) before he succeeded Weierstrass at Berlin University.

Theorem 7.1.14.

Let $\|\cdot\|$ be a unitarily invariant norm. Then $\|A\|$ is a symmetric function of the singular values

$$\|A\| = \Phi(\sigma_1, \ldots, \sigma_n).$$

Proof. Let the singular value decomposition of A be $A = U\Sigma V^T$. Then the invariance implies that $\|A\| = \|\Sigma\|$, which shows that $\Phi(A)$ only depends on Σ. Since the ordering of the singular values in Σ is arbitrary Φ must be symmetric in σ_i, $i = 1 : n$. □

Unitarily invariant norms were characterized by John von Neumann [52], who showed that the converse of Theorem 7.1.14 is true: A function $\Phi(\sigma_1, \ldots, \sigma_n)$ which is symmetric in its arguments and satisfies the three properties in the Definition 7.1.10 of a vector norm defines a unitarily invariant matrix norm. In this connection such functions are called **symmetric gauge functions**. Two examples are

$$\|A\|_2 = \max_i \sigma_i, \qquad \|A\|_F = \left(\sum_{i=1}^{n} \sigma_i^2\right)^{1/2}.$$

One use of norms is the study of *limits of sequences of vectors and matrices* (see Sec. 9.2.4). Consider an infinite sequence $x_1, x_2, \ldots$ of elements of a vector space $\mathbf{V}$ and let $\|\cdot\|$ be a norm on $\mathbf{V}$. The sequence is said to converge (strongly if V is infinite dimensional) to a limit $x \in \mathbf{V}$, and we write $\lim_{k\to\infty} x_k = x$ if

$$\lim_{k\to\infty} \|x_k - x\| = 0,$$

For a finite dimensional vector space it follows from the equivalence of norms that convergence is independent of the choice of norm. The particular choice $\|\cdot\|_\infty$ shows that convergence of vectors in $\mathbf{C}^n$ is equivalent to convergence of the n sequences of scalars formed by the components of the vectors. By considering matrices in $\mathbf{C}^{m\times n}$ as vectors in $\mathbf{C}^{mn}$ the same conclusion holds for matrices.

7.1.8 Conditioning of Linear Systems

Consider a linear system $Ax = b$ where A is nonsingular and $b \neq 0$. The sensitivity of the solution x and the inverse A^{-1} to perturbations in A and b is of practical importance, since the matrix A and vector b are rarely known exactly. They may be subject to observational errors, or given by formulas which involve roundoff errors in their evaluation. (Even if they were known exactly, they may not be represented exactly as floating-point numbers in the computer.)

We start with deriving some results that are needed in the analysis.

Lemma 7.1.15.

Let $E \in \mathbf{R}^{n\times n}$ be a matrix for which $\|E\| < 1$. Then the matrix $(I - E)$ is nonsingular and for its inverse we have the estimate

$$\|(I - E)^{-1}\| \leq 1/(1 - \|E\|). \tag{7.1.60}$$

Proof. If $(I - E)$ is singular there exists a vector $x \neq 0$ such that $(I - E)x = 0$. Then $x = Ex$ and $\|x\| = \|Ex\| \leq \|E\| \, \|x\| < \|x\|$, which is a contradiction since $\|x\| \neq 0$. Hence $(I - E)$ is nonsingular.

Next consider the identity $(I - E)(I - E)^{-1} = I$ or

$$(I - E)^{-1} = I + E(I - E)^{-1}.$$

Taking norms we get

$$\|(I - E)^{-1}\| \leq 1 + \|E\| \, \|(I - E)^{-1}\|,$$

and (7.1.60) follows. (For another proof, see hint in Problem 7.2.19.) □

Corollary 7.1.16.

Assume that $\|B - A\|\|B^{-1}\| = \eta < 1$. *Then it holds that*

$$\|A^{-1}\| \leq \frac{1}{1-\eta}\|B^{-1}\|, \qquad \|A^{-1} - B^{-1}\| \leq \frac{\eta}{1-\eta}\|B^{-1}\|.$$

Proof. We have $\|A^{-1}\| = \|A^{-1}BB^{-1}\| \leq \|A^{-1}B\| \, \|B^{-1}\|$. The first inequality then follows by taking $E = B^{-1}(B - A) = I - B^{-1}A$ in Lemma 7.1.15. From the identity

$$A^{-1} - B^{-1} = A^{-1}(B - A)B^{-1} \tag{7.1.61}$$

we have $\|A^{-1} - B^{-1}\| \leq \|A^{-1}\| \, \|B - A\| \, \|B^{-1}\|$. The second inequality now follows from the first. □

Let x be the solution x to a system of linear equations $Ax = b$, and let $x + \delta x$ satisfy the perturbed system

$$(A + \delta A)(x + \delta x) = b + \delta b,$$

where δA and δb are perturbations in A and b. Subtracting out $Ax = b$ we get

$$(A + \delta A)\delta x = -\delta A x + \delta b.$$

Assuming that A and $A + \delta A$ are nonsingular, we can multiply by A^{-1} and solve for δx. This yields

$$\delta x = (I + A^{-1}\delta A)^{-1}A^{-1}(-\delta A x + \delta b), \tag{7.1.62}$$

which is the basic identity for the perturbation analysis.

In the simple case that $\delta A = 0$, we have $\delta x = A^{-1}\delta b$, which implies that $|\delta x| = |A^{-1}| \, |\delta b|$, Taking norms

$$\|\delta x\| \leq \|A^{-1}\| \, \|\delta b\|.$$

Usually it is more appropriate to consider *relative* perturbations,

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A,x)\frac{\|\delta b\|}{\|b\|}, \quad \kappa(A,x) := \frac{\|Ax\|}{\|x\|}\|A^{-1}\|. \tag{7.1.63}$$

Here $\kappa(A,x)$ is the condition number with respect to perturbations in b. It is important to note that this implies that the size of the residual vector $r = b - A\bar{x}$ gives no direct indication of the *error* in an approximate solution $\bar{x}$. For this we need information about A^{-1} or the condition number $\kappa(A,x)$.

The inequality (7.1.63) is sharp in the sense that for any matrix norm and for any A and b there exists a perturbation δb, such that equality holds. From $\|b\| = \|Ax\| \leq \|A\|\,\|x\|$ it follows that

$$\kappa(A,x) \leq \|A\|\,\|A^{-1}\|, \tag{7.1.64}$$

but here *equality will hold only for rather special right hand sides* b. Equation (7.1.64) motivates the following definition:

Definition 7.1.17. *For a square nonsingular matrix* A *the* **condition number** *is*

$$\kappa = \kappa(A) = \|A\|\,\|A^{-1}\|. \tag{7.1.65}$$

where $\|\cdot\|$ *denotes any matrix norm.*

Clearly $\kappa(A)$ depends on the chosen matrix norm. If we want to indicate that a particular norm is used, then we write, e.g., $\kappa_\infty(A)$ etc. For the 2-norm we have using the SVD that $\|A\|_2 = \sigma_1$ and $\|A^{-1}\| = 1/\sigma_n$, where σ_1 and σ_n are the largest and smallest singular values of A. Hence

$$\kappa_2(A) = \sigma_1/\sigma_n. \tag{7.1.66}$$

Note that $\kappa(\alpha A) = \kappa(A)$, i.e., the condition number is invariant under multiplication of A by a scalar. From the definition it also follows easily that

$$\kappa(AB) \leq \kappa(A)\kappa(B).$$

Further, for all p-norms it follows from the identity $AA^{-1} = I$ that

$$\kappa_p(A) = \|A\|_p\|A^{-1}\|_p \geq \|I\|_p = 1,$$

that is, the condition number is always greater or equal to one.

We now show that $\kappa(A)$ also is the condition number with respect to perturbations in A.

Theorem 7.1.18.

Consider the linear system $Ax = b$*, where the matrix* $A \in \mathbf{R}^{n\times n}$ *is nonsingular. Let* $(A + \delta A)(x + \delta x) = b + \delta b$ *be a perturbed system and assume that*

$$\eta = \|A^{-1}\|\,\|\delta A\| < 1.$$

Then $(A + \delta A)$ is nonsingular and the norm of the perturbation δx is bounded by

$$\|\delta x\| \le \frac{\|A^{-1}\|}{1-\eta}\left(\|\delta A\|\,\|x\| + \|\delta b\|\right). \tag{7.1.67}$$

Proof. Taking norms in equation (7.1.62) gives

$$\|\delta x\| \le \|(I + A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\big(\|\delta A\|\,\|x\| + \|\delta b\|\big).$$

By assumption $\|A^{-1}\delta A\| \le \|A^{-1}\|\,\|\delta A\| = \eta < 1$. Using Lemma 7.1.15 it follows that $(I + A^{-1}\delta A)$ is nonsingular and

$$\|(I + A^{-1}\delta A)^{-1}\| \le 1/(1-\eta),$$

which proves the result. □

In most practical situations it holds that $\eta \ll 1$ and therefore $1/(1-\eta] \approx 1$. Therefore, if upper bounds

$$\|\delta A\| \le \epsilon_A\|A\|, \qquad \|\delta b\| \le \epsilon_b\|b\|, \tag{7.1.68}$$

for $\|\delta A\|$ and $\|\delta b\|$ are known, then for the normwise relative perturbation it holds that

$$\frac{\|\delta x\|}{\|x\|} \lessapprox \kappa(A)\left(\epsilon_A + \epsilon_b\frac{\|b\|}{\|A\|\|x\|}\right).$$

Substituting $b = I$, $\delta b = 0$ and $x = A^{-1}$ in (7.1.67) and proceeding similarly from $(A + \delta A)(X + \delta X) = I$, we obtain the perturbation bound for $X = A^{-1}$

$$\frac{\|\delta X\|}{\|X\|} \le \frac{\kappa(A)}{1-\eta}\frac{\|\delta A\|}{\|A\|}. \tag{7.1.69}$$

This shows that $\kappa(A)$ is indeed the condition number of A with respect to inversion.

The relative distance of a matrix A to the set of singular matrices in some norm is defined as

$$\operatorname{dist}(A) := \min\left\{\frac{\|\delta A\|}{\|A\|} \mid (A + \delta A) \text{ singular}\right\}. \tag{7.1.70}$$

The following theorem shows that the reciprocal of the condition number $\kappa(A)$ can be interpreted as a measure of the nearness to singularity of a matrix A.

Theorem 7.1.19 (Kahan [47]).
Let $A \in \mathbf{C}^{n\times n}$ be a nonsingular matrix and $\kappa(A) = \|A\|\|A^{-1}\|$ the condition number with respect to a norm $\|\cdot\|$ subordinate to some vector norm. Then

$$\operatorname{dist}(A) = \kappa^{-1}(A). \tag{7.1.71}$$

Proof. If $(A+\delta A)$ is singular, then there is a vector $x \neq 0$ such that $(A+\delta A)x = 0$. Then, setting $y = Ax$, it follows that

$$\|\delta A\| \geq \frac{\|\delta A\, x\|}{\|x\|} = \frac{\|Ax\|}{\|x\|} = \frac{\|y\|}{\|A^{-1}y\|} \geq \frac{1}{\|A^{-1}\|} = \frac{\|A\|}{\kappa(A)},$$

or $\|\delta A\|/\|A\| \geq 1/\kappa(A)$.

Now let x be a vector with $\|x\| = 1$ such that $\|A^{-1}x\| = \|A^{-1}\|$. Set $y = A^{-1}x/\|A^{-1}\|$ so that $\|y\| = 1$ and $Ay = x/\|A^{-1}\|$. Let z be a dual vector to y so that (see Definition 7.1.11) $\|z\|_D\|y\| = z^H y = 1$, where $\|\cdot\|_D$ is the dual norm. Then $\|z\|_D = 1$, and if we take

$$\delta A = -xz^H/\|A^{-1}\|,$$

it follows that

$$(A + \delta A)y = Ay - xz^H y/\|A^{-1}\| = (x - x)/\|A^{-1}\| = 0.$$

Hence $(A + \delta A)$ is singular. Further

$$\|\delta A\|\|A^{-1}\| = \|xz^H\| = \max_{\|v\|=1} \|(xz^H)v\| = \|x\| \max_{\|v\|=1} |z^H v| = \|z\|_D = 1,$$

and thus $\|\delta A\| = 1/\|A^{-1}\|$, which proves the theorem. □

The result in Theorem 7.1.19 can be used to get a *lower bound* for the condition number $\kappa(A)$, see, Problem 21. For the 2-norm the result follows directly from the SVD $A = U\Sigma V^H$. The closest singular matrix then equals $A + \delta A$, where

$$\delta A = -\sigma_n u_n v_n^H, \quad \|\delta A\|_2 = \sigma_n = 1/\|A^{-1}\|_2. \tag{7.1.72}$$

Matrices with small condition numbers are said to be **well-conditioned**. For any real, orthogonal matrix Q we have $\kappa_2(Q) = \|Q\|_2\|Q^{-1}\|_2 = 1$, so Q is perfectly conditioned in the 2-norm. Furthermore, for any orthogonal P and Q, we have $\kappa_2(PAQ) = \kappa_2(A)$, i.e., $\kappa_2(A)$ is invariant under orthogonal transformations.

When a linear system is ill-conditioned, i.e. $\kappa(A) \gg 1$, roundoff errors will in general cause a computed solution to have a large error. It is often possible to show that a small **backward error** in the following sense:

Definition 7.1.20.

An algorithm for solving a linear system $Ax = b$ is said to be (normwise) **backward stable** *if, for any data $A \in \mathbf{R}^{n\times n}$ and $b \in \mathbf{R}^n$, there exist perturbation matrices and vectors δA and δb, such that the solution $\bar{x}$ computed by the algorithm is the* exact *solution to a neighbouring system*

$$(A + \delta A)\bar{x} = (b + \delta b), \tag{7.1.73}$$

where

$$\|\delta A\| \leq c_1(n)u\|A\|, \qquad \|\delta b\| \leq c_2(n)u\|b\|.$$

A computed solution $\bar{x}$ is called a (normwise) stable solution if it satisfies (7.1.73).

Since the data A and b usually are subject to errors and not exact, it is reasonable to be satisfied with the computed solution $\bar{x}$ if the backward errors δA and δb are small in comparison to the uncertainties in A and b. As seen from (7.1.64), this does not mean that $\bar{x}$ is close to the exact solution x.

Review Questions

1. Define the concepts:
(i) Real symmetric matrix. (ii) Real orthogonal matrix.
(iii) Real skew-symmetric matrix. (iv) Triangular matrix.
(v) Hessenberg matrix.

2. (a) Given a vector norm define the matrix subordinate norm.
(b) Give explicit expressions for the matrix p norms for $p = 1, 2, \infty$.

3. Define the p norm of a vector x. Show that

$$\frac{1}{n}\|x\|_1 \leq \frac{1}{\sqrt{n}}\|x\|_2 \leq \|x\|_\infty,$$

which are special cases of (7.1.49).

4. Verify the formulas (7.1.21) for the inverse of a 2×2 block triangular matrices.

Problems

1. Show that if $R \in \mathbf{R}^{n\times n}$ is strictly upper triangular, then $R^n = 0$.

2. If A and B are square upper triangular matrices show that AB is upper triangular, and that A^{-1} is upper triangular if it exists. Is the same true for lower triangular matrices?

3. To solve a linear system $Ax = b$, where $A \in \mathbf{R}^n$, by Cramer's rule (see Equation (7.1.10) requires the evaluation of $n + 1$ determinants of order n. Estimate the number of multiplications needed for $n = 50$ if the determinants are evaluated in the naive way. Estimate the time it will take on a computer performing 10^9 floating point operations per second!

4. Show that if the complex matrix $U = Q_1 + iQ_2$ is unitary, then the real matrix

$$\tilde{U} = \begin{pmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{pmatrix}$$

is orthogonal.

5. Let $A \in \mathbf{R}^{n\times n}$ be a given matrix. Show that if $Ax = y$ has *at least one* solution for any $y \in \mathbf{R}^n$, then it has *exactly one* solution for any $y \in \mathbf{R}^n$. (This is a useful formulation for showing uniqueness of approximation formulas.)

6. Let $A \in \mathbf{R}^{m\times n}$ have rows a_i^T, i.e., $A^T = (a_1, \ldots, a_m)$. Show that

$$A^T A = \sum_{i=1}^{m} a_i a_i^T.$$

What is the corresponding expression for $A^T A$ if A is instead partitioned into columns?

7. Let $S \in \mathbf{R}^{n\times n}$ be skew-Hermitian, i.e. $S^H = -S$.

(a) Show that $I - S$ is nonsingular.

(b) Show that the matrix $Q = (I-S)^{-1}(I+S)$, known as the **Cayley transform**[9] is unitary, i.e., $Q^H Q = I$.

8. Show that for $x \in \mathbf{R}^n$,

$$\lim_{p\to\infty} \|x\|_p = \max_{1\le i\le n} |x_i|.$$

9. Prove that the following inequalities are valid and best possible:

$$\|x\|_2 \le \|x\|_1 \le n^{1/2}\|x\|_2, \qquad \|x\|_\infty \le \|x\|_1 \le n\|x\|_\infty.$$

Derive similar inequalities for the comparison of the operator norms $\|A\|_1$, $\|A\|_2$, and $\|A\|_\infty$.

10. Show that any vector norm is uniformly continuous by proving the inequality

$$|\,\|x\| - \|y\|\,| \le \|x - y\|, \qquad x, y \in \mathbf{R}^n.$$

11. Show that for any matrix norm there exists a consistent vector norm.

Hint: Take $\|x\| = \|xy^T\|$ for any vector $y \in \mathbf{R}^n$, $y \neq 0$.

12. Derive the formula for $\|A\|_\infty$ given in Theorem 7.1.12.

13. Make a table corresponding to Table 7.1.1 for the vector norms $p = 1, 2, \infty$.

14. Prove that for any subordinate matrix norm

$$\|A + B\| \le \|A\| + \|B\|, \qquad \|AB\| \le \|A\|\|B\|.$$

15. Show that $\|A\|_2 = \|PAQ\|_2$ if P and Q are orthogonal matrices.

16. Use the result $\|A\|_2^2 = \rho(A^T A) \le \|A^T A\|$, valid for any matrix operator norm $\|\cdot\|$, where $\rho(A^T A)$ denotes the spectral radius of $A^T A$, to deduce the upper bound in (7.1.55).

17. Prove the expression (7.1.54) for the matrix norm subordinate to the vector ∞-norm.

18. (a) Let T be a nonsingular matrix, and let $\|\cdot\|$ be a given vector norm. Show that the function $N(x) = \|Tx\|$ is a vector norm.

(b) What is the matrix norm subordinate to $N(x)$?

(c) If $N(x) = \max_i |k_i x_i|$, what is the subordinate matrix norm?

[9] Arthur Cayley (1821–1895), English mathematician, is credited with developing the algebra of matrices. Although he worked as a lawyer for many years before he became Sadlerian professor at Cambridge in 1863, he has authored more than 900 papers.

19. Consider an upper block triangular matrix

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

and suppose that R_{11}^{-1} and R_{22}^{-1} exists. Show that R^{-1} exists.

20. Use the Woodbury formula to prove the identity

$$(I - AB)^{-1} = I + A(I - BA)^{-1}B.$$

21. (a) Let A^{-1} be known and let B be a matrix coinciding with A except in one row. Show that if B is nonsingular then B^{-1} can be computed by about $2n^2$ multiplications using the Sherman–Morrison formula (7.1.26).

(b) Use the Sherman–Morrison formula to compute B^{-1} if

$$A = \begin{pmatrix} 1 & 0 & -2 & 0 \\ -5 & 1 & 11 & -1 \\ 287 & -67 & -630 & 65 \\ -416 & 97 & 913 & -94 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} 13 & 14 & 6 & 4 \\ 8 & -1 & 13 & 9 \\ 6 & 7 & 3 & 2 \\ 9 & 5 & 16 & 11 \end{pmatrix},$$

and B equals A except that the element 913 has been changed to 913.01.

22. Use the result in Theorem 7.1.19 to obtain the lower bound $\kappa_\infty(A) \geq 3/(2|\epsilon|) = 1.5|\epsilon|^{-1}$ for the matrix

$$A = \begin{pmatrix} 1 & -1 & 1 \\ -1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon \end{pmatrix}, \qquad 0 < |\epsilon| < 1.$$

(The true value is $\kappa_\infty(A) = 1.5(1 + |\epsilon|^{-1})$.)

7.2 Elimination Methods

7.2.1 Triangular Matrices

An **upper triangular** matrix is a matrix U for which $u_{ij} = 0$ whenever $i > j$. A square upper triangular matrix has form

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & \dots & u_{nn} \end{pmatrix}.$$

If also $u_{ij} = 0$ when $i = j$ then U is **strictly** upper triangular. Similarly a matrix L is **lower triangular** if $l_{ij} = 0$, $i < j$, and strictly lower triangular if $l_{ij} = 0$, $i \leq j$. Clearly the transpose of an upper triangular matrix is lower triangular and vice versa.

Triangular matrices have several nice properties. It is easy to verify that sums, products and inverses of square upper (lower) triangular matrices are again triangular matrices of the same type. The diagonal elements of the product $U = U_1U_2$ of two triangular matrices are just the product of the diagonal elements in U_1 and U_2. From this it follows that the diagonal elements in U^{-1} are the inverse of the diagonal elements in U.

Triangular linear systems are easy to solve. In an upper triangular linear system $Ux = b$, the unknowns can be computed recursively from

$$x_n = b_n/u_{nn} \qquad x_i = \Big(b_i - \sum_{k=i+1}^{n} u_{ik}x_k\Big)\Big/u_{ii}, \quad i = n-1:1. \tag{7.2.1}$$

Since the unknowns are solved for in *backward* order, this is called **back-substitution**. Similarly, in a lower triangular linear system $Ly = c$, the unknowns can be computed by **forward-substitution**.

$$y_1 = c_1/u_{11} \qquad y_i = \Big(c_i - \sum_{k=1}^{i-1} l_{ik}y_k\Big)\Big/l_{ii}, \quad i = 2:n. \tag{7.2.2}$$

When implementing a matrix algorithm such as (7.2.1) or (7.2.2) on a computer, the *order of operations* in matrix algorithms may be important. One reason for this is the economizing of storage, since even matrices of moderate dimensions have a large number of elements. When the initial data is not needed for future use, computed quantities may overwrite data. To resolve such ambiguities in the description of matrix algorithms it is important to be able to describe computations in a more precise form. For this purpose we will use an informal programming language, which is sufficiently precise for our purpose but allows the suppression of cumbersome details. These concepts are illustrated for the back-substitution (7.2.1). in the following program where the solution vector x overwrites the data vector b.

Algorithm 7.2.1 Back-substitution.
Given an upper triangular matrix $U \in \mathbf{R}^{n\times n}$ and a vector $b \in \mathbf{R}^n$, the following algorithm computes $x \in \mathbf{R}^n$ such that $Ux = b$:

for $i = n:-1:1$
$\quad s := \sum_{k=i+1}^{n} u_{ik}b_k;$
$\quad b_i := (b_i - s)/u_{ii};$
end

Here := is the assignment symbol and $x := y$ means that the value of y is assigned to x. Note that in order to minimize round-off errors b_i is added *last* to the sum; compare the error bound (2.4.3).

Another possible sequencing of the operations is:

```
for k = n : −1 : 1
    b_k := b_k/u_kk;
    for i = k − 1 : −1 : 1
        b_i := b_i − u_ik b_k;
    end
end
```

Here the elements in U are accessed column-wise instead of row-wise as in the previous algorithm. Such differences can influence the efficiency when implementing matrix algorithms. For example, if U is stored column-wise as is the convention in Fortran, the second version is to be preferred.

We will often use the concept of a **flop**, to mean roughly the amount of work associated with the computation

$$s := s + a_{ik}b_{kj},$$

i.e., one floating point addition and multiplication and some related subscript computation.[10] With this notation solving a triangular system requires $\frac{1}{2}n^2$ flam.

7.2.2 Gaussian Elimination

Consider a linear system $Ax = b$, where the matrix $A \in \mathbf{R}^{m\times n}$, and vector $b \in \mathbf{R}^m$ are given and the vector $x \in \mathbf{R}^n$ is to be determined, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{7.2.3}$$

A fundamental observation is that the following elementary operation can be performed on the system without changing the set of solutions:

- Adding a multiple of the ith equation to the jth equation.
- Interchange two equations.

These correspond in an obvious way to row operations on the augmented matrix (A, b). It is also possible to interchange two columns in A provided we make the corresponding interchanges in the components of the solution vector x.

The idea behind **Gaussian elimination**[11] is to use such elementary operations to eliminate the unknowns in the system $Ax = b$ in a systematic way, so that

[10] Note that in older textbooks this was called a flop. However, in more recent books (e.g., Higham [41, 2002]) a flop is instead defined as a floating point add *or* multiply.

[11] Named after Carl Friedrich Gauss (1777–1855), but known already in China as early as the first century BC.

at the end an equivalent upper triangular system is produced, which is then solved by back-substitution. If $a_{11} \neq 0$, then in the first step we eliminate x_1 from the last $(n-1)$ equations by subtracting the multiple

$$l_{i1} = a_{i1}/a_{11}, \quad i = 2:n,$$

of the first equation from the ith equation. This produces a reduce system of $(n-1)$ equations in the $(n-1)$ unknowns $x_2 : x_n$, where the new coefficients are given by

$$a_{ij}^{(2)} = a_{ij} - l_{i1}a_{1j}, \qquad b_i^{(2)} = b_i - l_{i1}b_1, \quad i = 2:n.$$

If $a_{22}^{(2)} \neq 0$, we can next in a similar way eliminate x_2 from the last $(n-2)$ of these equations. After $k-1$ steps, $k \leq \min(m,n)$, of Gaussian elimination the matrix A has been reduced to the form

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots & & \vdots \\ & & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{mk}^{(k)} & \cdots & a_{mn}^{(k)} \end{pmatrix}, \qquad b^{(k)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_k^{(k)} \\ \vdots \\ b_m^{(k)} \end{pmatrix}, \tag{7.2.4}$$

where we have put $A^{(1)} = A$, $b^{(1)} = b$. The diagonal elements a_{11}, $a_{22}^{(2)}$, $a_{33}^{(3)}$, $\ldots$, which appear during the elimination are called **pivotal elements**.

Let A_k denote the kth leading principal submatrix of A. Since the determinant of a matrix does not change under row operations the determinant of A_k equals the product of the diagonal elements then by (7.2.5)

$$\det(A_k) = a_{11}^{(1)} \cdots a_{kk}^{(k)}, \quad k = 1:n.$$

For a square system, i.e. $m = n$, this implies that all pivotal elements $a_{ii}^{(i)}$, $i = 1:n$, in Gaussian elimination are nonzero if and only if $\det(A_k) \neq 0$, $k = 1:n$. In this case we can continue the elimination until after $(n-1)$ steps we get the single equation

$$a_{nn}^{(n)} x_n = b_n^{(n)} \qquad (a_{nn}^{(n)} \neq 0).$$

We have now obtained an upper triangular system $A^{(n)}x = b^{(n)}$, which can be solved recursively by back-substitution (7.2.1). We also have

$$\det(A) = a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)}. \tag{7.2.5}$$

We have seen that if in Gaussian elimination a zero pivotal element is encountered, i.e., $a_{kk}^{(k)} = 0$ for some $k \leq n$, then we cannot proceed and it seems the algorithm breaks down.

Example 7.2.1. Consider the system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

For $\epsilon \neq 1$ this system is nonsingular and has the unique solution $x_1 = -x_2 = -1/(1-\epsilon)$. However, when $a_{11} = \epsilon = 0$ the first step in Gaussian elimination cannot be carried through. The remedy here is obviously to interchange the two equations, which directly gives an upper triangular system.

Suppose that in step k of Gaussian elimination we have $a_{kk}^{(k)} = 0$. (The equations may have been reordered in previous steps, but we assume that the notations have been changed accordingly.) If A is nonsingular, then in particular its first k columns are linearly independent. This must also be true for the first k columns of the reduced matrix. Hence some element $a_{ik}^{(k)}$, $i = k : n$ must be nonzero, say $a_{pk}^{(k)} \neq 0$. By interchanging rows k and p this element can be taken as pivot and it is possible to proceed with the elimination. The important conclusion is that *any nonsingular system of equations can be reduced to triangular form by Gaussian elimination if appropriate row interchanges are used.*

Note that when rows are interchanged in A the same interchanges must be made in the elements of the vector b. Note also that the determinant formula (7.2.5) must be modified to

$$\det(A) = (-1)^s a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)}, \tag{7.2.6}$$

where s denotes the total number of row and columns interchanges performed.

If $\operatorname{rank}(A) < n$ then it is possible that at some step k we have $a_{ik}^{(k)} = 0$, $i = k : n$. If the entire submatrix $a_{ij}^{(k)}$, $i, j = k : n$, is zero, then $\operatorname{rank}(A) = k$ and we stop. Otherwise there is a nonzero element, say $a_{pq}^{(k)} \neq 0$, which can be brought into pivoting position by interchanging rows k and p and columns k and q. (Note that when columns are interchanged in A the same interchanges must be made in the elements of the solution vector x.) Proceeding in this way any matrix A can always be reduced to upper **trapezoidal form**,

$$A^{(r)} = \left(\begin{array}{ccc|ccc} a_{11}^{(1)} & \cdots & a_{1r}^{(1)} & a_{1,r+1}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & \ddots & \vdots & & \vdots & \vdots \\ \vdots & & a_{rr}^{(r)} & a_{r,r+1}^{(r)} & \cdots & a_{rn}^{(r)} \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right), \qquad b^{(r)} = \left(\begin{array}{c} b_1^{(1)} \\ \vdots \\ b_r^{(r)} \\ \hline b_{r+1}^{(r+1)} \\ \vdots \\ b_m^{(r+1)} \end{array}\right). \tag{7.2.7}$$

Here the number r of linearly independent rows in a matrix A equals the number of independent columns in A is the **rank** of A.

From the reduced form (7.2.7) we can read off the rank of A. The two rectangular zero blocks in $A^{(r)}$ have dimensions $(m-r) \times r$ and $(m-r) \times (n-r)$, respectively. We deduce the following:

1. The system $Ax = b$ has a unique solution if and only if $r = m = n$.

2. If $b_k^{(r+1)} = 0$, $k = r + 1 : m$, then the system $Ax = b$ is consistent and has an infinite number of solutions. We can assign arbitrary values to the last $n - r$ components of (the possibly permuted) solution vector x. The first r components are then uniquely determined and obtained using back-substitution with the nonsingular triangular matrix in the upper left corner.

3. If $b_k^{(r+1)} \neq 0$, for some $k > r$, the the system $Ax = b$ is inconsistent and has no solution. Then we have to be content with finding x such that the residual vector $r = b - Ax$ is small in some sense.

In principle, the reduced trapezoidal form (7.2.7) obtained by Gaussian elimination yields the rank of a the matrix A, and also answers the question whether the given system is consistent or not. However, this is the case only if exact arithmetic is used. In floating point calculations it may be difficult to decide if a pivot element, or an element in the transformed right hand side, should be considered as zero or nonzero. For example, a zero pivot in exact arithmetic will almost invariably be polluted by roundoff errors in such a way that it equals some small nonzero number. What tolerance to use to decide when a pivot should be taken to be zero will depend on the context. In order to treat this question in a satisfactory way we need the concept **numerical rank**, which will be introduced in Section 8.3.3.

Another question, which we have to leave unanswered for a while, concerns underdetermined and overdetermined systems, which arise quite frequently in practice! Problems where there are more parameters than needed to span the right hand side lead to underdetermined systems. In this case we need additional information in order to decide which solution to pick. On the other hand, overdetermined systems arise when there is more data than needed to determine the solution. In this case the system usually is inconsistent and there is no solution. Now the question can be posed, how to find a solution, which in some sense best approximates the right hand side? These questions are related, are best treated using *orthogonal* transformations rather than GE. This topic is again deferred to Chapter 8.

When the matrix A is square and of a full rank Gaussian Elimination can be described as follows:

Algorithm 7.2.2 Gaussian Elimination; square case.

Given a matrix $A = A^{(1)} \in \mathbf{R}^{n\times n}$ and a vector $b = b^{(1)} \in \mathbf{R}^n$, the following algorithm reduces the system $Ax = b$ to the upper triangular form, provided that the pivotal elements $a_{kk}^{(k)} \neq 0$, $k = 1 : n$:

for $k = 1 : n - 1$
  for $i = k + 1 : n$
    $l_{ik} := a_{ik}^{(k)} / a_{kk}^{(k)}$;
    for $j = k + 1 : n$
      $a_{ij}^{(k+1)} := a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$;

```
        end
        b_i^(k+1) := b_i^(k) − l_ik b_k^(k);
    end
end
```

Note that when l_{ik} is computed the element $a_{ik}^{(k)}$ is put equal to zero. Thus memory space can be saved by storing the multipliers in the lower triangular part of the matrix. If the multipliers l_{ik} are saved, then the operations on the vector b can be deferred to a later stage. This observation is important in that it shows that *when solving a sequence of linear systems*

$$Ax_i = b_i, \quad i = 1 : p,$$

with the same matrix A but different right hand sides, the operations on A only have to be carried out once!

From Algorithm 7.2.2 it follows that $(n-k)$ divisions and $(n-k)^2$ multiplications and additions are used in step k to transform the elements of A. A further $(n-k)$ multiplications and additions are used to transform the elements of b. Summing over k and neglecting low order terms we find that the total number of flams required by Gaussian elimination is

$$\sum_{k=1}^{n-1}(n-k)^2 \approx n^3/3, \qquad \sum_{k=1}^{n-1}(n-k) \approx n^2/2$$

for A and each right hand side respectively. Comparing with the approximately $\frac{1}{2}n^2$ flops needed to solve a triangular system we conclude that, except for very small values of n, *the reduction of A to triangular form dominates the work.*[12] (see Sections 7.4 and 7.8, respectively). Operation counts like these are meant only as a rough appraisal of the work and one should not assign too much meaning to their precise value. On modern computer architectures the rate of transfer of data between different levels of memory often limits the actual performance.

Algorithm 7.2.2 for Gaussian Elimination algorithm has three nested loops. It is possible to reorder these loops in $3 \cdot 2 \cdot 1 = 6$ ways. In each of those version the operations does the basic operation

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} - a_{kj}^{(k)} a_{ik}^{(k)} / a_{kk}^{(k)},$$

and only the ordering in which they are done differs. The version given above uses row operations and may be called the "kij" variant, where k refers to step number, i to row index, and j to column index. This version is not suitable for Fortran 77, and other languages in which matrix elements are stored and accessed sequentially by columns. In such a language the form "kji" should be preferred, which is the column oriented variant of Algorithm 7.2.2 (see Problem 5).

We now show that Gaussian elimination can be interpreted as computing the matrix factorization $A = LU$. The LU factorization is a prime example of the

[12] This conclusion is not in general true for banded and sparse systems

decompositional approach to matrix computation. This approach came into favor in the 1950s and early 1960s and has been named as one of the ten algorithms with most influence on science and engineering in the 20th century.

For simplicity we assume that $m = n$ and that GE can be carried out without pivoting. We will show that in this case GE provides a factorization of A into the product of a unit lower triangular matrix L and an upper triangular matrix U. Depending on whether the element a_{ij} lies on or above or below the principal diagonal we have

$$a_{ij}^{(n)} = \begin{cases} \ldots = a_{ij}^{(i+1)} = a_{ij}^{(i)}, & i \le j; \\ \ldots = a_{ij}^{(j+1)} = 0, & i > j. \end{cases}$$

Thus in GE the elements a_{ij}, $1 \le i, j \le n$, are transformed according to

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}, \quad k = 1 : p, \quad p = \min(i-1, j). \tag{7.2.8}$$

If these equations are summed for $k = 1 : p$, we obtain

$$\sum_{k=1}^{p} (a_{ij}^{(k+1)} - a_{ij}^{(k)}) = a_{ij}^{(p+1)} - a_{ij} = -\sum_{k=1}^{p} l_{ik} a_{kj}^{(k)}.$$

This can also be written

$$a_{ij} = \begin{cases} a_{ij}^{(i)} + \displaystyle\sum_{k=1}^{i-1} l_{ik} a_{kj}^{(k)}, & i \le j; \\ 0 + \displaystyle\sum_{k=1}^{j} l_{ik} a_{kj}^{(k)}, & i > j, \end{cases}$$

or, if we define $l_{ii} = 1$, $i = 1 : n$,

$$a_{ij} = \sum_{k=1}^{r} l_{ik} u_{kj}, \quad u_{kj} = a_{kj}^{(k)}, \quad r = \min(i, j). \tag{7.2.9}$$

However, these equations are equivalent to the matrix equation $A = LU$, where $L = (l_{ik})$ and $U = (u_{kj})$ are lower and upper triangular matrices, respectively. Hence GE computes a factorization of A into a product of a lower and an upper triangular matrix, the **LU factorization** of A.

It was shown in Section 7.2.2 that if A is nonsingular, then Gaussian elimination can always be carried through provided row interchanges are allowed. Also, such row interchanges are in general needed to ensure the numerical stability of Gaussian elimination. We now consider how the LU factorization has to be modified when such interchanges are incorporated.

Row interchanges and row permutations can be expressed as pre-multiplication with certain matrices, which we now introduce. A matrix

$$I_{ij} = (\ldots, e_{i-1}, e_j, e_{i+1}, \ldots, e_{j-1}, e_i, e_{j+1}),$$

which is equal to the identity matrix except that columns i and j have been interchanged is called a **transposition matrix**. If a matrix A is premultiplied by I_{ij} this results in the interchange of *rows* i and j. Similarly post-multiplication results in the interchange of *columns* i and j. $I_{ij}^T = I_{ij}$, and by its construction it immediately follows that $I_{ij}^2 = I$ and hence $I_{ij}^{-1} = I_{ij}$.

A **permutation matrix** $P \in \mathbf{R}^{n\times n}$ is a matrix whose columns are a permutation of the columns of the unit matrix, that is,

$$P = (e_{p_1}, \ldots, e_{p_n}),$$

where $(p_1, \ldots, p_n)$ is a permutation of $(1, \ldots, n)$. Notice that in a permutation matrix every row and every column contains just one unity element. The transpose P^T of a permutation matrix is therefore again a permutation matrix Since P is uniquely represented by the integer vector $(p_1, \ldots, p_n)$ it need never be explicitly stored.

If P is a permutation matrix then PA is the matrix A with its rows permuted and AP is A with its columns permuted. Any permutation may be expressed as a sequence of transposition matrices. Therefore any permutation matrix can be expressed as a product of transposition matrices $P = I_{i_1,j_1} I_{i_2,j_2} \cdots I_{i_k,j_k}$. Since $I_{i_p,j_p}^{-1} = I_{i_p,j_p}$, we have

$$P^{-1} = I_{i_k,j_k} \cdots I_{i_2,j_2} I_{i_1,j_1} = P^T,$$

that is permutation matrices are orthogonal and P^T effects the reverse permutation.

Assume that in the kth step, $k = 1 : n-1$, we select the pivot element from row p_k, and interchange the rows k and p_k. Notice that in these row interchanges also previously computed multipliers l_{ij} must take part. At completion of the elimination, we have obtained lower and upper triangular matrices L and U. We now make the important observation that these are the same triangular factors that are obtained if we *first* carry out the row interchanges $k \leftrightarrow p_k$, $k = 1 : n-1$, on the *original matrix* A to get a matrix PA, where P is a permutation matrix, and then perform Gaussian elimination on PA *without any interchanges*. This means that Gaussian elimination with row interchanges computes the LU factors of the matrix PA. We now summarize the results and prove the uniqueness of the LU factorization:

Theorem 7.2.1. *The LU factorization*

Let $A \in \mathbf{R}^{n\times n}$ be a given nonsingular matrix. Then there is a permutation matrix P such that Gaussian elimination on the matrix $\tilde{A} = PA$ can be carried out without pivoting giving the factorization

$$PA = LU, \tag{7.2.10}$$

where $L = (l_{ij})$ is a unit lower triangular matrix and $U = (u_{ij})$ an upper triangular matrix. The elements in L and U are given by

$$u_{ij} = \tilde{a}_{ij}^{(i)}, \quad 1 \le i \le j \le n,$$

and

$$l_{ij} = \tilde{l}_{ij}, \quad l_{ii} = 1, \quad 1 \le j < i \le n,$$

where $\tilde{l}_{ij}$ are the multipliers occurring in the reduction of $\tilde{A} = PA$. For a fixed permutation matrix P, this factorization is uniquely determined.

Proof. We prove the uniqueness. Suppose we have two factorizations

$$PA = L_1U_1 = L_2U_2.$$

Since PA is nonsingular so are the factors, and it follows that $L_2^{-1}L_1 = U_2U_1^{-1}$. The left-hand matrix is the product of two unit lower triangular matrices and is therefore unit lower triangular, while the right hand matrix is upper triangular. It follows that both sides must be the identity matrix. Hence $L_2 = L_1$, and $U_2 = U_1$. □

Writing $PAx = LUx = L(Ux) = Pb$ it follows that if the LU factorization of PA is known, then the solution x can be computed by solving the two triangular systems

$$Ly = Pb, \qquad Ux = y, \tag{7.2.11}$$

which involves about $2 \cdot \frac{1}{2}n^2 = n^2$ flops.

Although the LU factorization is just a different interpretation of Gaussian elimination it turns out to have important conceptual advantages. It divides the solution of a linear system into two independent steps:

1. The factorization $PA = LU$.
2. Solution of the systems $Ly = Pb$ and $Ux = y$.

As an example of the use of the factorization consider the problem of solving the transposed system $A^Tx = b$. Since $P^TP = I$, and

$$(PA)^T = A^TP^T = (LU)^T = U^TL^T,$$

we have that $A^TP^TPx = U^T(L^TPx) = b$. It follows that $\tilde{x} = Px$ can be computed by solving the two triangular systems

$$U^Tc = b, \qquad L^T\tilde{x} = c. \tag{7.2.12}$$

We then obtain $x = P^{-1}\tilde{x}$ by applying the interchanges $k \leftrightarrow p_k$, in reverse order $k = n-1 : 1$ to $\tilde{x}$. Note that it is not at all trivial to derive this algorithm from the presentation of Gaussian elimination in the previous section!

In the general case when $A \in \mathbf{R}^{m\times n}$ of $\operatorname{rank}(A) = r \le \min\{m, n\}$, it can be shown that matrix $P_rAP_c \in \mathbf{R}^{m\times n}$ can be factored into a product of a unit lower **trapezoidal matrix** $L \in \mathbf{R}^{m\times r}$ and an upper trapezoidal matrix $U \in \mathbf{R}^{r\times n}$. Here P_r and P_c are permutation matrices performing the necessary row and column permutations, respectively. The factorization can be written in block form as

$$P_rAP_c = LU = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \end{pmatrix}, \tag{7.2.13}$$

where the matrices L_{11} and U_{11} are triangular and non-singular. Note that the block L_{21} is empty if the matrix A has full row rank, i.e. $r = m$; the block U_{12} is empty if the matrix A has full column rank, i.e. $r = n$.

To solve the system

$$P_r A P_c (P_c^T x) = LU\tilde{x} = P_r b = \tilde{b}, \quad x = P_c \tilde{x},$$

using this factorization we set $y = Ux$ and consider

$$\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} y = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}.$$

This uniquely determines y as the solution to $L_{11}y = \tilde{b}_1$. Hence *the system is consistent if and only if* $L_{21}y = \tilde{b}_2$. Further, we have $U\tilde{x} = y$, or

$$\begin{pmatrix} U_{11} & U_{12} \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = y.$$

For an arbitrary $\tilde{x}_2$ this system uniquely determines $\tilde{x}_1$ as the solution to the triangular system

$$U_{11}\tilde{x}_1 = y - U_{12}\tilde{x}_2.$$

Thus, if consistent the system has a unique solution only if A has full column rank.

7.2.3 Elementary Elimination Matrices

The reduction of a matrix to triangular form by Gaussian elimination can be expressed entirely in matrix notations using **elementary elimination matrices**. This way of looking at Gaussian elimination, first systematically exploited by J. H. Wilkinson,[13] has the advantage that it suggests ways of deriving other matrix factorization

Elementary elimination matrices are lower triangular matrices of the form

$$L_j = I + l_j e_j^T = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{j+1,j} & 1 & \\ & & \vdots & & \ddots \\ & & l_{n,j} & & 1 \end{pmatrix}, \tag{7.2.14}$$

where only the elements *below* the main diagonal in the jth column differ from the

[13] James Hardy Wilkinson (1919–1986) English mathematician graduated from Trinity College, Cambridge. He became Alan Turing's assistant at the National Physical Laboratory in London in 1946, where he worked on the ACE computer project. He did pioneering work on numerical methods for solving linear systems and eigenvalue problems and developed software and libraries of numerical routines.

unit matrix. If a vector x is premultiplied by L_j we get

$$L_j x = (I + l_j e_j^T)x = x + l_j x_j = \begin{pmatrix} x_1 \\ \vdots \\ x_j \\ x_{j+1} + l_{j+1,j} x_j \\ \vdots \\ x_n + l_{n,j} x_j \end{pmatrix},$$

i.e., to the last $n - j$ components of x are *added* multiples of the component x_j. Since $e_j^T l_j = 0$ it follows that

$$(I - l_j e_j^T)(I + l_j e_j^T) = I + l_j e_j^T - l_j e_j^T - l_j (e_j^T l_j) e_j^T = I$$

so we have

$$L_j^{-1} = I - l_j e_j^T.$$

The computational significance of elementary elimination matrices is that they can be used to introduce zero components in a column vector x. Assume that $e_k^T x = x_k \neq 0$. We show that there is a unique elementary elimination matrix $L_k^{-1} = I - l_k e_k^T$ such that

$$L_k^{-1}(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)^T = (x_1, \ldots, x_k, 0, \ldots, 0)^T.$$

Since the last $n-k$ components of $L_k^{-1}x$ are to be zero it follows that we must have $x_i - l_{i,k} x_k = 0$, $i = k+1 : n$, and hence

$$l_k = (0, \ldots, 0, x_{k+1}/x_k, \ldots, x_n/x_k)^T.$$

The product of two elementary elimination matrices $L_j L_k$ is a lower triangular matrix which differs from the unit matrix in the two columns j and k below the main diagonal,

$$L_j L_k = (I + l_j e_j^T)(I + l_k e_k^T) = I + l_j e_j^T + l_k e_k^T + l_j (e_j^T l_k) e_k^T.$$

If $j \leq k$, then $e_j^T l_k = 0$, and the following simple multiplication rule holds:

$$L_j L_k = I + l_j e_j^T + l_k e_k^T, \quad j \leq k. \tag{7.2.15}$$

Note that no products of the elements l_{ij} occur! However, if $j > k$, then in general $e_j^T l_k \neq 0$, and the product $L_j L_k$ has a more complex structure.

We now show that Gaussian elimination with partial pivoting can be accomplished by premultiplication of A by a sequence of elementary elimination matrices combined with transposition matrices to express the interchange of rows. For simplicity we first consider the case when $\operatorname{rank}(A) = m = n$. In the first step assume that $a_{p_1,1} \neq 0$ is the pivot element. We then interchange rows 1 and p_1 in A by premultiplication of A by a transposition matrix,

$$\tilde{A} = P_1 A, \quad P_1 = I_{1,p_1}.$$

If we next premultiply $\tilde{A}$ by the elementary elimination matrix

$$L_1^{-1} = I - l_1 e_1^T, \quad l_{i1} = \tilde{a}_{i1}/\tilde{a}_{11}, \quad i = 2:n,$$

this will zero out the elements under the main diagonal in the first column, i.e.

$$A^{(2)} e_1 = L_1^{-1} P_1 A e_1 = \tilde{a}_{11} e_1.$$

All remaining elimination steps are similar to this first one. The second step is achieved by forming $\tilde{A}^{(2)} = P_2 A^{(2)}$ and

$$A^{(3)} = L_2^{-1} P_2 A^{(2)} = L_2^{-1} P_2 L_1^{-1} P_1 A.$$

Here $P_2 = I_{2,p_2}$, where $a_{p_2,2}^{(2)}$ is the pivot element from the second column and $L_2^{-1} = I - l_2 e_2^T$ is an elementary elimination matrix with nontrivial elements equal to $l_{i2} = \tilde{a}_{i2}^{(2)}/\tilde{a}_{22}^{(2)}$, $i = 3:n$. Continuing, we have after $n-1$ steps reduced A to upper triangular form

$$U = L_{n-1}^{-1} P_{n-1} \cdots L_2^{-1} P_2 L_1^{-1} P_1 A. \tag{7.2.16}$$

To see that (7.2.16) is equivalent with the LU factorization of PA we first note that since $P_2^2 = I$ we have after the first two steps that

$$A^{(3)} = L_2^{-1} \tilde{L}_1^{-1} P_2 P_1 A$$

where

$$\tilde{L}_1^{-1} = P_2 L_1^{-1} P_2 = I - (P_2 l_1)(e_1^T P_2) = I - \tilde{l}_1 e_1^T.$$

Hence $\tilde{L}_1^{-1}$ is again an elementary elimination matrix of the same type as L_1^{-1}, except that two elements in l_1 have been interchanged. Premultiplying by $\tilde{L}_1 L_2$ we get

$$\tilde{L}_1 L_2 A^{(3)} = P_2 P_1 A,$$

where the two elementary elimination matrices on the left hand side combine trivially. Proceeding in a similar way it can be shown that (7.2.16) implies

$$\tilde{L}_1 \tilde{L}_2 \ldots \tilde{L}_{n-1} U = P_{n-1} \ldots P_2 P_1 A,$$

where $\tilde{L}_{n-1} = L_{n-1}$ and

$$\tilde{L}_j = I + \tilde{l}_j e_j^T, \qquad \tilde{l}_j = P_{n-1} \cdots P_{j+1} l_j, \quad j = 1:n-2.$$

Using the result in (7.2.15), the elimination matrices can trivially be multiplied together and it follows that

$$PA = LU, \quad P = P_{n-1} \cdots P_2 P_1,$$

where the elements in L are given by $l_{ij} = \tilde{l}_{ij}$, $l_{ii} = 1$, $1 \le j < i \le n$. This is the LU factorization of Theorem 7.2.1. It is important to note that nothing new, except the notations, has been introduced. In particular, the transposition matrices and

elimination matrices used here are, of course, never explicitly stored in a computer implementation.

In Gaussian elimination we use in the kth step the pivot row to eliminate elements *below* the main diagonal in column k. In **Gauss–Jordan elimination**[14] the elements *above* the main diagonal are eliminated simultaneously. After $n-1$ steps the matrix A has then been transformed into a *diagonal* matrix containing the nonzero pivot elements. Gauss–Jordan elimination was used in many early versions of linear programming and also for implementing stepwise regression in statistics.

Gauss–Jordan elimination can be described by introducing the elementary matrices

$$M_j = \begin{pmatrix} 1 & & & l_{1j} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & l_{j-1,j} & & & \\ & & & 1 & & & \\ & & & l_{j+1,j} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & l_{n,j} & & & 1 \end{pmatrix}. \tag{7.2.17}$$

If partial pivoting is carried out we can write, cf. (7.2.16)

$$D = M_n M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1 A,$$

where the l_{ij} are chosen to annihilate the (i,j)th element. Multiplying by D^{-1} we get

$$A^{-1} = D^{-1} M_n M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1. \tag{7.2.18}$$

This expresses the inverse of A as a product of elimination and transposition matrices, and is called the **product form of the inverse**. The operation count for this elimination process is $\approx n^3/2$ flops, i.e., higher than for the LU factorization by Gaussian elimination. For some parallel implementations Gauss–Jordan elimination may still have advantages.

To solve a linear system $Ax = b$ we apply these transformations to the vector b to obtain

$$x = A^{-1}b = D^{-1} M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1 b. \tag{7.2.19}$$

This requires n^2 flops. Note that no back-substitution is needed!

The stability of Gauss–Jordan elimination has been analyzed by Peters and Wilkinson [53]. They remark that the residuals $b - A\bar{x}$ corresponding to the Gauss–Jordan solution $\bar{x}$ can be a larger by a factor $\kappa(A)$ than those corresponding to the solution by Gaussian elimination. Although the method is not backward stable in general it can be shown to be stable for so called diagonally dominant matrices (see Definition 7.2.4). It is also forward stable, i.e., will give about the same numerical accuracy in the computed solution $\bar{x}$ as Gaussian elimination.

[14] Named after Wilhelm Jordan (1842–1899), who used this method to compute the covariance matrix in least squares problems.

7.2.4 Pivoting Strategies

We saw that in Gaussian elimination row and column interchanges were needed in case a zero pivot was encountered. A basic rule of numerical computation says that if an algorithm breaks down when a zero element is encountered, then we can expect some form of instability and loss of precision also for nonzero but small elements! Again, this is related to the fact that in floating point computation the difference between a zero and nonzero number becomes fuzzy because of the effect of rounding errors.

Example 7.2.2. For $\epsilon \neq 1$ the system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

is nonsingular and has the unique solution $x_1 = -x_2 = -1/(1-\epsilon)$. Suppose $\epsilon = 10^{-6}$ is accepted as pivot in Gaussian elimination. Multiplying the first equation by 10^6 and subtracting from the second we obtain $(1-10^6)x_2 = -10^6$. By rounding this could give $x_2 = 1$, which is correct to six digits. However, back-substituting to obtain x_1 we get $10^{-6}x_1 = 1-1$, or $x_1 = 0$, which is completely wrong.

The simple example above illustrates that in general it is necessary to perform row (and/or column) interchanges *not only when a pivotal element is exactly zero, but also when it is small*. The two most common pivoting strategies are **partial pivoting** and **complete pivoting**. In partial pivoting the pivot is taken as the largest element in magnitude in the unreduced part of the kth column. In complete pivoting the pivot is taken as the largest element in magnitude in the whole unreduced part of the matrix.

Partial Pivoting. At the start of the kth stage choose interchange rows k and r, where r is the smallest integer for which

$$|a_{rk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|. \tag{7.2.20}$$

Complete Pivoting. At the start of the kth stage interchange rows k and r and columns k and s, where r and s are the smallest integers for which

$$|a_{rs}^{(k)}| = \max_{k \leq i,j \leq n} |a_{ij}^{(k)}|. \tag{7.2.21}$$

Complete pivoting requires $O(n^3)$ in total compared with only $O(n^2)$ for partial pivoting. Hence, complete pivoting involves a fairly high overhead since about as many arithmetic comparisons as floating point operations has to be performed. Since practical experience shows that partial pivoting works well, this is the standard choice. Note, however, that when $\operatorname{rank}(A) < n$ then complete pivoting must be used

A major breakthrough in the understanding of GE came with the famous backward rounding error analysis of Wilkinson [65, 1961]. Using the standard model for

floating point computation Wilkinson showed that the computed triangular factors $\bar{L}$ and $\bar{U}$ of A, obtained by Gaussian elimination *are the exact triangular factors of a perturbed matrix*

$$\bar{L}\bar{U} = A + E, \quad E = (e_{ij})$$

where, since e_{ij} is the sum of $\min(i-1, j)$ rounding errors

$$|e_{ij}| \leq 3u \min(i-1, j) \max_k |\bar{a}_{ij}^{(k)}|. \tag{7.2.22}$$

Note that the above result holds without any assumption about the size of the multipliers. *This shows that the purpose of any pivotal strategy is to avoid growth in the size of the computed elements $\bar{a}_{ij}^{(k)}$, and that the size of the multipliers is of no consequence* (see the remark on possible large multipliers for positive-definite matrices, Section 7.4.2).

The growth of elements during the elimination is usually measured by the **growth ratio**.

Definition 7.2.2.

Let $a_{ij}^{(k)}$, $k = 2 : n$, be the elements in the kth stage of Gaussian elimination applied to the matrix $A = (a_{ij})$. Then the **growth ratio** *in the elimination is*

$$\rho_n = \max_{i,j,k} |a_{ij}^{(k)}| / \max_{i,j} |a_{ij}|. \tag{7.2.23}$$

It follows that $E = (e_{ij})$ can be bounded component-wise by

$$|E| \leq 3\rho_n u \max_{ij} |a_{ij}| F. \tag{7.2.24}$$

where F the matrix with elements $f_{i,j} = \min\{i-1, j\}$. Strictly speaking this is not correct unless we use the growth factor $\bar{\rho}_n$ for the *computed elements*. Since this quantity differs insignificantly from the theoretical growth factor ρ_n in (7.2.23), we ignore this difference here and in the following. Slightly refining the estimate

$$\|F\|_\infty \leq (1 + 2 + \cdots + n) - 1 \leq \frac{1}{2} n(n+1) - 1$$

and using $\max_{ij} |a_{ij}| \leq \|A\|_\infty$, we get the normwise backward error bound:

Theorem 7.2.3.

Let $\bar{L}$ and $\bar{U}$ be the computed triangular factors of A, obtained by GE with floating-point arithmetic with unit roundoff u has been used, there is a matrix E such that

$$\bar{L}\bar{U} = A + E, \quad \|E\|_\infty \leq 1.5n^2 \rho_n u \|A\|_\infty. \tag{7.2.25}$$

If pivoting is employed so that the computed multipliers satisfy the inequality

$$|l_{ik}| \leq 1, \quad i = k+1 : n.$$

Then it can be shown that an estimate similar to (7.2.25) holds with the constant 1 instead of 1.5. For both partial and complete pivoting it holds that

$$|a_{ij}^{(k+1)}| < |a_{ij}^{(k)}| + |l_{ik}||a_{kj}^{(k)}| \leq |a_{ij}^{(k)}| + |a_{kj}^{(k)}| \leq 2 \max_{i,j} |a_{ij}^{(k)}|,$$

and the bound $\rho_n \leq 2^{n-1}$ follows by induction. For partial pivoting this bound is the best possible, and can be attained for special matrices. For complete pivoting a much better bound can be proved, and in practice the growth very seldom exceeds n; see Section 7.6.2.

A pivoting scheme that gives a pivot of size between that of partial and complete pivoting is **rook pivoting**. In this scheme we pick a pivot element which is largest in magnitude in *both its column and its row.*

Rook Pivoting. At the start of the kth stage rows k and r and columns k and s are interchanged, where

$$|a_{rs}^{(k)}| = \max_{k \leq i \leq n} |a_{ij}^{(k)}| = \max_{k \leq j \leq n} |a_{ij}^{(k)}|. \tag{7.2.26}$$

We start by finding the element of maximum magnitude in the first column. If this element is also of maximum magnitude in its row we accept it as pivot. Otherwise we compare the element of maximum magnitude in the row with other elements in its column, etc. The name derives from the fact that the pivot search resembles the moves of a rook in chess; see Figure 7.2.1..

1	10	1	2	4
0	5	2	9	8
3	0	4	1	7
2	2	5	6	1
1	4	3	2	3

Figure 7.2.1. *Illustration of rook pivoting in a* 5×5 *matrix with positive integer entries as shown. The* $(2, 4)$ *element 9 is chosen as pivot.*

Rook pivoting involves at least twice as many comparisons as partial pivoting. In the worst case it can take $O(n^3)$ comparisons, i.e., the same order of magnitude as for complete pivoting. Numerical experience shows that the cost of rook pivoting usually equals a small multiple of the cost for partial pivoting. A pivoting related to rook pivoting is used in the solution of symmetric indefinite systems; see Sec. 7.3.4.

It is important to realize that the choice of pivots is influenced by the scaling of equations and unknowns. If, for example, the unknowns are physical quantities a different choices of units will correspond to a different scaling of the unknowns and the columns in A. Partial pivoting has the important property of being invariant

under column scalings. In theory we could perform partial pivoting by *column* interchanges, which then would be invariant under row scalings. but in practice this turns out to be less satisfactory. Likewise, an unsuitable column scaling can also make complete pivoting behave badly.

For certain important classes of matrices a bound independent of n can be given for the growth ratio in Gaussian elimination without pivoting or with partial pivoting. For these Gaussian elimination is backward stable.

- If A is real symmetric matrix $A = A^T$ and positive definite (i.e. $x^T Ax > 0$ for all $x \neq 0$) then $\rho_n(A) \leq 1$ with no pivoting (see Theorem 7.3.7).
- If A is row or column diagonally dominant then $\rho_n \leq 2$ with no pivoting.
- If A is Hessenberg then $\rho_n \leq n$ with partial pivoting.
- If A is tridiagonal then $\rho_n \leq 2$ with partial pivoting.

For the last two cases we refer to Sec. 7.4. We now consider the case when A is diagonally dominant.

Definition 7.2.4. *A matrix A is said to be* **diagonally dominant** *by rows, if*

$$\sum_{j \neq i} |a_{ij}| \leq |a_{ii}|, \quad i = 1 : n. \tag{7.2.27}$$

A is **diagonally dominant** *by columns if A^T is diagonally dominant by rows.*

Theorem 7.2.5.

Let A be nonsingular and diagonally dominant by rows or columns. Then A has an LU factorization without pivoting and the growth ratio $\rho_n(A) \leq 2$. If A is diagonally dominant by columns, then the multipliers in this LU factorization satisfy $|l_{ij}| \leq 1$, for $1 \leq j < i \leq n$.

Proof. (Wilkinson [65, pp.288–289])

Assume that A is nonsingular and diagonally dominant by columns. Then $a_{11} \neq 0$, since otherwise the first column would be zero and A singular. In the first stage of Gaussian elimination without pivoting we have Hence

$$a_{ij}^{(2)} = a_{ij} - l_{i1} a_{1j}, \quad l_{i1} = a_{i1}/a_{11}, \quad i, j \geq 2, \tag{7.2.28}$$

where

$$\sum_{i=2}^{n} |l_{i1}| \leq \sum_{i=2}^{n} |a_{i1}|/|a_{11}| \leq 1. \tag{7.2.29}$$

For $j = i$, using the definition and (7.2.29), it follows that

$$\begin{aligned} |a_{ii}^{(2)}| \geq |a_{ii}| - |l_{i1}|\,|a_{1i}| &\geq \sum_{j \neq i} |a_{ji}| - \Big(1 - \sum_{j \neq 1,i} |l_{j1}|\Big)|a_{1i}| \\ &= \sum_{j \neq 1,i} \big(|a_{ji}| + |l_{j1}||a_{1i}|\big) \geq \sum_{j \neq 1,i} |a_{ji}^{(2)}|. \end{aligned}$$

Hence the reduced matrix $A^{(2)} = (a_{ij}^{(2)})$, is also nonsingular and diagonally dominant by columns. It follows by induction that all matrices $A^{(k)} = (a_{ij}^{(k)})$, $k = 2 : n$ are nonsingular and diagonally dominant by columns.

Further using (7.2.28) and (7.2.29), for $i \geq 2$,

$$\sum_{i=2}^{n} |a_{ij}^{(2)}| \leq \sum_{i=2}^{n} \left(|a_{ij}| + |l_{i1}||a_{1j}| \right) \leq \sum_{i=2}^{n} |a_{ij}| + |a_{1j}| \sum_{i=2}^{n} |l_{i1}|$$
$$\leq \sum_{i=2}^{n} |a_{ij}| + |a_{1j}| = \sum_{i=1}^{n} |a_{ij}|.$$

Hence the sum of the moduli of the elements of any column of $A^{(k)}$ does not increase as k increases. Hence

$$\max_{i,j,k} |a_{ij}^{(k)}| \leq \max_{i,k} \sum_{j=k}^{n} |a_{ij}^{(k)}| \leq \max_{i} \sum_{j=1}^{n} |a_{ij}| \leq 2 \max_{i} |a_{ii}| = 2 \max_{ij} |a_{ij}|.$$

It follows that

$$\rho_n = \max_{i,j,k} |a_{ij}^{(k)}| / \max_{i,j} |a_{ij}| \leq 2.$$

The proof for matrices which are *row* diagonally dominant is similar. (Notice that Gaussian elimination with pivoting essentially treats rows and columns symmetrically!) □

We conclude that for a row or column diagonally dominant matrix Gaussian elimination without pivoting is backward stable. If A is diagonally dominant by rows then the multipliers can be arbitrarily large, but this does not affect the stability.

If (7.2.27) holds with strict inequality for all i, then A is said to be **strictly diagonally dominant** by rows. If A is strictly diagonally dominant, then it can be shown that all reduced matrices have the same property. In particular, all pivot elements must then be strictly positive and the nonsingularity of A follows. We mention a useful result for strictly diagonally dominant matrices.

Lemma 7.2.6.

Let A be strictly diagonally dominant by rows, and set

$$\alpha = \min_i \alpha_i, \quad \alpha_i := |a_{ii}| - \sum_{j \neq i} |a_{ij}| > 0, \quad i = 1 : n. \tag{7.2.30}$$

Then A is nonsingular, and $\|A^{-1}\|_\infty \leq \alpha^{-1}$.

Proof. By the definition of a subordinate norm (7.1.50) we have

$$\frac{1}{\|A^{-1}\|_\infty} = \inf_{y \neq 0} \frac{\|y\|_\infty}{\|A^{-1}y\|_\infty} = \inf_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \min_{\|x\|_\infty = 1} \|Ax\|_\infty.$$

Assume that equality holds in (7.2.30) for $i = k$. Then

$$\frac{1}{\|A^{-1}\|_\infty} = \min_{\|x\|_\infty=1} \max_i \Big| \sum_j a_{ij}x_j \Big| \geq \min_{\|x\|_\infty=1} \Big| \sum_j a_{kj}x_j \Big|$$
$$\geq |a_{kk}| - \sum_{j,j\neq k} |a_{kj}| = \alpha.$$

□

If A is strictly diagonally dominant by columns, then since $\|A\|_1 = \|A^T\|_\infty$ it holds that $\|A^{-1}\|_1 \leq \alpha^{-1}$. If A is strictly diagonally dominant in both rows and columns, then from $\|A\|_2 \leq \sqrt{\|A\|_1\|A\|_\infty}$ it follows that $\|A^{-1}\|_2 \leq \alpha^{-1}$.

7.2.5 Computational Variants

In Gaussian Elimination, as described in Sec. 7.2.3, the kth step consists of modifying the unreduced part of the matrix by an *outer product* of the vector of multipliers and the pivot row. Using the equivalence of Gaussian elimination and LU factorization it is easy to see that the the calculations can be arranged in several diferent ways so that the elements in L and U are determined directly.

For simplicity, we first assume that any row or column interchanges on A have been carried out in advance. The matrix equation $A = LU$ written in componentwise form (see (7.2.9))

$$a_{ij} = \sum_{k=1}^{r} l_{ik}u_{kj}, \quad 1 \leq i, j \leq n, \quad r = \min(i, j),$$

together with the normalization conditions $l_{ii} = 1$, $i = 1 : n$, can be thought of as $n^2 + n$ equations for the $n^2 + n$ unknown elements in L and U. We can solve these equations in n steps, $k = 1 : n$, where in the kth step we use the equations

$$a_{kj} = \sum_{p=1}^{k} l_{kp}u_{pj}, \quad j \geq k, \qquad a_{ik} = \sum_{p=1}^{k} l_{ip}u_{pk}, \quad i > k \tag{7.2.31}$$

to determine the kth *row* of U and the kth *column* of L. In this algorithm the main work is performed in matrix-vector multiplications.

Algorithm 7.2.3 Doolittle's Algorithm.

for $k = 1 : n$
  for $j = k : n$
    $u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj};$
  end
  for $i = k + 1 : n$

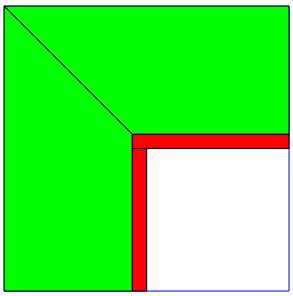

Figure 7.2.2. *Computations in the kth step of Doolittle's method.*

$$l_{ik} = \Big(a_{ik} - \sum_{p=1}^{k-1} l_{ip}u_{pk}\Big)/u_{kk};$$

end

$l_{kk} = 1;$

end

Since the LU factorization is unique this algorithm produces the same factors L and U as Gaussian elimination. In fact, successive partial sums in the equations (7.2.31) equal the elements $a_{ij}^{(k)}$, $j > k$, in Gaussian elimination. It follows that if each term in (7.2.31) is rounded separately, the compact algorithm is also *numerically equivalent* to Gaussian elimination. If the inner products can be accumulated in higher precision, then the compact algorithm is less affected by rounding errors. Algorithm 7.2.3 is usually referred to as Doolittle's algorithm. In Crout's algorithm the upper triangular matrix U is normalized to have a unit diagonal. [15]

Algorithm 7.2.5 can be modified to include partial pivoting. Changing the order of operations, we first calculate $\tilde{l}_{ik} = l_{ik}u_{kk}$, $i = k : n$, and determine the element of maximum magnitude. The corresponding row is then permuted to pivotal position. In this row exchange the already computed part of L and remaining part of A also take part. Next we normalize by setting $l_{kk} = 1$, which determines l_{ik}, $i = 1 : k$, and also u_{kk}. Finally, the remaining part of the kth row in U is computed.

It is possible to sequence the computations in Doolittle's and Crout's algorithms in many different ways. Indeed any element in $(L\backslash U)$ can be computed as soon as the corresponding elements in L to the left and in U above have been deter-

[15] In the days of hand computations these algorithms had the advantage that they did away with the necessity in Gaussian elimination to write down $\approx n^3/3$ intermediate results—one for each multiplication.

mined. For example, three possible orderings are schematically illustrated below,

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 3 & 3 & 3 & 3 \\ 2 & 4 & 5 & 5 & 5 \\ 2 & 4 & 6 & 7 & 7 \\ 2 & 4 & 6 & 8 & 9 \end{pmatrix}, \quad \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 3 & 5 & 7 & 9 \\ 2 & 4 & 5 & 7 & 9 \\ 2 & 4 & 6 & 7 & 9 \\ 2 & 4 & 6 & 8 & 9 \end{pmatrix}, \quad \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 3 & 5 & 7 & 9 \\ 4 & 4 & 5 & 7 & 9 \\ 6 & 6 & 6 & 7 & 9 \\ 8 & 8 & 8 & 8 & 9 \end{pmatrix}.$$

Here the entries indicate in which step a certain element l_{ij} and r_{ij} is computed, so the first example corresponds to the ordering in the algorithm given above. (Compare the comments after Algorithm 7.2.2.) Note that it is *not* easy to do complete pivoting with any of these variants.

The Bordering Method

Before the kth step, $k = 1 : n$, of the bordering method we have have computed the LU-factorization $A_{11} = L_{11}U_{11}$ of the leading principal submatrix A_{11} of order $k-1$ of A. To proceed we seek the LU-factorization

$$\begin{pmatrix} A_{11} & a_{1k} \\ a_{k1}^T & \alpha_{kk} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ l_{k1}^T & 1 \end{pmatrix} \begin{pmatrix} U_{11} & u_{1k} \\ 0 & u_{kk} \end{pmatrix}.$$

Identifying the (1,2)-blocks we find

$$L_{11}u_{1k} = a_{1k}, \tag{7.2.32}$$

which is a lower triangular system for u_{1k}. Identifying the (2,1)-blocks and transposing gives

$$U_{11}^T l_{k1} = a_{k1}, \tag{7.2.33}$$

another lower triangular system for l_{k1}. Finally, from the (2,2)-block we get $l_{k1}^T u_{1k} + u_{kk} = \alpha_{kk}$, or

$$u_{kk} = \alpha_{kk} - l_{k1}^T u_{1k}. \tag{7.2.34}$$

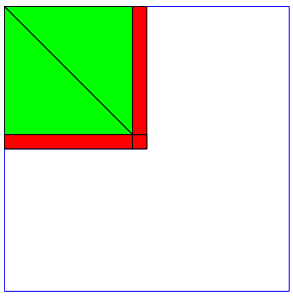

Figure 7.2.3. *Computations in the kth step of the bordering method.*

The main work in this variant is done in solving the triangular systems (7.2.32) and (7.2.33). A drawback of the bordering method is that it cannot be combined with partial pivoting.

The Sweep Methods

In the **column sweep** method at the kth step the first k columns of L and U in LU-factorization of A are computed. Assume that we have computed L_{11}, L_{21}, and U_{11} in the factorization

$$\begin{pmatrix} A_{11} & a_{1k} \\ A_{21} & a_{2k} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & l_{2k} \end{pmatrix} \begin{pmatrix} U_{11} & u_{1k} \\ 0 & u_{kk} \end{pmatrix} \in \mathbf{R}^{n\times k}.$$

As in the bordering method, identifying the (1,2)-blocks we find

$$L_{11}u_{1k} = a_{1k}, \tag{7.2.35}$$

a lower triangular system for u_{1k}. From the (2,2)-blocks we get $L_{21}u_{1k} + l_{2k}u_{kk} = a_{2k}$, or

$$l_{2k}u_{kk} = a_{2k} - L_{21}u_{1k}. \tag{7.2.36}$$

Together with the normalizing condition that the first component in the vector l_{2k} equals one this determines u_{kk} and l_{2k}.

Partial pivoting can be implemented with this method as follows. When the right hand side in (7.2.36) has been evaluated we determine the element of maximum modulus in the vector $a_{2k} - L_{21}u_{1k}$. We then permute this element to top position and perform the same row exchanges in L_{21}^T and the unprocessed part of A.

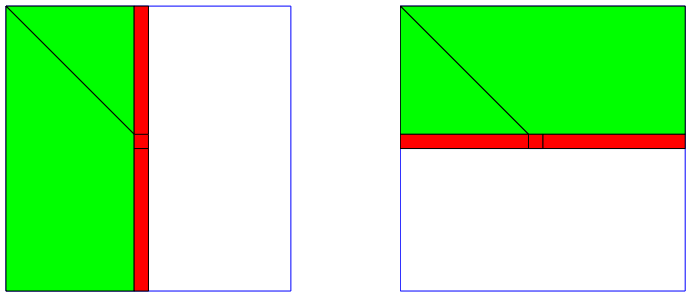

Figure 7.2.4. *Computations in the kth step of the sweep methods. Left: The column sweep method. Right: The row sweep method.*

In the column sweep method L and U are determined column by column. It possible to determine L and U row by row. In the kth step of this **row sweep** method the kth row of A is processed and we write

$$\begin{pmatrix} A_{11} & A_{12} \\ a_{k1}^T & a_{k2}^T \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ l_{k1}^T & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & u_{2k}^T \end{pmatrix} \in \mathbf{R}^{k\times n}.$$

Identifying the (2,1)- and (2,2)-blocks we get

$$U_{11}^T l_{k1} = a_{k1}, \tag{7.2.37}$$

and

$$u_{2k}^T = a_{k2}^T - l_{k1}U_{12}. \tag{7.2.38}$$

Note that Doolittle's method can be viewed as alternating between the two sweep methods.

Consider now the case when $A \in \mathbf{R}^{m\times n}$ is a rectangular matrix with rank $(A) = r = \min(m, n)$. If $m > n$ it is advantageous to process the matrix column by column. Then after n steps we have $AP_c = LU$, where L is lower trapezoidal,

$$L = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \in \mathbf{R}^{m\times n}, \tag{7.2.39}$$

and $U \in \mathbf{R}^{n\times n}$ is square upper triangular. If $m < n$ and the matrix is processed row by row, we have after n steps an LU factorization with $L \in \mathbf{R}^{m\times m}$ and

$$U = (\, U_{11} \quad U_{12}\,) \in \mathbf{R}^{m\times n}$$

upper trapezoidal.

7.2.6 Computing the Inverse

If the inverse matrix A^{-1} is known, then the solution of $Ax = b$ can be obtained through a matrix vector multiplication by $x = A^{-1}b$. This is theoretically satisfying, but in most practical computational problems it is unnecessary and inadvisable to compute A^{-1}. As succinctly expressed by G. E. Forsythe and C. B. Moler [27]:

Almost anything you can do with A^{-1} can be done without it!

The work required to compute A^{-1} is about n^3 flops, i.e., three times greater than for computing the LU factorization. (If A is a band matrix, then the savings can be much more spectacular; see Sec. 7.4.) To solve a linear system $Ax = b$ the matrix vector multiplication $A^{-1}b$ requires n^2 flops. This is exactly the same as for the solution of the two triangular systems $L(Ux) = b$ resulting from LU factorization of A. (Note, however, that on some parallel computers matrix multiplication can be performed much faster than solving triangular systems.)

One advantage of computing the inverse matrix is that A^{-1} can be used to get a strictly reliable error estimate for a computed solution $\bar{x}$. A similar estimate is not directly available from the LU factorization. However, alternative ways to obtain error estimates are the use of a condition estimator (Section 7.5.3) or iterative refinement (Section 7.6.4).

Not only is the inversion approach three times more expensive but if A is ill-conditioned the solution computed from $A^{-1}b$ usually is much less accurate than than that computed from the LU factorization. Using LU factorization the residual vector of the computed solution will usually be of order machine precision even when A is ill-conditioned.

Nevertheless, there are some applications where A^{-1} is required, e.g., in some methods for computing the matrix square root and the logarithm of a matrix; see Sec. 9.2.4. The inverse of a symmetric positive definite matrix is needed to obtain estimates of the covariances in regression analysis. However, usually only certain elements of A^{-1} are needed and not the whole inverse matrix.

We first consider computing the inverse of a lower triangular matrix L. Setting $L^{-1} = Y = (y_1, \ldots, y_n)$, we have $LY = I = (e_1, \ldots, e_n)$. This shows that the columns of Y satisfy

$$Ly_j = e_j, \quad j = 1 : n.$$

These lower triangular systems can be solved by forward substitution. Since the vector e_j has $(j-1)$ leading zeros the first $(j-1)$ components in y_j are zero. Hence L^{-1} is also a lower triangular matrix, and its elements can be computed recursively from

$$y_{jj} = 1/l_{jj}, \qquad y_{ij} = \Big(-\sum_{k=j}^{i-1} l_{ik} y_{kj} \Big) \Big/ l_{ii}, \quad i = j+1 : n, \tag{7.2.40}$$

Note that the diagonal elements in L^{-1} are just the inverses of the diagonal elements of L. If the columns are computed in the order $j = 1 : n$, then Y can overwrite L in storage.

Similarly, if U is upper triangular matrix then $Z = U^{-1}$ is an upper triangular matrix, whose elements can be computed from:

$$z_{jj} = 1/u_{jj}, \qquad z_{ij} = \Big(-\sum_{k=i+1}^{j} u_{ik} z_{kj} \Big) \Big/ u_{ii}, \quad i = j-1 : -1 : 1. \tag{7.2.41}$$

If the columns are computed in the order $j = n : -1 : 1$, the Z can overwrite U in storage. The number of flops required to compute L^{-1} or U^{-1} is approximately equal to $n^3/6$. Variants of the above algorithm can be obtained by reordering the loop indices.

Now let $A^{-1} = X = (x_1, \ldots, x_n)$ and assume that an LU factorization $A = LU$ has been computed. Then

$$Ax_j = L(Ux_j) = e_j, \quad j = 1 : n, \tag{7.2.42}$$

and the columns of A^{-1} are obtained by solving n linear systems, where the right hand sides equal the columns in the unit matrix. Setting (7.2.42) is equivalent to

$$Ux_j = y_j, \qquad Ly_j = e_j, \quad j = 1 : n. \tag{7.2.43}$$

This method for inverting A requires $n^3/6$ flops for inverting L and $n^3/2$ flops for solving the n upper triangular systems giving a total of n^3 flops.

A second method uses the relation

$$A^{-1} = (LU)^{-1} = U^{-1}L^{-1}. \tag{7.2.44}$$

Since the matrix multiplication $U^{-1}L^{-1}$ requires $n^3/3$ flops (show this!) the total work to compute A^{-1} by the second method method (7.2.44) also is n^3 flops. If we take advantage of that $y_{jj} = 1/l_{jj} = 1$, and carefully sequence the computations then L^{-1}, U^{-1} and finally A^{-1} can overwrite A so that no extra storage is needed.

There are many other varaints of computing the inverse $X = A^{-1}$. From $XA = I$ we have

$$XLU = I \quad \text{or} \quad XL = U^{-1}.$$

In the MATLAB function inv(A), U^{-1} is first computed by a column oriented algorithm. Then the system $XL = U^{-1}$ is solved for X. The stability properties of this and several other different matrix inversion algorithms are analyzed in [21]; see also Higham [41, Sec. 14.2].

The inverse can also be obtained from the Gauss–Jordan factorization. Using (7.2.19) where b is taken to be the columns of the unit matrix, we compute

$$A^{-1} = D^{-1}M_{n-1}^{-1}P_{n-1}\cdots M_2^{-1}P_2M_1^{-1}P_1(e_1,\ldots,e_n).$$

Again n^3 flops are required if the computations are properly organized. The method can be arranged so that the inverse emerges in the original array. However, the numerical properties of this method are not as good as for the methods described above.

If row interchanges have been performed during the LU factorization, we have $PA = LU$, where $P = P_{n-1}\cdots P_2P_1$ and P_k are transposition matrices. Then $A^{-1} = (LU)^{-1}P$. Hence we obtain A^{-1} by performing the interchanges in *reverse order on the columns* of $(LU)^{-1}$.

An approximative inverse of a matrix $A = I - B$ can sometimes be computed from a matrix series expansion. To derive this we form the product

$$(I-B)(I+B+B^2+B^3+\cdots+B^k) = I - B^{k+1}.$$

Suppose that $\|B\| < 1$ for some matrix norm. Then it follows that

$$\|B^{k+1}\| \le \|B\|^{k+1} \to 0, \quad k \to \infty,$$

and hence the **Neumann expansion**

$$(I-B)^{-1} = I + B + B^2 + B^3 + \cdots, \tag{7.2.45}$$

converges to $(I-B)^{-1}$. (Note the similarity with the Maclaurin series for $(1-x)^{-1}$.) Alternatively one can use the more rapidly converging **Euler expansion**

$$(I-B)^{-1} = (I+B)(I+B^2)(I+B^4)\cdots. \tag{7.2.46}$$

It can be shown by induction that

$$(I+B)(I+B^2)\cdots(I+B^{2^k}) = I + B + B^2 + B^3 + \cdots B^{2^{k+1}}.$$

Finally we mention an iterative method for computing the inverse, the **Newton–Schultz iteration**

$$X_{k+1} = X_k(2I - AX_k) = (2I - AX_k)X_k. \tag{7.2.47}$$

This is an analogue to the iteration $x_{k+1} = x_k(2 - ax_k)$, for computing the inverse of a scalar. It can be shown that if $X_0 = \alpha_0 A^T$ and $0 < \alpha_0 < 2/\|A\|_2^2$, then $\lim_{k\to\infty} X_k = A^{-1}$. Convergence can be slow initially but ultimately quadratic,

$$E_{k+1} = E_k^2, \quad E_k = I - AX_k \ \text{ or } \ I - X_kA.$$

Since about $2\log_2 \kappa_2(A)$ (see [59]) iterations are needed for convergence it cannot in general compete with direct methods for dense matrices. However, a few steps of the iteration (7.2.47) can be used to improve an approximate inverse.

Review Questions

1. How many operations are needed (approximately) for
   (a) The LU factorization of a square matrix?
   (b) The solution of $Ax = b$, when the triangular factorization of A is known?
2. Show that if the kth diagonal entry of an upper triangular matrix is zero, then its first k columns are linearly dependent.
3. What is meant by partial and complete pivoting in Gaussian elimination? Mention two classes of matrices for which Gaussian elimination can be performed stably without any pivoting?
4. What is the LU-decomposition of an n by n matrix A, and how is it related to Gaussian elimination? Does it always exist? If not, give sufficient conditions for its existence.
5. How is the LU-decomposition used for solving a linear system? What are the advantages over using the inverse of A? Give an approximate operation count for the solution of a dense linear system with p different right hand sides using the LU-decomposition.
6. Let B be a strictly lower or upper triangular matrix. Prove that the Neumann and Euler expansions for $(I - L)^{-1}$ are finite.

Problems

1. (a) Compute the LU factorization of A and $\det(A)$, where

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{pmatrix}.$$

   (b) Solve the linear system $Ax = b$, where $b = (2, 10, 44, 190)^T$.
2. (a) Show that $P = (e_n, \ldots, e_2, e_1)$ is a permutation matrix and that $P = P^T = P^{-1}$, and that Px reverses the order of the elements in the vector x.
   (b) Let the matrix A have an LU factorization. Show that there is a related factorization $PAP = UL$, where U is upper triangular and L lower triangular.
3. In Algorithm 7.2.2 for Gaussian elimination the elements in A are assessed in row-wise order in the innermost loop over j. If implemented in Fortran this algorithm may be inefficient since this language stores two-dimensional arrays by columns. Modify Algorithm 7.2.2 so that the innermost loop instead involves a fixed column index and a varying row index.
4. What does M_j^{-1}, where M_j is defined in (7.2.17), look like?

5. Compute the inverse matrix A^{-1}, where

$$A = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 1 & 2 \end{pmatrix},$$

(a) By solving $AX = I$, using Gaussian elimination with partial pivoting.
(b) By LU factorization and using $A^{-1} = U^{-1}L^{-1}$.

7.3 Symmetric Matrices

7.3.1 Symmetric Positive Definite Matrices

Gaussian elimination can be adopted to several classes of matrices of special structure. As mentioned in Sec. sec7.2.5, one case when Gaussian elimination can be performed stably without any pivoting is when A is Hermitian or real symmetric and positive definite. Solving such systems is one of the most important problems in scientific computing.

Definition 7.3.1.
A matrix $A \in \mathbf{C}^{n\times n}$ is called **Hermitian** *if $A = A^H$, the conjugate transpose of A. If A is Hermitian, then the quadratic form $(x^HAx)^H = x^HAx$ is real and A is said to be* **positive definite** *if*

$$x^HAx > 0, \quad \forall x \in \mathbf{C}^n, \quad x \neq 0, \tag{7.3.1}$$

and **positive semidefinite** *if $x^TAx \geq 0$, for all $x \in \mathbf{R}^n$. Otherwise it is called* **indefinite**.

It is well known that all eigenvalues of an Hermitian matrix are real. An equivalent condition for an Hermitian matrix to be positive definite is that all its eigenvalues are positive

$$\lambda_k(A) > 0, \quad k = 1 : n.$$

Since this condition can be difficult to verify the following sufficient condition is useful. A Hermitian matrix A, which has positive diagonal elements and is diagonally dominant

$$a_{ii} > \sum_{j\neq i} |a_{ij}|, \quad i = 1 : n,$$

can be shown to be positive definite, since it follows from Gerschgorin's Theorem (Theorem 9.3.1) that the eigenvalues of A are all positive.

Clearly a positive definite matrix is nonsingular, since if it were singular there should be a null vector $x \neq 0$ such that $Ax = 0$ and then $x^HAx = 0$. Positive definite (semidefinite) matrices have the following important property:

Theorem 7.3.2. *Let $A \in \mathbf{C}^{n\times n}$ be positive definite and let $X \in \mathbf{C}^{n\times p}$ have full column rank. Then X^HAX is positive definite (semidefinite). In particular any*

principal $p \times p$ submatrix

$$\tilde{A} = \begin{pmatrix} a_{i_1 i_1} & \dots & a_{i_1 i_p} \\ \vdots & & \vdots \\ a_{i_p i_1} & \dots & a_{i_p i_p} \end{pmatrix} \in \mathbf{C}^{p\times p}, \quad 1 \le p < n,$$

is positive definite definite (semidefinite). In particular, taking $p = 1$, all diagonal elements in A are real positive (nonnegative).

Proof. Let $x \neq 0$ and let $y = Xx$. Then since X is of full column rank $y \neq 0$ and $x^H(X^HAX)x = y^HAy > 0$ by the positive definiteness of A. In particular any principal submatrix of A can be written as X^HAX, where the columns of X are taken as the columns $k = i_j$, $j = 1, \dots, p$ of the identity matrix. The case when A is positive semidefinite follows similarly. □

A Hermitian or symmetric matrix A of oder n has only $\frac{1}{2}n(n+1)$ independent elements. If A also is positive definite then symmetry can be preserved in Gaussian elimination and the number of operations and storage needed can be reduced by half. Indeed Gauss's derived his original algorithm for the symmetric positive definite systems coming from least squares problems (see Chapter 8). We consider below the special case when A is real and symmetric but all results are easily generalized to the complex Hermitian case.

Lemma 7.3.3. *Let A be a real symmetric matrix. Then if Gaussian elimination can be carried trough without pivoting the reduced matrices*

$$A = A^{(1)}, A^{(2)}, \dots, A^{(n)}$$

are all symmetric.

Proof. Assume that $A^{(k)}$ is symmetric, for some k, where $1 \le k < n$. Then by Algorithm 7.2.2 we have after the k-th elimination step

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}a_{kj}^{(k)} = a_{ji}^{(k)} - \frac{a_{jk}^{(k)}}{a_{kk}^{(k)}}a_{ki}^{(k)} = a_{ji}^{(k+1)},$$

$k+1 \le i, j \le n$. This shows that $A^{(k+1)}$ *is also a symmetric matrix*, and the result follows by induction. □

A more general result is the following. Partition the Hermitian positive definite matrix A as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^H & A_{22} \end{pmatrix}$$

where A_{11} is a square matrix, Then by Theorem 7.3.2 both A_{11} and A_{22} are Hermitian positive definite and therefore nonsingular. It follows that the Schur complement of A_{11} in A, which is

$$S = A_{22} - A_{12}^H A_{11}^{-1} A_{12}$$

exists and is Hermitian. Moreover, for $x \neq 0$, we have

$$x^H(A_{22} - A_{12}^H A_{11}^{-1} A_{12})x = \begin{pmatrix} y^H & -x^H \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^H & A_{22} \end{pmatrix} \begin{pmatrix} y \\ -x \end{pmatrix} > 0$$

where $y = A_{11}^{-1} A_{12} x$, it follows that S is positive definite.

From Lemma 7.3.3 it follows that in Gaussian elimination without pivoting only the elements in $A^{(k)}$, $k = 2 : n$, on and below the main diagonal have to be computed. Since any diagonal element can be brought in pivotal position by a symmetric row and column interchange, the same conclusion holds if pivots are chosen arbitrarily along the diagonal.

Assume that the lower triangular part of the symmetric matrix A is given. The following algorithm computes, *if it can be carried through*, a *unit lower triangular* matrix $L = (l_{ik})$, and a diagonal matrix $D = \operatorname{diag}(d_1, \ldots, d_n)$ such that

$$A = LDL^T. \tag{7.3.2}$$

Algorithm 7.3.1 Symmetric Gaussian Elimination.

for $k = 1 : n-1$
  $d_k := a_{kk}^{(k)}$;
  for $i = k+1 : n$
    $l_{ik} := a_{ik}^{(k)} / d_k$;
    for $j = k+1 : i$
      $a_{ij}^{(k+1)} := a_{ij}^{(k)} - l_{ik} d_k l_{jk}$;
    end
  end
end

In the last line we have substituted $d_k l_{jk}$ for $a_{jk}^{(k)}$.

Note that the elements in L and D can overwrite the elements in the lower triangular part of A, so also the storage requirement is halved to $n(n+1)/2$. The uniqueness of the LDL^T factorization follows trivially from the uniqueness of the LU factorization.

Using the factorization $A = LDL^T$ the linear system $Ax = b$ decomposes into the two triangular systems

$$Ly = b, \qquad L^T x = D^{-1} y. \tag{7.3.3}$$

The cost of solving these triangular systems is about n^2 flams.

Example 7.3.1.

It may not always be possible to perform Gaussian elimination on a symmetric matrix, using pivots chosen from the diagonal. Consider, for example, the

nonsingular symmetric matrix

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

If we take $\epsilon = 0$, then both diagonal elements are zero, and symmetric Gaussian elimination breaks down. If $\epsilon \neq 0$, but $|\epsilon| \ll 1$, then choosing ϵ as pivot will not be stable. On the other hand, a row interchange will in general destroy symmetry!

We will prove that Gaussian elimination without pivoting can be carried out with positive pivot elements if and only if A is real and symmetric positive definite. (The same result applies to complex Hermitian matrices, but since the modifications necessary for this case are straightforward, we discuss here only the real case.) For symmetric semidefinite matrices symmetric pivoting can be used. The *indefinite* case requires more substantial modifications, which will be discussed in Section 7.3.4.

Theorem 7.3.4.

The symmetric matrix $A \in \mathbf{R}^{n\times n}$ is positive definite if and only if there exists a unit lower triangular matrix L and a diagonal matrix D with positive elements such that

$$A = LDL^T, \quad D = \operatorname{diag}(d_1, \ldots, d_n),$$

Proof. Assume first that we are given a symmetric matrix A, for which Algorithm 7.3.1 yields a factorization $A = LDL^T$ with positive pivotal elements $d_k > 0$, $k = 1 : n$. Then for all $x \neq 0$ we have $y = L^T x \neq 0$ and

$$x^T Ax = x^T LDL^T x = y^T Dy > 0.$$

It follows that A is positive definite.

The proof of the other part of the theorem is by induction on the order n of A. The result is trivial if $n = 1$, since then $D = d_1 = A = a_{11} > 0$ and $L = 1$. Now write

$$A = \begin{pmatrix} a_{11} & a^T \\ a & \tilde{A} \end{pmatrix} = L_1 D_1 L_1^T, \quad L_1 = \begin{pmatrix} 1 & 0 \\ d_1^{-1}a & I \end{pmatrix}, \quad D_1 = \begin{pmatrix} d_1 & 0 \\ 0 & B \end{pmatrix},$$

where $d_1 = a_{11}$, $B = \tilde{A} - d_1^{-1}aa^T$. Since A is positive definite it follows that D_1 is positive definite, and therefore $d_1 > 0$, and B is positive definite. Since B is of order $(n-1)$, by the induction hypothesis there exists a unique unit lower triangular matrix $\tilde{L}$ and diagonal matrix $\tilde{D}$ with positive elements such that $B = \tilde{L}\tilde{D}\tilde{L}^T$. Then it holds that $A = LDL^T$, where

$$L = \begin{pmatrix} 1 & 0 \\ d_1^{-1}a & \tilde{L} \end{pmatrix}, \qquad D = \begin{pmatrix} d_1 & 0 \\ 0 & \tilde{D} \end{pmatrix}.$$

□

Example 7.3.2. The Hilbert matrix $H_n \in \mathbf{R}^{n\times n}$ with elements

$$h_{ij} = 1/(i+j-1), \quad 1 \le i,j \le n,$$

is positive definite. Hence, if Gaussian elimination without pivoting is carried out then the pivotal elements are all positive. For example, for $n = 4$, symmetric Gaussian elimination yields the $H_4 = LDL^T$, where

$$D = \text{diag}\,(1, 1/12, 1/180, 1/2800), \qquad L = \begin{pmatrix} 1 & & & \\ 1/2 & 1 & & \\ 1/3 & 1 & 1 & \\ 1/4 & 9/10 & 3/2 & 1 \end{pmatrix}.$$

Theorem 7.3.4 also yields the following useful characterization of a positive definite matrix.

Theorem 7.3.5. Sylvester's Criterion
A symmetric matrix $A \in \mathbf{R}^{n\times n}$ is positive definite if and only if

$$\det(A_k) > 0, \quad k = 1, 2, \ldots, n,$$

where $A_k \in \mathbf{R}^{k\times k}$, $k = 1, 2 : n$, are the leading principal submatrices of A.

Proof. If symmetric Gaussian elimination is carried out without pivoting then

$$\det(A_k) = d_1 d_2 \cdots d_k.$$

Hence $\det(A_k) > 0$, $k = 1 : n$, if and only if all pivots are positive. However, by Theorem 7.3.2 this is the case if and only if A is positive definite. □

In prove a bound on the growth ratio for the symmetric positive definite we first show the following

Lemma 7.3.6. *For a symmetric positive definite matrix $A = (a_{ij}) \in \mathbf{R}^{n\times n}$ the maximum element of A lies on the diagonal.*

Proof. Theorem 7.3.2 and Sylvester's criterion imply that

$$0 < \det \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix} = a_{ii}a_{jj} - a_{ij}^2, \quad 1 \le i,j \le n.$$

Hence

$$|a_{ij}|^2 < a_{ii}a_{jj} \le \max_{1\le i\le n} a_{ii}^2,$$

from which the lemma follows. □

Theorem 7.3.7.

Let A be symmetric and positive definite. Then Gaussian elimination without pivoting is backward stable and the growth ratio satisfies $\rho_n \leq 1$.

Proof. In Algorithm 7.3.1 the diagonal elements are transformed in the k:th step of Gaussian elimination according to

$$a_{ii}^{(k+1)} = a_{ii}^{(k)} - (a_{ki}^{(k)})^2/a_{kk}^{(k)} = a_{ii}^{(k)}\Big(1 - (a_{ki}^{(k)})^2/\big(a_{ii}^{(k)}a_{kk}^{(k)}\big)\Big).$$

If A is positive definite so are $A^{(k)}$ and $A^{(k+1)}$. Using Lemma 7.3.6 it follows that $0 < a_{ii}^{(k+1)} \leq a_{ii}^{(k)}$, and hence the diagonal elements in the successive reduced matrices cannot increase. Thus we have

$$\max_{i,j,k} |a_{ij}^{(k)}| = \max_{i,k} a_{ii}^{(k)} \leq \max_{i} a_{ii} = \max_{i,j} |a_{ij}|,$$

which implies that $\rho_n \leq 1$. ∎

Any matrix $A \in \mathbf{R}^{n\times n}$ can be written as the sum of a symmetric and a skew-symmetric part, $A = H + S$, where

$$A_H = \tfrac{1}{2}(A + A^T), \qquad A_S = \tfrac{1}{2}(A - A^T). \tag{7.3.4}$$

A is symmetric if and only if $A_S = 0$. Sometimes A is called positive definite if its symmetric part A_H is positive definite. If the matrix A has a positive symmetric part then its leading principal submatrices are nonsingular and Gaussian elimination can be carried out to completion without pivoting. However, the resulting LU factorizing may not be stable as shown by the example

$$\begin{pmatrix} \epsilon & 1 \\ -1 & \epsilon \end{pmatrix} = \begin{pmatrix} 1 & \\ -1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ & \epsilon + 1/\epsilon \end{pmatrix}, \quad (\epsilon > 0).$$

These result can be extended to complex matrices with positive definite Hermitian part $A_H = \frac{1}{2}(A + A^H)$, for which its holds that $x^H Ax > 0$, for all nonzero $x \in \mathbf{C}^n$. Of particular interest are complex symmetric matrices, arising in computational electrodynamics, of the form

$$A = B + iC, \quad B, C \in \mathbf{R}^{n\times n}, \tag{7.3.5}$$

where $B = A_H$ and $C = A_S$ both are symmetric positive definite. It can be shown that for this class of matrices $\rho_n < 3$, so LU factorization without pivoting is stable (see [30]).

7.3.2 Cholesky Factorization

Let A be a symmetric positive definite matrix A. Then the $\mathrm{LDL}^{\mathrm{T}}$ factorization (7.3.2) exists and $D > 0$. Hence we can write

$$A = LDL^T = (LD^{1/2})(LD^{1/2})^T, \quad D^{1/2} = \operatorname{diag}(\sqrt{d}_1, \ldots, \sqrt{d}_n). \tag{7.3.6}$$

Defining the upper triangular matrix $R := D^{1/2}L^T$ we obtain the factorization

$$A = R^T R. \tag{7.3.7}$$

If we here take the diagonal elements of L to be positive it follows from the uniqueness of the $\mathrm{LDL}^{\mathrm{T}}$ factorization that this factorization is unique. The factorization (7.3.7) is called the **Cholesky factorization** of A. and R is called the **Cholesky factor** of A.[16]

The Cholesky factorization is obtained if in symmetric Gaussian elimination (Algorithm 7.3.1) we set $d_k = l_{kk} = (a_{kk}^{(k)})^{1/2}$. This gives the outer product version of Cholesky factorization in which in the kth step, the reduced matrix is modified by a rank-one matrix

$$A^{(k+1)} = A^{(k)} - l_k l_k^T,$$

where l_k denotes the column vector of multipliers.

In analogy to the compact schemes for LU factorization (see Section 7.2.6) it is possible to arrange the computations so that the elements in the Cholesky factor $R = (r_{ij})$ are determined directly. The matrix equation $A = R^T R$ with R upper triangular can be written

$$a_{ij} = \sum_{k=1}^{i} r_{ki} r_{kj} = \sum_{k=1}^{i-1} r_{ki} r_{kj} + r_{ii} r_{ij}, \quad 1 \le i \le j \le n. \tag{7.3.8}$$

This is $n(n+1)/2$ equations for the unknown elements in R. We remark that for $i = j$ this gives

$$\max_i r_{ij}^2 \le \sum_{k=1}^{j} r_{kj}^2 = a_j \le \max_i a_{ii},$$

which shows that the elements in R are bounded maximum diagonal element in A. Solving for r_{ij} from the corresponding equation in (7.3.8), we obtain

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii}, \quad i < j, \qquad r_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\Big)^{1/2}.$$

If properly sequenced, these equations can be used in a recursive fashion to compute the elements in R. For example the elements in R can be determined one row or one column at a time.

Algorithm 7.3.2 Cholesky Algorithm; column-wise order

for $j = 1 : n$
  for $i = 1 : j - 1$

[16] André-Louis Cholesky (1875–1918) was a French military officer involved in geodesy and surveying in Crete and North Africa just before World War I. He developed the algorithm named after him and his work was posthumously published by a fellow officer, Benoit in 1924.

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii};$$

end

$$r_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\Big)^{1/2};$$

end

The column-wise ordering has the advantage of giving the Cholesky factors of all leading principal submatrices of A. An algorithm which computes the elements of R in row-wise order is obtained by reversing the two loops in the code above.

Algorithm 7.3.3 Cholesky Algorithm; row-wise order.

for $i = 1 : n$

$$r_{ii} = \Big(a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2\Big)^{1/2};$$

for $j = i + 1 : n$

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii};$$

end

end

These two versions of the Cholesky algorithm are not only mathematically equivalent but also *numerically equivalent*, i.e., they will compute the same Cholesky factor, taking rounding errors into account. In the Cholesky factorization only

$$\begin{pmatrix} 1 & 2 & 4 & 7 & 11 \\ & 3 & 5 & 8 & 12 \\ & & 6 & 9 & 13 \\ & & & 10 & 14 \\ & & & & 15 \end{pmatrix}$$

Figure 7.3.1. *The mapping of array-subscript of an upper triangular matrix of order 5.*

elements in the upper triangular part of A are referenced and only these elements need to be stored. Since most programming languages only support rectangular arrays this means that the lower triangular part of the array holding A is not used. One possibility is then to use the lower half of the array to store R^T and not overwrite the original data. Another option is to store the elements of the upper triangular part of A column-wise in a vector, see Fig. 7.3.1. which is known as **packed storage**. This data is then and overwritten by the elements of R during the computations. Using packed storage complicates somewhat index computations but is useful when economizing storage is worthwhile.

Some applications lead to a linear systems where $A \in \mathbf{R}^{n\times n}$ is a symmetric positive *semidefinite* matrix ($x^T Ax \geq 0\ \forall x \neq 0$) with $\text{rank}(A) = r < n$. One example is rank deficient least squares problems; see Section 8.5. Another example is when the finite element method is applied to a problem where rigid body motion occurs, which implies $r \leq n-1$. In the semidefinite case a Cholesky factorization still exists, but symmetric pivoting needs to be incorporated. In the kth elimination step *a maximal diagonal element* $a_{ss}^{(k)}$ in the reduced matrix $A^{(k)}$ is chosen as pivot, i.e.,

$$a_{ss}^{(k)} = \max_{k\leq i\leq n} a_{ii}^{(k)}. \tag{7.3.9}$$

This pivoting strategy is easily implemented in Algorithm 7.3.1, the outer product version. Symmetric pivoting is also beneficial when A is close to a rank deficient matrix.

Since all reduced matrices are positive semidefinite their largest element lies on the diagonal. Hence diagonal pivoting is equivalent to complete pivoting in Gaussian elimination. In exact computation the Cholesky algorithm stops when all diagonal elements in the reduced matrix are zero. This implies that the reduced matrix is the zero matrix.

If A has rank $r < n$ the resulting Cholesky factorization has the upper trapezoidal form

$$P^T AP = R^T R, \qquad R = \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \tag{7.3.10}$$

where P is a permutation matrix and $R_{11} \in \mathbf{R}^{r\times r}$ with positive diagonal elements. The linear system $Ax = b$, or $P^T AP(P^T x) = P^T b$, then becomes

$$R^T R\tilde{x} = \tilde{b}, \quad \tilde{x} = P^T x, \quad \tilde{b} = P^T b.$$

Setting $z = R\tilde{x}$ the linear system reads

$$R^T z = \begin{pmatrix} R_{11}^T \\ R_{12}^T \end{pmatrix} z = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix},$$

and from the first r equations we obtain $z = R_{11}^{-T}\tilde{b}_1$. Substituting this in the last $n-r$ equations we get

$$0 = R_{12}^T z - \tilde{b}_2 = \begin{pmatrix} R_{12}^T R_{11}^{-T} & -I \end{pmatrix} \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}.$$

These equations are equivalent to $b \perp \mathcal{N}(A)$ and express the condition for the linear system $Ax = b$ to be consistent. If they are not satisfied a solution does not exist. It remains to solve $L^T\tilde{x} = z$, which gives

$$R_{11}\tilde{x}_1 = z - R_{12}\tilde{x}_2.$$

For an arbitrarily chosen $\tilde{x}_2$ we can uniquely determine $\tilde{x}_1$ so that these equations are satisfied. This expresses the fact that a consistent singular system has an infinite number of solutions. Finally the permutations are undone to obtain $x = P\tilde{x}$.

Rounding errors can cause negative elements to appear on the diagonal in the Cholesky algorithm even when A is positive semidefinite. Similarly, because of rounding errors the reduced matrix will in general be nonzero after r steps even when $\operatorname{rank}(A) = r$. The question arises when to terminate the Cholesky factorization of a semidefinite matrix. One possibility is to continue until

$$\max_{k \le i \le n} a_{ii}^{(k)} \le 0,$$

but this may cause unnecessary work in eliminating negligible elements. Further discussion of this aspect is postponed until Chapter 8.

7.3.3 Inertia of Symmetric Matrices

Let $A \in \mathbf{C}^{n\times n}$ be an Hermitian matrix. The **inertia** of A is defined as the number triple $\operatorname{in}(A) = (\pi, \nu, \delta)$ of positive, negative, and zero eigenvalues of A. If A is positive definite matrix and $Ax = \lambda x$, we have

$$x^H A x = \lambda x^H x > 0.$$

Hence all eigenvalues must be positive and the inertia is $(n, 0, 0)$.

Hermitian matrices arise naturally in the study of quadratic forms $\psi(x) = x^H A x$. By the coordinate transformation $x = Ty$ this quadratic form is transformed into

$$\psi(Ty) = y^H \hat{A} y, \quad \hat{A} = T^H A T.$$

The mapping of A onto $T^H A T$ is called a **congruence transformation** of A, and we say that A and $\hat{A}$ are **congruent**. (Notice that a congruence transformation with a nonsingular matrix means a transformation to a coordinate system which is usually not rectangular.) Unless T is unitary these transformations do not, in general, preserve eigenvalues. However, Sylvester's famous law of inertia says that the *signs of eigenvalues are preserved by congruence transformations.*

Theorem 7.3.8. Sylvester's Law of Inertia *If $A \in \mathbf{C}^{n\times n}$ is symmetric and $T \in \mathbf{C}^{n\times n}$ is nonsingular then A and $\hat{A} = T^H A T$ have the same inertia.*

Proof. Since A and $\hat{A}$ are Hermitian there exist unitary matrices U and $\hat{U}$ such that

$$U^H A U = D, \qquad \hat{U}^H \hat{A} \hat{U} = \hat{D},$$

where $D = \operatorname{diag}(\lambda_i)$ and $\hat{D} = \operatorname{diag}(\hat{\lambda}_i)$ are diagonal matrices of eigenvalues. By definition we have $\operatorname{in}(A) = \operatorname{in}(D)$, $\operatorname{in}(\hat{A}) = \operatorname{in}(\hat{D})$, and hence, we want to prove that $\operatorname{in}(D) = \operatorname{in}(\hat{D})$, where

$$\hat{D} = S^H D S, \quad S = U^H T \hat{U}.$$

Assume that $\pi \neq \hat{\pi}$, say $\pi > \hat{\pi}$, and that the eigenvalues are ordered so that $\lambda_j > 0$ for $j \le \pi$ and $\hat{\lambda}_j > 0$ for $j \le \hat{\pi}$. Let $x = S\hat{x}$ and consider the quadratic form $\psi(x) = x^H D x = \hat{x}^H \hat{D} \hat{x}$, or

$$\psi(x) = \sum_{j=1}^{n} \lambda_j |\xi_j|^2 = \sum_{j=1}^{n} \hat{\lambda}_j |\hat{\xi}_j|^2.$$

Let $x^* \neq 0$ be a solution to the $n - \pi + \hat{\pi} < n$ homogeneous linear relations

$$\xi_j = 0, \quad j > \pi, \qquad \hat{\xi}_j = (S^{-1}x)_j = 0, \quad j \leq \hat{\pi}.$$

Then

$$\psi(x^*) = \sum_{j=1}^{\pi} \lambda_j |\xi_j^*|^2 > 0, \quad \psi(x^*) = \sum_{j=\hat{\pi}}^{n} \hat{\lambda}_j |\hat{\xi}_j^*|^2 \leq 0.$$

This is a contradiction and hence the assumption that $\pi \neq \hat{\pi}$ is false, so A and $\hat{A}$ have the same number of positive eigenvalues. Using the same argument on $-A$ it follows that also $\nu = \hat{\nu}$, and since the number of eigenvalues is the same $\delta = \hat{\delta}$. □

Let $A \in \mathbf{R}^{n\times n}$ be a real symmetric matrix and consider the quadratic equation

$$x^T Ax - 2bx = c, \quad A \neq 0. \tag{7.3.11}$$

The solution sets of this equation are sometimes called **conical sections**. If $b = 0$, then the surface has its center at the origin reads $x^T Ax = c$. The inertia of A completely determines the geometric type of the conical section.

Sylvester's theorem tells that the geometric type of the surface can be determined without computing the eigenvalues? Since we can always multiply the equation by -1 we can assume that there are at least one positive eigenvalues. Then, for $n = 2$ there are three possibilities:

$$(2, 0, 0) \text{ ellipse}; \quad (1, 0, 1) \text{ parabola}; \quad (1, 1, 0) \text{ hyperbola}.$$

In n dimensions there will be $n(n+1)/2$ cases, assuming that at least one eigenvalue is positive.

7.3.4 Symmetric Indefinite Matrices

As shown by Example 7.3.1, the LDL^T factorization of a symmetric indefinite matrix, although efficient computationally, may not exist and can be unstable. This is true even when symmetric row and column interchanges are used, to select at each stage the largest diagonal element in the reduced matrix as pivot. One stable way of factorizing an indefinite matrix is of course to compute an unsymmetric LU factorization using Gaussian elimination with partial pivoting. However, this factorization does not give the inertia of A and we give up the savings of a factor one half in d storage.

The following example shows that in order to enable a stable $\mathrm{LDL}^{\mathrm{T}}$ factorization for a symmetric *indefinite* matrix A, it is necessary to consider a block factorization where D is block diagonal with also 2×2 diagonal blocks..

Example 7.3.3.

The symmetric matrix

$$A = \begin{pmatrix} \epsilon & 1 \\ 1 & \epsilon \end{pmatrix}, \quad 0 < \epsilon \ll 1,$$

is indefinite since $\det(A) = \lambda_1\lambda_2 = \epsilon^1 - 1 < 0$. If we compute the $\mathrm{LDL}^{\mathrm{T}}$ factorization of A without pivoting we obtain

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

which shows that there is unbounded element growth. However, A is well conditioned with inverse

$$A^{-1} = \frac{1}{\epsilon^2 - 1} \begin{pmatrix} \epsilon & 1 \\ 1 & \epsilon \end{pmatrix}, \quad 0 < \epsilon \ll 1.$$

It is quite straightforward to generalize Gaussian elimination to use any nonsingular 2×2 principal submatrix as pivot. By a symmetric permutation this submatrix is brought to the upper left corner, and the permuted matrix partitioned as

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix}, \quad A_{11} = \begin{pmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{pmatrix}.$$

Then the Schur complement of A_{11}, $S = A_{22} - A_{12}^T A_{11}^{-1} A_{12}$, exists where

$$A_{11}^{-1} = \frac{1}{\delta_{12}} \begin{pmatrix} a_{22} & -a_{21} \\ -a_{21} & a_{11} \end{pmatrix}, \quad \delta_{12} = \det(A_{11}) = a_{11}a_{22} - a_{21}^2. \tag{7.3.12}$$

We obtain the symmetric block factorization

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L & I \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & L^T \\ 0 & I \end{pmatrix}, \tag{7.3.13}$$

where $L = A_{12}^T A_{11}^{-1}$. This determines the first two columns of a unit lower triangular matrix $L = L_{21} = A_{21}A_{11}^{-1}$, in an $\mathrm{LDL}^{\mathrm{T}}$ factorization of A. The block A_{22} is transformed into the symmetric matrix $A_{22}^{(3)} = A_{22} - L_{21}A_{21}^T$ with components

$$a_{ij}^{(3)} = a_{ij} - l_{i1}a_{1j} - l_{i2}a_{2j}, \quad 2 \le j \le i \le n. \tag{7.3.14}$$

It can be shown that $A_{22}^{(3)}$ is the same reduced matrix as if two steps of Gaussian elimination were taken, first pivoting on the element a_{12} and then on a_{21}.

A similar reduction is used if 2×2 pivots are taken at a later stage in the factorization. Ultimately a factorization $A = LDL^T$ is computed in which D is block diagonal with in general a mixture of 1×1 and 2×2 blocks, and L is unit lower triangular with $l_{k+1,k} = 0$ when $A^{(k)}$ is reduced by a 2×2 pivot. Since the effect of taking a 2×2 step is to reduce A by the equivalent of *two* 1×1 pivot steps, the amount of work must be balanced against that. The part of the calculation which dominates the operation count is (7.3.14), and this is twice the work as for an 1×1 pivot. Therefore the leading term in the operations count is always $n^3/6$, whichever type of pivots is used.

The main issue then is to find a pivotal strategy that will give control of element growth without requiring too much search. One possible strategy is comparable to that of complete pivoting. Consider the first stage of the factorization and set

$$\mu_0 = \max_{ij} |a_{ij}| = |a_{pq}|, \qquad \mu_1 = \max_{i} |a_{ii}| = |a_{rr}|.$$

Then if

$$\mu_1/\mu_0 > \alpha = (\sqrt{17}+1)/8 \approx 0.6404,$$

the diagonal element a_{rr} is taken as an 1×1 pivot. Otherwise the 2×2 pivot.

$$\begin{pmatrix} a_{pp} & a_{qp} \\ a_{qp} & a_{qq} \end{pmatrix}, \quad p < q,$$

is chosen. In other words if there is a diagonal element not much smaller than the element of maximum magnitude this is taken as an 1×1 pivot. The magical number α has been chosen so as to minimize the bound on the growth per stage of elements of A, allowing for the fact that a 2×2 pivot is equivalent to two stages. The derivation, which is straight forward but tedious (see Higham [41, Sec. 11.1.1]) is omitted here.

With this choice the element growth can be shown to be bounded by

$$\rho_n \le (1 + 1/\rho)^{n-1} < (2.57)^{n-1}. \tag{7.3.15}$$

This exponential growth may seem alarming, but the important fact is that the reduced matrices cannot grow abruptly from step to step. No example is known where significant element growth occur at every step. The bound in (7.3.15) can be compared to the bound 2^{n-1}, which holds for Gaussian elimination with partial pivoting. The elements in L can be bounded by $1/(1-\alpha) < 2.781$ and this pivoting strategy therefore gives a backward stable factorization.

Since the complete pivoting strategy above requires the whole active submatrix to be searched in each stage, it requires $O(n^3)$ comparisons. The same bound for element growth (7.3.15) can be achieved using the following partial pivoting strategy due to **Bunch–Kaufman** [11]. For simplicity of notations we restrict our attention to the first stage of the elimination. All later stages proceed similarly. First determine the off-diagonal element of largest magnitude in the first column,

$$\lambda = |a_{r1}| = \max_{i \neq 1} |a_{i1}|.$$

If $|a_{11}| \ge \rho\lambda$, then take a_{11} as pivot. Else, determine the largest off-diagonal element in column r,

$$\sigma = \max_{1 \le i \le n} |a_{ir}|, \quad i \neq r.$$

If $|a_{11}| \ge \rho\lambda^2/\sigma$, then again take a_{11} as pivot, else if $|a_{rr}| \ge \rho\sigma$, take a_{rr} as pivot. Otherwise take as pivot the 2×2 principal submatrix

$$\begin{pmatrix} a_{11} & a_{1r} \\ a_{1r} & a_{rr} \end{pmatrix}.$$

Note that at most 2 columns need to be searched in each step, and at most $O(n^2)$ comparisons are needed in all.

Normwise backward stability can be shown to hold also for the Bunch–Kaufman partial pivoting strategy. However, it is no longer true that the elements of L are bounded independently of A. The following example (Higham [41, Sec. 11.1.2]) shows that for partial pivoting L is unbounded:

$$A = \begin{pmatrix} 0 & \epsilon & 0 \\ \epsilon & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ \epsilon^{-1} & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & \epsilon & \\ \epsilon & 0 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \epsilon^{-1} \\ & 1 & 0 \\ & & 1 \end{pmatrix}. \tag{7.3.16}$$

Note that whenever a 2×2 pivot is used, we have

$$a_{11}a_{rr} \le \rho^2 |a_{1r}|^2 < |a_{1r}|^2.$$

Hence with both pivoting strategies any 2×2 block in the block diagonal matrix D has a negative determinant $\delta_{1r} = a_{11}a_{rr} - a_{1r}^2 < 0$ and by Sylvester's Theorem corresponds to one positive and one negative eigenvalue. Hence a 2×2 pivot cannot occur if A is positive definite and in this case all pivots chosen by the Bunch–Kaufman strategy will be 1×1.

For solving a linear system $Ax = b$ the $\mathrm{LDL}^{\mathrm{T}}$ factorization produced by the Bunch–Kaufman pivoting strategy is satisfactory. For certain other applications the possibility of a large L factor is not acceptable. A bounded L factor can be achieved with the modified pivoting strategy suggested in [4]. This symmetric pivoting is roughly similar to rook pivoting and has a total cost of between $O(n^2)$ and $O(n^3)$ comparisons. Probabilistic results suggest that on the average the cost is only $O(n^2)$. In this strategy a search is performed until two indices r and s have been found such that the element a_{rs} bounds in modulus the other off-diagonal elements in the r and s columns (rows). Then either the 2×2 pivot D_{rs} or the largest in modulus of the two diagonal elements as an 1×1 pivot is taken, according to the test

$$\max(|a_{rr}|, |a_{ss}|) \ge \alpha |a_{rs}|.$$

Aasen [1] has given an algorithm that for a symmetric matrix $A \in \mathbf{R}^{n \times n}$ computes the factorization

$$PAP^T = LTL^T, \tag{7.3.17}$$

where L is unit lower triangular and T symmetric tridiagonal.

None of the algorithms described here preserves the band structure of the matrix A. In this case Gaussian elimination with partial pivoting can be used but as remarked before this will destroy symmetry and does not reveal the inertia. For the special case of a *tridiagonal* symmetric indefinite matrices an algorithm for computing an $\mathrm{LDL}^{\mathrm{T}}$ factorization will be given in Sec. 7.4.3.

A block $\mathrm{LDL}^{\mathrm{T}}$ factorization can also be computed for a real skew-symmetric matrix A. Note that $A^T = -A$ implies that such a matrix has zero diagonal elements. Further, since

$$(x^T Ax)^T = x^T A^T x = -x^T Ax,$$

it follows that all nonzero eigenvalues come in pure imaginary complex conjugate pairs. In the first step of the factorization if the first column is zero there is nothing to do. Otherwise we look for an off-diagonal element $a_{p,q}$, $p > q$ such that

$$|a_{p,q}| = \max\{\max_{1<i\le n} |a_{i,1}|, \max_{1<i\le n} |a_{i,2}|\},$$

and take the 2×2 pivot

$$\begin{pmatrix} 0 & -a_{p,q} \\ a_{p,q} & 0 \end{pmatrix}.$$

It can be shown that this pivoting the growth ratio is bounded by $\rho_n \le (\sqrt{3})^{n-2}$, which is smaller than for Gaussian elimination with partial pivoting for a general matrix.

Review Questions

1. (a) Give two necessary and sufficient conditions for a real symmetric matrix A to be positive definite.

(b) Show that if A is symmetric positive definite so is its inverse A^{-1}.

2. What simplifications occur in Gaussian elimination applied to a symmetric, positive definite matrix?

3. What is the relation of Cholesky factorization to Gaussian elimination? Give an example of a symmetric matrix A for which the Cholesky decomposition does not exist.

4. Show that if A is skew-symmetric, then iA is Hermitian.

5. Show that the Cholesky factorization is unique for positive definite matrices provided R is normalized to have positive diagonal entries.

6. (a) Formulate and prove Sylvester's law of inertia.

(b) Show that for $n = 3$ there are six different geometric types of conical sections $x^T Ax - 2b^T x = c$, provided that $A \neq 0$ and is normalized to have at least one positive eigenvalue.

Problems

1. If A is a symmetric positive definite matrix how should you compute $x^T Ax$ for a given vector x?

2. Show that if A is symmetric and positive definite then $|a_{ij}| \le (a_{ii} + a_{jj})/2$.

3. Show by computing the Cholesky factorization $A = LL^T$ that the matrix

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

is positive definite.

4. The Hilbert matrix $H_n \in \mathbf{R}^{n\times n}$ with elements

$$a_{ij} = 1/(i+j-1), \quad 1 \le i,j \le n,$$

is symmetric positive definite for all n. Denote by $\bar{H}_4$ the corresponding matrix with elements rounded to five decimal places, and compute its Cholesky factor $\bar{L}$. Then compute the difference $(\bar{L}\bar{L}^T - \bar{A})$ and compare it with $(A - \bar{A})$.

5. Let $A + iB$ be Hermitian and positive definite, where $A, B \in \mathbf{R}^{n\times n}$. Show that the real matrix

$$C = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

is symmetric and positive definite. How can a linear system $(A+iB)(x+iy) = b+ic$ be solved using a Cholesky factorization of C?

6. Implement the Cholesky factorization using packed storage for A and R.

7.4 Banded Linear Systems

7.4.1 Banded Matrices

Linear systems $Ax = b$ where the matrix A is banded arise in problems where each variable x_i is coupled by an equation only to a few other variables x_j such that $|j-i|$ is small. We make the following definition (note that it applies also to matrices which are not square):

We recall from Definition 7.1.1 that a matrix A is said to have upper bandwidth r and lower bandwidth s if

$$a_{ij} = 0, \quad j > i + r, \qquad a_{ij} = 0, \quad i > j + s,$$

respectively. This means that the number of non-zero diagonals above and below the main diagonal are r and s respectively. The maximum number of nonzero elements in any row is then $w = r + s + 1$, which is the **bandwidth** of A.

For a matrix $A \in \mathbf{R}^{m\times n}$ which is not square we define the bandwidth as

$$w = \max_{1\le i\le m} \{j - k + 1 \mid a_{ij}a_{ik} \neq 0\}.$$

Note that the bandwidth of a matrix depends on the ordering of its rows and columns. An important, but hard, problem is to find optimal orderings that minimize the bandwidth. However, there are good heuristic algorithms that can be used in practice and give almost optimal results; see Section 7.6.3.

It is convenient to introduce some additional notations for manipulating band matrices.[17]

[17] These notations are taken from MATLAB .

Definition 7.4.1.

If $a = (a_1, a_2, \ldots, a_{n-r})^T$ is a column vector with $n - r$ components then $A = \operatorname{diag}(a, k)$, $|k| < n$, denotes a square matrix of order n with the elements of a on its kth diagonal; $k = 0$ is the main diagonal; $k > 0$ is above the main diagonal; $k < 0$ is below the main diagonal.

If A is a square matrix of order n, then $\operatorname{diag}(A, k) \in \mathbf{R}^{(n-k)}$, $|k| < n$, is the column vector formed from the elements of the kth diagonal of A.

Assume that A and B are banded matrices of order n, which both have a small bandwidth compared to n. Then, since there are few nonzero elements in the rows and columns of A and B the usual algorithms for forming the product AB are not effective on vector computers. We now give an algorithm for multiplying matrices by diagonals, which overcomes this drawback.

Lemma 7.4.2.

Let $A = \operatorname{diag}(a, r)$ and $B = \operatorname{diag}(b, s)$ and set $C = AB$. If $|r + s| \geq n$ then $C = 0$; otherwise $C = \operatorname{diag}(c, r+s)$, where the elements of the vector $c \in \mathbf{R}^{(n-|r+s|)}$ are obtained by pointwise multiplication of shifted vectors a and b:

$$c = \begin{cases} (a_1 b_{r+1}, \ldots, a_{n-r-s} b_{n-s})^T, & \text{if } r, s \geq 0, \\ (a_{|s|+1} b_1, \ldots, a_{n-|r|} b_{n-|r+s|})^T, & \text{if } r, s \leq 0. \\ (0, \ldots, 0, a_1 b_1, \ldots, a_{n-s} b_{n-s})^T, & \text{if } r < 0, \quad s > 0, \quad r + s \geq 0. \\ (0, \ldots, 0, a_1 b_1, \ldots, a_{n-|r|} b_{n-|r|})^T, & \text{if } r < 0, \quad s > 0, \quad r + s < 0. \\ (a_1 b_{|r+s|+1}, \ldots, a_{n-r} b_{n-|s|}, 0, \ldots, 0)^T, & \text{if } r > 0, \quad s < 0, \quad r + s \geq 0. \\ (a_{r+1} b_1, \ldots, a_{n-r} b_{n-|s|}, 0, \ldots, 0)^T, & \text{if } r > 0, \quad s < 0, \quad r + s < 0. \end{cases} \tag{7.4.1}$$

Note that when $rs < 0$, zeros are added at the beginning or end to get a vector c of length $n - |r + s|$.

The number of cases in this lemma looks a bit forbidding, so to clarify the situation a bit more we consider a specific case.

Example 7.4.1.

Let A and B be tridiagonal matrices of size 5×5

$$A = \begin{pmatrix} a_1 & c_1 & & & \\ b_1 & a_2 & c_2 & & \\ & b_2 & a_3 & c_3 & \\ & & b_3 & a_4 & c_4 \\ & & & b_4 & a_5 \end{pmatrix}, \qquad B = \begin{pmatrix} d_1 & f_1 & & & \\ e_1 & d_2 & f_2 & & \\ & e_2 & d_3 & f_3 & \\ & & e_3 & d_4 & f_4 \\ & & & e_4 & d_5 \end{pmatrix}.$$

Then $C = AB$ will be a banded matrix with upper and lower bandwidth equal to two. The five diagonals of C are

$$\begin{aligned} \operatorname{diag}(C, 0) = {} & (a_1 d_1, a_2 d_2, a_3 d_3, a_4 d_4, a_5 d_5) \\ & + (0, b_1 f_1, b_2 f_2, b_3 f_3, b_4 f_4) \\ & + (c_1 e_1, c_2 e_2, c_3 e_3, c_4 e_4, 0), \end{aligned}$$

$$\begin{aligned}\operatorname{diag}(C,1) &= (a_1f_1, a_2f_2, a_3f_3, a_4f_4)\\ &\quad + (c_1d_2, c_2d_3, c_3d_4, c_4d_5),\\ \operatorname{diag}(C,-1) &= (b_1d_1, b_2d_2, b_3d_3, b_4d_4)\\ &\quad + (a_2e_1, a_3e_2, a_4e_3, a_5e_4),\\ \operatorname{diag}(C,2) &= (c_1f_2, c_2f_3, c_3f_4),\\ \operatorname{diag}(C,-2) &= (b_2e_1, b_3e_2, b_4e_3).\end{aligned}$$

The number of operations are exactly the same as in the conventional schemes, but only $3^2 = 9$ pointwise vector multiplications are required.

Lemma 7.4.3.
Let $A, B \in \mathbf{R}^{n\times n}$ have lower (upper) bandwidth r and s respectively. Then the product AB has lower (upper) bandwidth $r + s$.

7.4.2 LU Factorization of Banded Matrices

A matrix A for which all nonzero elements are located in consecutive diagonals is called a **band matrix**.

Many applications give rise to linear systems $Ax = b$, where the nonzero elements in the matrix A are located in a band centered along the principal diagonal. Such matrices are called **band matrices** and are the simplest examples of **sparse** matrices, i.e., matrices where only a small proportion of the n^2 elements are nonzero. Such matrices arise frequently in, for example, the numerical solution of boundary value problems for ordinary and partial differential equations.

Several classes of band matrices that occur frequently have special names. Thus, a matrix for which $r = s = 1$ is called **tridiagonal**, if $r = 0$, $s = 1$ it is called (lower) **bidiagonal** etc. For example, the matrix

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & a_{53} & a_{54} & a_{55} & a_{56} \\ & & & a_{64} & a_{65} & a_{66} \end{pmatrix}$$

has $r = 1$, $s = 2$ and $w = 4$.

Band matrices are well-suited for Gaussian elimination, since if no pivoting is required *the band structure is preserved*. Recall that pivoting is not needed for stability, e.g., when A is diagonally dominant.

Theorem 7.4.4. *Let A be a band matrix with upper bandwidth r and lower band width s. If A has an LU-decomposition $A = LU$, then U has upper bandwidth r and L lower bandwidth s.*

Proof. The factors L and U are unique and can be computed, for example, by Doolittle's method (7.2.15). Assume that the first $k-1$ rows of U and columns of L have bandwidth r and s, that is, for $p = 1 : k-1$

$$l_{ip} = 0, \quad i > p+s, \qquad u_{pj} = 0, \quad j > p+r. \tag{7.4.2}$$

The proof is by induction in k. The assumption is trivially true for $k = 1$. Since $a_{kj} = 0$, $j > k+r$ we have from (7.2.9) and (7.4.2)

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj} = 0 - 0 = 0, \quad j > k+r.$$

Similarly it follows that $l_{ik} = 0$, $i > k+s$, which completes the induction step. □

A band matrix $A \in \mathbf{R}^{n\times n}$ may be stored by diagonals in an array of dimension $n \times (r+s+1)$ or $(r+s+1) \times n$. For example, the matrix above can be stored as

$$\begin{matrix} * & * & a_{11} & a_{12} \\ * & a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} & a_{45} \\ a_{53} & a_{54} & a_{55} & a_{56} \\ a_{64} & a_{65} & a_{66} & * \end{matrix}, \quad \text{or} \quad \begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{matrix}.$$

Notice that except for a few elements indicated by asterisks in the initial and final rows, only nonzero elements of A are stored. For example, passing along a row in the second storage scheme above moves along a diagonal of the matrix, and the columns are aligned.

For a general band matrix Algorithm 7.2.2, Gaussian elimination without pivoting, should be modified as follows to operate only on nonzero elements: The algorithms given below are written as if the matrix was conventionally stored. It is a useful exercise to rewrite them for the case when A, L, and U are stored by diagonals!

Algorithm 7.4.1 Banded Gaussian Elimination.
Let $A = \in \mathbf{R}^{n\times n}$ be a given matrix with upper bandwidth r and lower bandwidth s. The following algorithm computes the LU factorization of A, *provided it exists.* The element a_{ij} is overwritten by l_{ij} if $i > j$ and by u_{ij} otherwise.

```
for k = 1 : n − 1
    for i = k + 1 : min(k + s, n)
        l_ik := a_ik^(k) / a_kk^(k);
        for j = k + 1 : min(k + r, n)
            a_ij^(k+1) := a_ij^(k) − l_ik a_kj^(k);
        end
    end
end
```

An operation count shows that this algorithm requires t flams, where

$$t = \begin{cases} nr(s+1) - \frac{1}{2}rs^2 - \frac{1}{6}r^3, & \text{if } r \le s; \\ ns(s+1) - \frac{2}{3}s^3, & \text{if } r = s; \\ ns(r+1) - \frac{1}{2}sr^2 - \frac{1}{6}s^3, & \text{if } r > s. \end{cases}$$

Whenever $r \ll n$ *or* $s \ll n$ this is much less than the $n^3/3$ flams required in the full case.

Analogous savings can be made in forward- and back-substitution. Let L and U be the triangular factors computed by Algorithm 7.4.2. The solution of the two banded triangular systems $Ly = b$ and $Ux = y$ are obtained from

$$y_i = b_i - \sum_{max(1,i-s)}^{i-1} l_{ij}y_j, \quad i = 1:n$$

$$x_i := \Big(y_i - \sum_{j=i+1}^{\min(i+r,n)} u_{ij}x_j\Big)/u_{ii}, \quad i = n:(-1):1.$$

These algorithms require $ns - \frac{1}{2}s^2$ and $(n - \frac{r}{2})(r+1)$ flops, respectively. They are easily modified so that y and x overwrites b in storage.

Unless A is diagonally dominant or symmetric positive definite, partial pivoting should be used. The pivoting will cause the introduction of elements outside the band. This is illustrated below for the case when $s = 2$ and $r = 1$. The first step of the elimination is shown, where it is assumed that a_{31} is chosen as pivot and therefore rows 1 and 3 interchanged:

$$\begin{matrix} a_{31} & a_{32} & a_{33} & a_{34} & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ a_{11} & a_{12} & & & & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & & \\ & & a_{53} & a_{54} & a_{55} & a_{56} & \\ & & & \cdots & & & \end{matrix} \quad \Longrightarrow \quad \begin{matrix} u_{11} & u_{12} & u_{13} & u_{14} & & \\ l_{21} & a_{22}^{(2)} & a_{23}^{(2)} & \mathbf{a}_{24}^{(2)} & & \\ l_{31} & a_{32}^{(2)} & \mathbf{a}_{33}^{(2)} & \mathbf{a}_{34}^{(2)} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & a_{53} & a_{54} & a_{55} & a_{56} \\ & & & \cdots & & \end{matrix} .$$

where fill-in elements are shown in boldface. Hence *the upper bandwidth of U may increase to $r+s$. The matrix L will still have only s elements below the main diagonal in all columns but no useful band structure.* This can be seen from the example above where, e.g., the elements l_{21} and l_{31} may be subject to later permutations, destroying the band-structure of the first column.

Example 7.4.2.

A class of matrices with unsymmetric band structure is upper (lower) **Hessenberg matrices**[18] for which $s = 1$ ($r = 1$). These are of particular interest

[18] Karl Hessenberg (1904–1959) German mathematician and engineer.

in connection with unsymmetric eigenproblems. An upper Hessenberg matrix of order five has the structure

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ 0 & h_{32} & h_{33} & h_{34} & h_{35} \\ 0 & 0 & h_{43} & h_{44} & h_{45} \\ 0 & 0 & 0 & h_{54} & h_{55} \end{pmatrix}.$$

Performing Gaussian elimination the first step will only affect the first two rows of the matrix. The reduced matrix is again Hessenberg and all the remaining steps are similar to the first. If partial pivoting is used then in the first step either h_{11} or h_{21} will be chosen as pivot. Since these rows have the same structure the Hessenberg form will be preserved during the elimination. Clearly only $t = \frac{1}{2}n(n+1)$ flams are needed. Note that with partial pivoting the elimination will not give a factorization $PA = LU$ with L lower bidiagonal. Whenever we pivot, the interchanges should be applied also to L, which will spread out the elements. Therefore L will be lower triangular with only one nonzero off-diagonal element in each column. However, it is more convenient to leave the elements in L in place.

If $A \in \mathbf{R}^{n\times n}$ is Hessenberg then $\rho_n \leq n$ with partial pivoting. This follows the since at the start of the k stage row $k+1$ of the reduced matrix has not been changed and the elements the pivot row has elements of modulus at most k times the largest element of H.

In the special case when A is a symmetric positive definite banded matrix with upper and lower bandwidth $r = s$, the factor L in the Cholesky factorization $A = LL^T$ has lower bandwidth r. From Algorithm 7.3.2 we easily derive the following banded version:

Algorithm 7.4.2 Band Cholesky Algorithm, column-wise Order.

for $j = 1 : n$
  $p = \max(1, j - r)$;
  for $i = p : j - 1$
    $r_{ij} = \Big(a_{ij} - \sum_{k=p}^{i-1} r_{ki} r_{kj}\Big)/r_{ii}$;
  end
  $r_{jj} = \Big(a_{jj} - \sum_{k=p}^{j-1} r_{kj}^2\Big)^{1/2}$;
end

If $r \ll n$ this algorithm requires about $\frac{1}{2}nr(r+3)$ flops and n square roots. As input we just need the upper triangular part of A, which can be stored in an $n \times (r+1)$ array.

7.4.3 Tridiagonal Linear Systems

A matrix of the form

$$A = \begin{pmatrix} a_1 & c_2 & & & \\ b_2 & a_2 & c_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_n \\ & & & b_n & a_n \end{pmatrix}. \tag{7.4.3}$$

is called **tridiagonal**. Note that the $3n-2$ nonzero elements in A are conveniently stored in three vectors a, c, and d. A is said to be irreducible if b_i and c_i are nonzero for $i = 2 : n$. Let A be reducible, say $c_k = 0$. Then A can be written as a lower block triangular form

$$A = \begin{pmatrix} A_1 & 0 \\ L_1 & A_2 \end{pmatrix},$$

where A_1 and A_2 are tridiagonal. If A_1 or A_2 is reducible then this blocking can be applied recursively until a block form with irreducible tridiagonal blocks is obtained..

If Gaussian elimination with partial pivoting is applied to A then a factorization $PA = LU$ is obtained, where L has at most one nonzero element below the diagonal in each column and U has upper bandwidth two (cf. the Hessenberg case in Example 7.4.2). If A is diagonally dominant, then no pivoting is requires and the factorization $A = LU$ exists. By Theorem 7.4.4 it has the form

$$A = LU = \begin{pmatrix} 1 & & & & \\ \gamma_2 & 1 & & & \\ & \gamma_3 & \ddots & & \\ & & \ddots & 1 & \\ & & & \gamma_n & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & c_2 & & & \\ & \alpha_2 & c_3 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & c_n \\ & & & & \alpha_n \end{pmatrix}. \tag{7.4.4}$$

By equating elements in A and LU it is verified that the upper diagonal in U equals that in A, and for the other elements in L and U we obtain the recursions

$$\alpha_1 = a_1, \quad \gamma_k = b_k/\alpha_{k-1}, \qquad \alpha_k = a_k - \gamma_k c_k, \quad k = 2 : n. \tag{7.4.5}$$

Note that the elements γ_k and α_k can overwrite b_k and a_k, respectively. The solution to the system $Ax = f$ can then be computed by solving $Ly = f$ by $Ux = y$ by back- and forward-substitution

$$y_1 = f_1, \qquad y_i = f_i - \gamma_i y_{i-1}, \quad i = 2 : n, \tag{7.4.6}$$

$$x_n = y_n/\alpha_n, \qquad x_i = (y_i - c_{i+1}x_{i+1})/\alpha_i, \quad i = n-1 : 1. \tag{7.4.7}$$

The total number of flops is about $1.5n$ for the factorization and $2.5n$ for the solution.

If A is tridiagonal then it is easily proved by induction that $\rho_n \leq 2$ with partial pivoting. This result is a special case of a more general result.

Theorem 7.4.5. [Bothe [10]] *If $A \in \mathbf{C}^{n\times n}$ has upper and lower bandwidth p then the growth factor in GE with partial pivoting satisfies*

$$\rho_n \le 2^{2p-1} - (p-1)2^{p-2}.$$

In particular for a tridiagonal matrix ($p = 1$) $\rho_n \le 2$.

When A is symmetric positive definite and tridiagonal (7.4.3)

$$A = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & b_n \\ & & & b_n & a_n \end{pmatrix}, \tag{7.4.8}$$

we can write the factorization

$$A = LDL^T, \quad D = \operatorname{diag}(\alpha_1, \ldots, \alpha_n), \tag{7.4.9}$$

where L is as in (7.4.4). The algorithm then reduces to

$$\alpha_1 = a_1, \quad \gamma_k = b_k/\alpha_{k-1}, \qquad \alpha_k = a_k - \gamma_k b_k, \quad k = 2:n. \tag{7.4.10}$$

Sometimes it is more convenient to write

$$A = U^T D^{-1} U, \quad D = \operatorname{diag}(a_1, \ldots, a_n).$$

In the scalar case U is given by (7.4.4) (with $c_k = b_k$), and the elements in U and D are computed from

$$\alpha_1 = a_1, \qquad \alpha_k = a_k - b_k^2/\alpha_{k-1}. \quad k = 2:n. \tag{7.4.11}$$

The recursion (7.4.5) for the LU factorization of a tridiagonal matrix is highly serial. An algorithm for solving tridiagonal systems, which has considerable inherent parallelism, is **cyclic reduction** also called **odd-even reduction**. This is the most preferred method for solving large tridiagonal systems on parallel computers.

The basic step in cyclic reduction is to eliminate all the odd unknowns to obtain a reduced tridiagonal system involving only even numbered unknowns. This process is repeated recursively until a system involving only a small order of unknowns remains. This is then solved separately and the other unknowns can then be computed in a back-substitution process. We illustrate this process on a tridiagonal system $Ax = f$ of order $n = 2^3 - 1 = 7$. If P is a permutation matrix such that $P(1, 2, \ldots, 7) = (1, 3, 5, 7, 2, 4, 6)^T$ the transformed system $PAP^T(Px) = P^T f$, will have the form

$$\left(\begin{array}{cccc|ccc} a_1 & & & & c_2 & & \\ & a_3 & & & b_3 & c_4 & \\ & & a_5 & & & b_5 & c_6 \\ & & & a_7 & & & b_7 \\ \hline b_2 & c_3 & & & a_2 & & \\ & b_4 & c_5 & & & a_4 & \\ & & b_6 & c_7 & & & a_6 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_3 \\ x_5 \\ x_7 \\ \hline x_2 \\ x_4 \\ x_6 \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_3 \\ f_5 \\ f_7 \\ \hline f_2 \\ f_4 \\ f_6 \end{array}\right).$$

It is easily verified that after eliminating the odd variables from the even equations the resulting system is again tridiagonal. Rearranging these as before the system becomes

$$\left(\begin{array}{cc|c} a_2' & & c_4' \\ & a_6' & b_6' \\ \hline b_4' & c_6' & a_4' \end{array}\right) = \begin{pmatrix} x_2 \\ x_6 \\ x_4 \end{pmatrix} = \begin{pmatrix} f_2' \\ f_6' \\ f_4' \end{pmatrix}.$$

After elimination we are left with one equation in one variable

$$a_4'' x_4 = f_4''.$$

Solving for x_4 we can compute x_2 and x_6 from the first two equations in the previous system. Substituting these in the first four equations we get the odd unknowns x_1, x_3, x_5, x_7. Clearly this scheme can be generalized. For a system of dimension $n = 2^p - 1$, p steps are required in the reduction. Note, however, that it is possible to stop at any stage, solve a tridiagonal system and obtain the remaining variables by substitution. Therefore it can be used for any dimension n.

The derivation shows that cyclic reduction is equivalent to Gaussian elimination without pivoting on a reordered system. Therefore it is stable if the matrix is diagonally dominant or symmetric positive definite. In contrast to the conventional algorithm there is some fill during the elimination and about 2.7 times more operations are needed.

Example 7.4.3.

Consider the linear system $Ax = b$, where A is a symmetric positive definite tridiagonal matrix. Then A has positive diagonal elements and the symmetrically scaled matrix DAD, where $D = \operatorname{diag}(d_1, \ldots, d_n)$, $d_i = 1/\sqrt{a_i}$, has unit diagonal elements. After an odd-even permutation the system has the 2×2 block form

$$\begin{pmatrix} I & F \\ F^T & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}, \tag{7.4.12}$$

with F lower bidiagonal. After block elimination the Schur complement system becomes

$$(I - F^T F)x = d - F^T c.$$

Here $I - F^T F$ is again a positive definite tridiagonal matrix. Thus the process can be repeated recursively.

Boundary value problems, where the solution is subject to *periodic boundary conditions*, often lead to matrices of the form

$$B = \left(\begin{array}{ccccc|c} a_1 & c_2 & & & & b_1 \\ b_2 & a_2 & c_3 & & & \\ & \ddots & \ddots & \ddots & & \\ & & b_{n-1} & a_{n-1} & & c_n \\ \hline c_1 & & & b_n & & a_n \end{array}\right), \tag{7.4.13}$$

which are tridiagonal except for the two corner elements b_1 and c_1. We now consider the is real symmetric case, $b_i = c_i$, $i = 1 : n$ Partitioning B in 2×2 block form as above, we seek a factorization

$$B = \begin{pmatrix} A & u \\ v^T & a_n \end{pmatrix} = \begin{pmatrix} L & 0 \\ y^T & 1 \end{pmatrix} \begin{pmatrix} U & z \\ 0 & d_n \end{pmatrix}$$

where $u = b_1e_1+c_ne_{n-1}$, $v = c_1e_1+b_ne_{n-1}$. Multiplying out we obtain the equations

$$A = LU, \quad u = Lz, \quad v^T = y^TU, \quad a_n = y^Tz + d_n$$

Assuming that no pivoting is required the factorization $A = LU$, where L and U are bidiagonal, is obtained using (7.4.5). The vectors y and z are obtained from the lower triangular systems

$$Lz = b_1e_1 + c_ne_{n-1}, \qquad U^Ty = c_1e_1 + c_ne_{n-1},$$

and $d_n = a_n - y^Tz$. Note that y and z will be full vectors.

Cyclic reduction can be applied to systems $Bx = f$, where B has the tridiagonal form in (7.4.13). If n is even the reduced system obtained after eliminating the odd variables in the even equations will again have the form (7.4.13). For example, when $n = 2^3 = 8$ the reordered system is

$$\left(\begin{array}{cccc|cccc} a_1 & & & & c_2 & & & b_1 \\ & a_3 & & & b_3 & c_4 & & \\ & & a_5 & & & b_5 & c_6 & \\ & & & a_7 & & & b_7 & c_8 \\ \hline b_2 & c_3 & & & a_2 & & & \\ & b_4 & c_5 & & & a_4 & & \\ & & b_6 & c_7 & & & a_6 & \\ c_1 & & & b_8 & & & & a_8 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_3 \\ x_5 \\ x_7 \\ \hline x_2 \\ x_4 \\ x_6 \\ x_8 \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_3 \\ f_5 \\ f_7 \\ \hline f_2 \\ f_4 \\ f_6 \\ f_8 \end{array}\right).$$

If $n = 2^p$ the process can applied recursively. After p steps one equation in a single unknown is obtained. Cyclic reduction here does not require extra storage and also has a slightly lower operation count than ordinary Gaussian elimination.

We finally consider the case when A is a **symmetric indefinite tridiagonal** matrix. It would be possible to use LU factorization with partial pivoting, but this destroys symmetry and gives no information about the inertia of A. Instead a block factorization $A = LDL^T$ can be computed using no interchanges as follows. Set $\sigma = \max_{1\le i\le n} |a_{ij}|$ and $\alpha = (\sqrt{5} - 1)/2 \approx= 0.62$. In the first stage we take a_{11} as pivot if $\sigma|a_{11}| \ge a_{21}^2$. Otherwise we take the 2×2 pivot

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

This factorization can be shown to be normwise backward stable and is a good way to solve such symmetric indefinite tridiagonal linear systems.

7.4.4 Inverses of Banded Matrices

It is important to note that *the inverse A^{-1} of a banded matrix in general has no zero elements*. Hence one should never attempt to explicitly compute the elements of the inverse of a band matrix. Since banded systems often have very large dimensions even storing the elements in A^{-1} may be infeasible!

The following theorem states that the lower triangular part of the inverse of an upper Hessenberg matrix has a very simple structure.

Theorem 7.4.6.

Let $H =\in \mathbf{R}^{n\times n}$ be an upper Hessenberg matrix with nonzero elements in the subdiagonal, $h_{i+1,i} \neq 0$, $i = 1 : n-1$. Then there are vectors p and q such that

$$(H^{-1})_{ij} = p_i q_j, \quad i \geq j. \tag{7.4.14}$$

Proof. See Ikebe [46] □

A tridiagonal matrix A is both lower and upper Hessenberg. Hence if A is irreducible it follows that there are vectors x, y, p and q such that

$$(A^{-1})_{ij} = \begin{cases} x_i y_j, & i \leq j, \\ p_i q_j, & i \geq j. \end{cases} \tag{7.4.15}$$

Note that $x_1 \neq 0$ and $y_n \neq 0$, since otherwise the entire first row or last column of A^{-1} would be zero, contrary to the assumption of the nonsingularity of A. The vectors x and y (as well as p and q) are unique up to scaling by a nonzero factor. There is some redundancy in this representation since $x_i y_i = p_i q_i$. It can be shown that $3n-2$ parameters are needed to represent the inverse, which equals the number of nonzero elements in A.

The following algorithm has been suggested by N. J. Higham to compute the vectors x, y, p and q:

1. Compute the LU factorization of A.

2. Use the LU factorization to solve for the vectors y and z, where $A^T y = e_1$ and $Az = e_n$. Similarly solve for p and r, where $Ap = e_1$ and $A^T r = e_n$.

3. Set $q = p_n^{-1} r$ and $x = y_n^{-1} z$.

This algorithm is not foolproof and can fail because of overflow.

Example 7.4.4. Let A be a symmetric, positive definite tridiagonal matrix with elements $a_1 = 1$,

$$a_i = 2, \quad b_i = c_i = -1, \quad i = 2 : 5.$$

Although the Cholesky factor L of A is bidiagonal the inverse

$$A^{-1} = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

is full. Here $x = p$, $y = q$, can be determined up to a scaling factor from the first and last columns of A^{-1}.

The inverse of any banded matrix has a special structure related to low rank matrices. The first study of inverse of general banded matrices was Asplund [5].

Review Questions

1. Give an example of matrix multiplication by diagonals.

2. (a) If a is a column vector what is meant by $\operatorname{diag}(a, k)$?

(b) If A is a square matrix what is meant by $\operatorname{diag}(A, k)$?

3. (a) Let $A \in \mathbf{R}^{n\times n}$ be a banded matrix with upper bandwidth p and lower bandwidth q. Show how A can be efficiently stored when computing the LU factorization.

(b) Assuming that the LU factorization can be carried out without pivoting, what are the structures of the resulting L and U factors of A?

(c) What can you say about the structure of the inverses of L and U?

4. Let $A \in \mathbf{R}^{n\times n}$ be a banded matrix with upper bandwidth p and lower bandwidth q. Assuming that the LU factorization of A can be carried out without pivoting, roughly how many operations are needed? You need only give the dominating term when $p, q \ll n$.

5. Give a bound for the growth ratio ρ_n in Gaussian elimination with partial pivoting, when the matrix A is: (a) Hessenberg; (b) tridiagonal.

Problems

1. (a) Let $A, B \in \mathbf{R}^{n\times n}$ have lower (upper) bandwidth r and s respectively. Show that the product AB has lower (upper) bandwidth $r + s$.

(b) An upper Hessenberg matrix H is a matrix with lower bandwidth $r = 1$. Using the result in (a) deduce that the product of H and an upper triangular matrix is again an upper Hessenberg matrix.

2. Show that an irreducible nonsymmetric tridiagonal matrix A can be written $A = DT$, where T is symmetric tridiagonal and $D = \operatorname{diag}(d_k)$ is diagonal with

elements

$$d_1 = 1, \quad d_k = \prod_{j=2}^{k} c_j/b_j, \quad k = 2 : n. \tag{7.4.16}$$

3. (a) Let $A \in \mathbf{R}^{n\times n}$ be a symmetric, tridiagonal matrix such that $\det(A_k) \neq 0$, $k = 1 : n$. Then the decomposition $A = LDL^T$ exists and can be computed by the formulas given in (7.4.10). Use this to derive a recursion formula for computing $\det(A_k)$, $k = 1 : n$.

(b) Determine the largest n for which the symmetric, tridiagonal matrix

$$A = \begin{pmatrix} 2 & 1.01 & & & \\ 1.01 & 2 & 1.01 & & \\ & 1.01 & \ddots & \ddots & \\ & & \ddots & \ddots & 1.01 \\ & & & 1.01 & 2 \end{pmatrix} \in \mathbf{R}^{n\times n}$$

is positive definite.

4. (a) Show that for $\lambda \geq 2$ it holds that $B = \mu LL^T$, where

$$B = \begin{pmatrix} \mu & -1 & & & \\ -1 & \lambda & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \lambda & -1 \\ & & & -1 & \lambda \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & & & & \\ -\sigma & 1 & & & \\ & -\sigma & \ddots & & \\ & & \ddots & 1 & \\ & & & -\sigma & 1 \end{pmatrix},$$

and

$$\mu = \lambda/2 \pm (\lambda^2/4 - 1)^{1/2}, \quad \sigma = 1/\mu.$$

Note that L has constant diagonals.

(b) Suppose we want to solve and equation system $Ax = b$, where the matrix A differs from B in the element (1,1),

$$A = B + \delta e_1 e_1^T, \quad \delta = \lambda - \mu, \quad e_1^T = (1, 0, \ldots, 0).$$

Show, using the Sherman–Morrison formula (7.1.26), that the solution $x = A^{-1}b$ can be computed from

$$x = y - \gamma L^{-T} f, \quad \gamma = \delta(e_1^T y)/(\mu + \delta f^T f)$$

where y and f satisfies $\mu LL^T y = b$, $Lf = e_1$.

5. Consider the symmetric tridiagonal matrix

$$A_n = \begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}.$$

For $n = 20, 40$ use the Cholesky factorization of A_n and Higham's algorithm to determine vectors x and y so that $(A_n^{-1})_{ij} = x_i y_j$ for $i, j = 1 : n$. Verify that there is a range of approximately θ^n in the size of the components of these vectors, where $\theta = 2 + \sqrt{3}$.

5. (a) Write a function implementing the multiplication $C = AB$, where $A = \operatorname{diag}(a, r)$ and $B = \operatorname{diag}(b, s)$ both consist of a single diagonal. Use the formulas in Lemma 7.4.2.

(b) Write a function for computing the product $C = AB$ of two banded matrices using the $w_1 w_2$ calls to the function in (a), where and w_2 are the bandwidth of A and B, respectively.

6. Derive expressions for computing δ_k, $k = 1 : n-1$ and α_n in the factorization of the periodic tridiagonal matrix A in (7.4.13).

7. Let B be a symmetric matrix of the form (7.4.13). Show that

$$B = T + \sigma u u^T, \quad u = (1, 0, \ldots, 0, -1)^T.$$

where T is a certain symmetric, tridiagonal matrix. What is σ and T. Derive an algorithm for computing L by modifying the algorithm (7.4.10).

7.5 Perturbation Theory and Condition Estimation

7.5.1 Component-Wise Perturbation Analysis

In Sec. 7.1.8 bounds were derived for the perturbation in the solution x to a linear system $Ax = b$, when the data A and b are perturbed. Sharper bounds can often be obtained in case if the data is subject to the perturbations, which are bounded component-wise. Assume that

$$|\delta a_{ij}| \le \omega e_{ij}, \quad |\delta b_i| \le \omega f_i. \quad i, j = 1 : n,$$

where $e_{ij} \ge 0$ and $f_i \ge 0$ are known. These bounds can be written as

$$|\delta A| \le \omega E, \quad |\delta b| \le \omega f, \tag{7.5.1}$$

where the absolute value of a matrix A and vector b is defined by

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

The partial ordering "$\le$" for matrices A, B and vectors x, y, is to be interpreted component-wise[19] It is easy to show that if $C = AB$, then

$$|c_{ij}| \le \sum_{k=1}^{n} |a_{ik}|\,|b_{kj}|,$$

and hence $|C| \le |A|\,|B|$. A similar rule $|Ax| \le |A|\,|x|$ holds for matrix-vector multiplication.

For deriving the componentwise bounds we need the following result.

[19]Note that $A \le B$ in other contexts means that $B - A$ is positive semidefinite.

Lemma 7.5.1.

Let $F \in \mathbf{R}^{n\times n}$ be a matrix for which $\| \, |F| \, \| < 1$. Then the matrix $(I - |F|)$ is nonsingular and

$$|(I - F)^{-1}| \leq (I - |F|)^{-1}. \tag{7.5.2}$$

Proof. The nonsingularity follows form Lemma 7.1.15. Using the identity $(I - F)^{-1} = I + F(I - F)^{-1}$ we obtain

$$|(I - F)^{-1}| \leq I + |F||(I - F)^{-1}|$$

from which the inequality (7.5.2) follows. □

Theorem 7.5.2.

Consider the perturbed linear system $(A + \delta A)(x + \delta x) = b + \delta b$, where A is nonsingular. Assume that δA and δb satisfy the componentwise bounds in (7.5.1) and that

$$\omega \| \, |A^{-1}| \, E \, \| < 1.$$

Then $(A + \delta A)$ is nonsingular and

$$\|\delta x\| \leq \frac{\omega}{1 - \omega\kappa_E(A)} \| \, |A^{-1}|(E \, |x| + f)\|, \tag{7.5.3}$$

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

Proof. Taking absolute values in (7.1.62) gives

$$|\delta x| \leq |(I + A^{-1}\delta A)^{-1}| \, |A^{-1}|(|\delta A||x| + |\delta b|). \tag{7.5.4}$$

Using Lemma 7.5.1 it follows from the assumption that the matrix $(I - |A^{-1}|\delta A|)$ is nonsingular and from (7.5.4) we get

$$|\delta x| \leq (I - |A^{-1}||\delta A|)^{-1} \, |A^{-1}|(|\delta A||x| + |\delta b|).$$

Using the componentwise bounds in (7.5.1) we get

$$|\delta x| \leq \omega(I - \omega|A^{-1}|E)^{-1}|A^{-1}|(E|x| + f), \tag{7.5.5}$$

provided that $\omega\kappa_E(A) < 1$ Taking norms in (7.5.5) and using Lemma 7.1.15 with $F = A^{-1}\delta A$ proves (7.5.3). □

Taking $E = |A|$ and $f = |b|$ in (7.5.1) corresponds to bounds for the **componentwise relative errors** in A and b,

$$|\delta A| \leq \omega|A|, \qquad |\delta b| \leq \omega|b|. \tag{7.5.6}$$

For this special case Theorem 7.5.2 gives

$$\|\delta x\| \leq \frac{\omega}{1 - \omega\kappa_{|A|}(A)} \| \, |A^{-1}|(|A| \, |x| + |b|) \, \|, \tag{7.5.7}$$

where

$$\kappa_{|A|}(A) = \| \, |A^{-1}||A| \, \|, \tag{7.5.8}$$

(or cond (A)) is the **Bauer–Skeel condition number** of the matrix A. Note that since $|b| \leq |A| \, |x|$, it follows that

$$\|\delta x\| \leq 2\omega \| \, |A^{-1}||A| \, |x| \, \| + O(\omega^2) \leq 2\omega \kappa_{|A|}(A)\|x\| + O(\omega^2).$$

If $\hat{A} = DA$, $\hat{b} = Db$ where $D > 0$ is a diagonal scaling matrix, then $|\hat{A}^{-1}| = |A^{-1}||D^{-1}|$. Since the perturbations scale similarly, $\delta\hat{A} = D\delta A$, $\delta\hat{b} = D\delta b$, it follows that

$$|\hat{A}^{-1}||\delta\hat{A}| = |A^{-1}||\delta A|, \qquad |\hat{A}^{-1}||\delta\hat{b}| = |A^{-1}||\delta b|.$$

Thus the bound in (7.5.7) and also $\kappa_{|A|}(A)$ are *invariant under row scalings*.

For the l_1-norm and l_∞-norm it holds that

$$\kappa_{|A|}(A) = \| \, |A^{-1}||A| \, \| \leq \| \, |A^{-1}| \, \| \, \| \, |A| \, \| = \|A^{-1}\| \, \|A\| = \kappa(A),$$

i.e., the solution of $Ax = b$ is no more badly conditioned with respect to the component-wise relative perturbations than with respect to normed perturbations. On the other hand, it is possible for $\kappa_{|A|}(A)$ to be much smaller than $\kappa(A)$.

The analysis in Sec. 7.1.8 may not be adequate, when the perturbations in the elements of A or b are of different magnitude, as illustrated by the following example.

Example 7.5.1.

The linear system $Ax = b$, where

$$A = \begin{pmatrix} 1 & 10^4 \\ 1 & 10^{-4} \end{pmatrix}, \quad b = \begin{pmatrix} 10^4 \\ 1 \end{pmatrix},$$

has the approximate solution $x \approx (1, 1)^T$. Assume that the vector b is subject to a perturbation δb such that $|\delta b| \leq (1, 10^{-4})^T$. Using the ∞-norm we have $\|\delta b\|_\infty = 1$, $\|A^{-1}\|_\infty = 1$ (neglecting terms of order 10^{-8}). Theorem 7.1.18 then gives the gross overestimate $\|\delta x\|_\infty \leq 1$.

Multiplying the first equation by 10^{-4}, we get an equivalent system $\hat{A}x = \hat{b}$ where

$$\hat{A} = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The perturbation in the vector b is now $|\delta\hat{b}| \leq 10^{-4}(1, 1)^T$, and from $\|\delta\hat{b}\|_\infty = 10^{-4}$, $\|(\hat{A})^{-1}\|_\infty = 1$, we get the sharp estimate $\|\delta x\|_\infty \leq 10^{-4}$. The original matrix A is only **artificially ill-conditioned**. By a scaling of the equations we obtain a well-conditioned system. How to scale linear systems for Gaussian elimination is a surprisingly intricate problem, which is further discussed in Sec. 7.6.3.

Consider the linear systems in Example 7.5.1. Neglecting terms of order 10^{-8} we have

$$|\hat{A}^{-1}||\hat{A}| = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix} \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix} = \begin{pmatrix} 1 & 2 \cdot 10^{-4} \\ 2 \cdot 10^{-4} & 1 \end{pmatrix},$$

By the scaling invariance $\operatorname{cond}(A) = \operatorname{cond}(\hat{A}) = 1 + 2 \cdot 10^{-4}$ in the ∞-norm. Thus the componentwise condition number correctly reveals that the system is well-conditioned for componentwise small perturbations.

7.5.2 Backward Error Bounds

We now derive a simple **a posteriori** bound for the backward error of a computed solution $\bar{x}$. These bounds are usually much sharper than a priori bounds and hold regardless of the algorithm used to compute $\bar{x}$.

Given $\bar{x}$, there are an infinite number of perturbations δA and δb for which $(A + \delta A)\bar{x} = b + \delta b$ holds. Clearly δA and δb must satisfy

$$\delta A\bar{x} - \delta b = b - A\bar{x} = r,$$

where $r = b - A\bar{x}$ is the residual vector corresponding to the computes solution. An obvious choice is to take $\delta A = 0$, and $\delta b = -r$. If we instead take $\delta b = 0$, we get the following result.

Theorem 7.5.3.

Let $\bar{x}$ be a purported solution to $Ax = b$, and set $r = b - A\bar{x}$. Then if

$$\delta A = r\bar{x}^T/\|\bar{x}\|_2^2, \tag{7.5.9}$$

$\bar{x}$ satisfies $(A + \delta A)\bar{x} = b$ and this has the smallest l_2-norm $\|\delta A\|_2 = \|r\|_2/\|\bar{x}\|_2$ of any such δA.

Proof. Clearly $\bar{x}$ satisfies $(A + \delta A)\bar{x} = b$ if and only if $\delta A\bar{x} = r$. For any such δA it holds that $\|\delta A\|_2\|\bar{x}\|_2 \geq \|r\|_2$ or $\|\delta A\|_2 \geq \|r\|_2/\|\bar{x}\|_2$. For the particular δA given by (7.5.9) we have $\delta A\bar{x} = r\bar{x}^T\bar{x}/\|\bar{x}\|^2 = r$. From

$$\|r\bar{x}^T\|_2 = \sup_{\|y\|_2=1} \|r\bar{x}^T y\|_2 = \|r\|_2 \sup_{\|y\|_2=1} |\bar{x}^T y| = \|r\|_2\|\bar{x}\|_2,$$

it follows that $\|\delta A\|_2 = \|r\|_2/\|\bar{x}\|_2$ and hence the δA in (7.5.9) is of minimum l_2-norm. □

Similar bounds for the l_1-norm and l_∞-norm are given in Problem 5.

It is often more useful to consider the **component-wise backward error** ω of a computed solution. The following theorem shows that also this can be cheaply computed

Theorem 7.5.4. (Oettli and Prager [1964]).

Let $r = b - A\bar{x}$, E and f be nonnegative and set

$$\omega = \max_i \frac{|r_i|}{(E|\bar{x}| + f)_i}, \tag{7.5.10}$$

where $0/0$ *is interpreted as* 0. *If* $\omega \neq \infty$, *there is a perturbation* δA *and* δb *with*

$$|\delta A| \leq \omega E, \qquad |\delta b| \leq \omega f, \tag{7.5.11}$$

such that

$$(A + \delta A)\bar{x} = b + \delta b. \tag{7.5.12}$$

Moreover, ω *is the smallest number for which such a perturbation exists.*

Proof. From (7.5.10) we have

$$|r_i| \leq \omega(E|\bar{x}| + f)_i,$$

which implies that $r = D(E|\bar{x}| + f)$, where $|D| \leq \omega I$. It is then readily verified that

$$\delta A = DE \operatorname{diag}\big(\operatorname{sign}(\bar{x}_1), \ldots, \operatorname{sign}(\bar{x}_n)\big), \qquad \delta b = -Df$$

are the required backward perturbations.

Further, given perturbations δA and δb satisfying equations (7.5.11)–(7.5.12) for some ω we have

$$|r| = |b - A\bar{x}| = |\delta A\bar{x} - \delta b| \leq \omega(E|\bar{x}| + f).$$

Hence $\omega \geq |r_i|/(E|\bar{x}| + f)_i$, which shows that ω as defined by (7.5.10) is optimal. □

In particular we can take $E = |A|$, and $f = |b|$ in Theorem 7.1.18, to get an expression for the component-wise relative backward error ω of a computed solution. This can then be used in (7.5.6) or (7.5.7) to compute a bound for $\|\delta x\|$.

Example 7.5.2. Consider the linear system $Ax = b$, where

$$A = \begin{pmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.8642 \\ 0.1440 \end{pmatrix}.$$

Suppose that we are given the approximate solution $\bar{x} = (0.9911, -0.4870)^T$. The residual vector corresponding to $\bar{x}$ is very small,

$$r = b - A\bar{x} = (-10^{-8}, 10^{-8})^T.$$

However, not a single figure in $\bar{x}$ is correct! The *exact* solution is $x = (2, -2)^T$, as can readily verified by substitution. *Although a zero residual implies an exact solution, a small residual alone does not necessarily imply an accurate solution.* (Compute the determinant of A and then the inverse A^{-1}!)

It should be emphasized that the system in this example is contrived. In practice one would be highly unfortunate to encounter such an ill-conditioned 2×2 matrix.[20]

[20] As remarked by a prominent expert in error-analysis "Anyone unlucky enough to encounter this sort of calamity has probably already been run over by a truck"!

7.5.3 Estimating Condition Numbers

The perturbation analysis has shown that the norm-wise relative perturbation in the solution x of a linear system can be bounded by

$$\|A^{-1}\| \left(\|\delta A\| + \|\delta b\|/\|x\|\right), \tag{7.5.13}$$

or, in case of componentwise analysis, by

$$\| \, |A^{-1}(| \, |E|x| + f|) \, \|. \tag{7.5.14}$$

To compute these upper bounds exactly is costly since $2n^3$ flops are required to compute A^{-1}, even if the LU factorization of A is known (see Section 7.2.5). In practice, it will will suffice with an *estimate* of $\|A^{-1}\|$ (or $\| \, |A^{-1}| \, \|$, which need not be very precise.

The first algorithm for condition estimation to be widely used was suggested by Cline, Moler, Stewart, and Wilkinson [14]. It is based on computing

$$y = (A^T A)^{-1} u = A^{-1}(A^{-T} u) \tag{7.5.15}$$

by solving the two systems $A^T w = u$ and $Ay = w$. A *lower bound* for $\|A^{-1}\|$ is then given by

$$\|A^{-1}\| \geq \|y\|/\|w\|. \tag{7.5.16}$$

If an LU factorization of A is known this only requires $O(n^2)$ flops. The computation of $y = A^{-T} w$ involves solving the two triangular systems

$$U^T v = u, \qquad L^T w = v.$$

Similarly the vector y and w are obtained by solving the triangular systems

$$Lz = w, \qquad Uy = z,$$

For (7.5.16) to be a reliable estimate the vector u must be carefully chosen so that it reflects any possible ill-conditioning of A. Note that if A is ill-conditioned this is likely to be reflected in U, whereas L, being unit upper triangular, tends to be well-conditioned. To enhance the growth of v we take $u_i = \pm 1$, $i = 1 : n$, where the sign is chosen to maximize $|v_i|$. The final estimate is taken to be

$$1/\kappa(A) \leq \|w\|/(\|A\|\|y\|), \tag{7.5.17}$$

since then a singular matrix is signaled by zero rather than by ∞ and overflow is avoided. We stress that (7.5.17) always *underestimates* $\kappa(A)$. Usually the l_1-norm is chosen because the matrix norm $\|A\|_1 = \max_j \|a_j\|_1$ can be computed from the columns a_j of A. This is often referred to as the LINPACK condition estimator. A detailed description of an implementation is given in the LINPACK Guide, Dongarra et al. [18, 1979, pp. 11-13]. In practice it has been found that the LINPACK condition estimator seldom is off by a factor more than 10. However, counter examples can be constructed showing that it can fail. This is perhaps to be expected for any estimator using only $O(n^2)$ operations.

Equation (7.5.15) can be interpreted as performing one step of the inverse power method on A^TA using the special starting vector u. As shown in Section 10.4.2 this is a standard method for computing the largest singular value $\sigma_1(A^{-1}) = \|A^{-1}\|_2$. An alternative to starting with the vector u is to use a *random* starting vector and perhaps carrying out several steps of inverse iteration with A^TA.

An alternative 1-norm condition estimator has been devised by Hager [36] and improved by Higham [39]. This estimates

$$\|B\|_1 = \max_j \sum_{i=1}^{n} |b_{ij}|,$$

assuming that Bx and B^Tx can be computed for an arbitrary vector x. It can also be used to estimate the infinity norm since $\|B\|_\infty = \|B^T\|_1$. It is based on the observation that

$$\|B\|_1 = \max_{x \in S} \|Bx\|_1, \quad S = \{x \in \mathbf{R}^n \mid \|x\|_1 \le 1\}.$$

is the maximum of a convex function $f(x) = \|Bx\|_1$ over the convex set S. This implies that the maximum is obtained at an extreme point of S, i.e. one of the $2n$ points

$$\{\pm e_j \mid j = 1 : n\},$$

where e_j is the jth column of the unit matrix. If $y_i = (Bx)_i \neq 0$, $\forall i$, then $f(x)$ is differentiable and by the chain rule the gradient is

$$\partial f(x) = \xi^T B, \quad \xi_i = \begin{cases} +1 & \text{if } y_i > 0, \\ -1 & \text{if } y_i < 0. \end{cases}$$

If $y_i = 0$, for some i, then $\partial f(x)$ is a subgradient of f at x. Note that the subgradient is not unique. Since f is convex, the inequality

$$f(y) \ge f(x) + \partial f(x)(y - x), \quad \forall x, y \in \mathbf{R}^n.$$

is always satisfied.

The algorithm starts with the vector $x = n^{-1}e = n^{-1}(1, 1, \ldots, 1)^T$, which is on the boundary of S. We set $\partial f(x) = z^T$, where $z = B^T\xi$, and find an index j for which $|z_j| = \max_i |z_i|$. It can be shown that $|z_j| \le z^Tx$ then x is a local maximum. If this inequaity is satisfied then we stop. By the convexity of $f(x)$ and the fact that $f(e_j) = f(-e_j)$ we conclude that $f(e_j) > f(x)$. Replacing x by e_j we repeat the process. Since the estimates are strictly increasing each vertice of S is visited at most once. The iteration must therefore terminate in a finite number of steps. It has been observed that usually it terminates after just four iterations with the exact value of $\|B\|_1$.

We now show that the final point generated by the algorithm is a local maximum. Assume first that $(Bx)_i \neq 0$ for all i. Then $f(x) = \|Bx\|_1$ is linear in a neighborhood of x. It follows that x is a local maximum of $f(x)$ over S if and only if

$$\partial f(x)(y - x) \le 0, \quad \forall y \in S.$$

If y is a vertex of S, then $\partial f(x)y = \pm\partial f(x)_i$, for some i since all but one component of y is zero. If $|\partial f(x)_i| \leq \partial f(x)x$, for all i, it follows that $\partial f(x)(y-x) \leq 0$ whenever y is a vertex of S. Since S is the convex hull of its vertices it follows that $\partial f(x)(y-x) \leq 0$, for all $y \in S$. Hence x is a locla maximum. In case some component of Bx is zero the above argument must be slightly modified; see Hager [36].

Algorithm 7.5.1 Hager's 1-norm estimator.

$x = n^{-1}e$
repeat
  $y = Bx$
  $\xi = \text{sign}\,(y)$
  $z = B^T\xi$
  if $\|z\|_\infty \leq z^T x$
    $\gamma = \|y\|_1$; quit
  end
  $x = e_j, \text{where} |z_j| = \|z\|_\infty$
end

To use this algorithm to estimate $\|A^{-1}\|_1 = \| \, |A^{-1}| \, \|_1$, we take $B = A^{-1}$. In each iteration we are then required to solve systems $Ay = x$ and $A^T z = \xi$.

It is less obvious that Hager's estimator can also be used to estimate the componentwise analysis 7.5.13). The problem is then to estimate a n expression of the form $\| \, |A^{-1}|g\|_\infty$, for a given vector $g > 0$. Using a clever trick devised by Arioli, Demmel and Duff [3], this can be reduced to estimating $\|B\|_1$ where

$$B = (A^{-1}G)^T, \quad G = \text{diag}\,(g_1, \ldots, g_n) > 0.$$

We have $g = Ge$ where $e = (1, 1, \ldots, 1)^T$ and hence

$$\| \, |A^{-1}|g\|_\infty = \| \, |A^{-1}|Ge\|_\infty = \| \, |A^{-1}G|e\|_\infty = \| \, |A^{-1}G|\|_\infty = \|(A^{-1}G)^T\|_1,$$

where in the last step we have used that the ∞ norm is absolute (see Sec. 7.1.5). Since Bx and $B^T y$ can be found by solving linear systems involving A^T and A the work involved is similar to that of the LINPACK estimator. This together with ω determined by (7.5.10) gives an approximate bound for the error in a computed solution $\bar{x}$. Hager's condition estimator is used MATLAB .

We note that the unit lower triangular matrices L obtained from Gaussian elimination with pivoting are not arbitrary but their off-diagonal elements satisfy $|l_{ij}| \leq 1$. When Gaussian elimination without pivoting is applied to a row diagonally dominant matrix it gives a row diagonally dominant upper triangular factor $U \in \mathbf{R}^{n\times n}$ satisfying

$$|u_{ii}| \geq \sum_{j=i+1}^{n} |u_{ij}|, \quad i = 1 : n-1. \tag{7.5.18}$$

and it holds that $cond(U) \leq 2n - 1$; (see [41, Lemma 8.8].

Definition 7.5.5. *For any triangular matrix T the* **comparison matrix** *is*

$$M(T) = (m_{ij}), \qquad m_{ij} = \begin{cases} |t_{ii}|, & i = j; \\ -|t_{ij}|, & i \neq j; \end{cases}$$

Review Questions

1. How is the condition number $\kappa(A)$ of a matrix A defined? How does $\kappa(A)$ relate to perturbations in the solution x to a linear system $Ax = b$, when A and b are perturbed? Outline roughly a cheap way to estimate $\kappa(A)$.

Problems

1. (a) Compute the inverse A^{-1} of the matrix A in Problem 6.4.1 and determine the solution x to $Ax = b$ when $b = (4, 3, 3, 1)^T$.

(b) Assume that the vector b is perturbed by a vector δb such that $\|\delta b\|_\infty \leq 0.01$. Give an upper bound for $\|\delta x\|_\infty$, where δx is the corresponding perturbation in the solution.

(c) Compute the condition number $\kappa_\infty(A)$, and compare it with the bound for the quotient between $\|\delta x\|_\infty/\|x\|_\infty$ and $\|\delta b\|_\infty/\|b\|_\infty$ which can be derived from (b).

2. Show that the matrix A in Example 7.5.2 has the inverse

$$A^{-1} = 10^8 \begin{pmatrix} 0.1441 & -0.8648 \\ -0.2161 & 1.2969 \end{pmatrix},$$

and that $\kappa_\infty = \|A\|_\infty \|A^{-1}\|_\infty = 2.1617 \cdot 1.5130 \cdot 10^8 \approx 3.3 \cdot 10^8$, which shows that the system is "perversely" ill-conditioned.

3. (Higham [41, p. 144]) Consider the triangular matrix

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

Show that $\text{cond}\,(U) = 5$ but $\text{cond}\,(U^T) = 1 + 2/\epsilon$. This shows that a triangular system can be much worse conditioned than its transpose.

4. Let the matrix $A \in \mathbf{R}^{n\times n}$ be nonnegative, and solve $A^T x = e$, where $e = (1, 1, \ldots, 1)^T$. Show that then $\|A^{-1}\|_1 = \|x\|_\infty$.

5. Let $\bar{x}$ be a computed solution and $r = b - A\bar{x}$ the corresponding residual. Assume that δA is such that $(A + \delta A)\bar{x} = b$ holds exactly. Show that the error of minimum l_1-norm and l_∞-norm respectively are given by

$$\delta A = r(s_1, \ldots, s_n)/\|\bar{x}\|_1, \quad \delta A = r(0, \ldots, 0, s_m, 0, \ldots, 0)/\|\bar{x}\|_\infty,$$

where $\|\bar{x}\|_\infty = |x_m|$, and $s_i = 1$, if $x_i \geq 0$; $s_i = -1$, if $x_i < 0$.

7.6 Rounding Error Analysis

7.6.1 Floating Point Arithmetic

In this section we first recall some basic results for floating point computations. For a detailed treatment of IEEE floating point standard and floating point arithmetic we refer Sections 2.2–2.3, Volume I.

If x and y are two floating point numbers, we denote by

$$fl\,(x+y), \quad fl\,(x-y), \quad fl\,(x\cdot y), \quad fl\,(x/y)$$

the results of floating addition, subtraction, multiplication, and division, which the machine stores in memory (after rounding or chopping). We will in the following assume that underflow or overflow does not occur. and that the following **standard model** for the arithmetic holds:

Definition 7.6.1.
Assume that $x, y \in F$. Then in the **standard model** *it holds*

$$fl\,(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \le u, \tag{7.6.1}$$

where u is the unit roundoff and "op" stands for one of the four elementary operations $+$, $-$, $\cdot$, and $/$.

The standard model holds with the default rounding mode for computers implementing the IEEE 754 standard. In this case we also have

$$fl\,(\sqrt{x}) = \sqrt{x}(1+\delta), \quad |\delta| \le u, \tag{7.6.2}$$

Bounds for roundoff errors for basic vector and matrix operations can easily be derived (Wilkinson [66, pp. 114–118]) using the following basic result:

Lemma 7.6.2. [N. J. Higham [41, Lemma 3.1]]
Let $|\delta_i| \le u$, $\rho_i = \pm 1$, $i = 1{:}n$, and

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n.$$

If $nu < 1$, then $|\theta_n| < \gamma_n$, where $\gamma_n = nu/(1-nu)$.

For an **inner product** $x^T y$ computed in the natural order we have

$$fl\,(x^T y) = x_1y_1(1+\delta_1) + x_2y_2(1+\delta_2) + \cdots + x_ny_n(1+\delta_n)$$

where

$$|\delta_1| < \gamma_n, \qquad |\delta_r| < \gamma_{n+2-i}, \quad i = 2:n.$$

The corresponding forward error bound becomes

$$|fl\,(x^T y) - x^T y| < \sum_{i=1}^{n} \gamma_{n+2-i}|x_i||y_i| < \gamma_n |x^T||y|, \tag{7.6.3}$$

where $|x|$, $|y|$ denote vectors with elements $|x_i|$, $|y_i|$.This bound is independent of the summation order and is valid also for floating point computation with no guard digit rounding.

For the outer product xy^T of two vectors $x, y \in \mathbf{R}^n$ it holds that $fl\,(x_i y_j) = x_i y_j(1+\delta_{ij})$, $\delta_{ij} \le u$, and so

$$|fl\,(xy^T) - xy^T| \le u\,|xy^T|. \tag{7.6.4}$$

This is a satisfactory result for many purposes, but the computed result is not in general a rank one matrix and it is not possible to find perturbations Δx and Δy such that $fl(xy^T) = (x+\Delta x)(x+\Delta y)^T$. This shows that matrix multiplication in floating point arithmetic is not always backward stable!

Similar error bounds can easily be obtained for matrix multiplication. Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, and denote by $|A|$ and $|B|$ matrices with elements $|a_{ij}|$ and $|b_{ij}|$. Then it holds that

$$|fl\,(AB) - AB| < \gamma_n |A||B|. \tag{7.6.5}$$

where the inequality is to be interpreted elementwise. Often we shall need bounds for some norm of the error matrix. From (7.6.5) it follows that

$$\|fl\,(AB) - AB\| < \gamma_n \| \,|A|\,\| \,\| \,|B|\,\|. \tag{7.6.6}$$

Hence, for the 1-norm, ∞-norm and the Frobenius norm we have

$$\|fl\,(AB) - AB\| < \gamma_n \|A\|\,\|B\|. \tag{7.6.7}$$

but unless A and B have non-negative elements, we have for the 2-norm only the weaker bound

$$\|fl\,(AB) - AB\|_2 < n\gamma_n \|A\|_2\,\|B\|_2. \tag{7.6.8}$$

In many matrix algorithms there repeatedly occurs expressions of the form

$$y = \Big(c - \sum_{i=1}^{k-1} a_i b_i\Big)\Big/ d.$$

A simple extension of the roundoff analysis of an inner product in Sec, 2.4.1 (cf. Problem 2.3.7) shows that if the term c is added last, then the computed $\bar{y}$ satisfies

$$\bar{y}d(1+\delta_k) = c - \sum_{i=1}^{k-1} a_i b_i (1+\delta_i), \tag{7.6.9}$$

where

$$|\delta_1| \le \gamma_{k-1}, \qquad |\delta_i| \le \gamma_{k+1-i}, \quad i = 2:k-1, \qquad |\delta_k| \le \gamma_2. \tag{7.6.10}$$

and $\gamma_k = ku/(1-ku)$ and u is the unit roundoff. Note that in order to prove a backward error result for GE, that does not perturb the right hand side vector b,

we have formulated the result so that c is not perturbed. It follows that the forward error satisfies

$$\left|\bar{y}d - c + \sum_{i=1}^{k-1} a_i b_i\right| \leq \gamma_k\Big(|\bar{y}d| + \sum_{i=1}^{k-1} |a_i||b_i|\Big), \tag{7.6.11}$$

and this inequality holds independent of the summation order.

7.6.2 Error Analysis of Gaussian Elimination

In the practical solution of a linear system of equations, rounding errors are introduced in each arithmetic operation and cause errors in the computed solution. In the early days of the computer era around 1946 many mathematicians were pessimistic about the numerical stability of Gaussian elimination. It was argued that the growth of roundoff errors would make it impractical to solve even systems of fairly moderate size. By the early 1950s experience revealed that this pessimism was unfounded. In practice Gaussian elimination with partial pivoting is a remarkably stable method and has become the universal algorithm for solving dense systems of equations.

The bound given in Theorem 7.2.3 is satisfactory only if the growth factor ρ_n is not too large, but this quantity is only known *after* the elimination has been completed. In order to obtain an *a priori bound* on ρ_n we use the inequality

$$|a_{ij}^{(k+1)}| = |a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}| \leq |a_{ij}^{(k)}| + |a_{kj}^{(k)}| \leq 2\max_k |\bar{a}_{ij}^{(k)}|,$$

valid If partial pivoting is employed. By induction this gives the upper bound $\rho_n \leq 2^{n-1}$, which is attained for matrices $A_n \in \mathbf{R}^{n\times n}$ of the form

$$A_4 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}. \tag{7.6.12}$$

Already for $n = 54$ we can have $\rho_n = 2^{53} \approx 0.9 \cdot 10^{16}$ and can lose all accuracy using IEEE double precision ($u = 1.11 \cdot 10^{-16}$). Hence the *worst-case behavior of partial pivoting is very unsatisfactory.*

For complete pivoting, Wilkinson [65, 1961] has proved that

$$\rho_n \leq (n \cdot 2^1 3^{1/2} 4^{1/3} \cdots n^{1/(n-1)})^{1/2} < 1.8\sqrt{n}n^{\frac{1}{4}\log n},$$

and that this bound is not attainable. This bound is *much* smaller than that for partial pivoting, e.g., $\rho_{50} < 530$. It was long conjectured that $\rho_n \leq n$ for real matrices and complete pivoting. This was finally disproved in 1991 when a matrix of order 13 was found for which $\rho_n = 13.0205$. A year later a matrix of order 25 was found for which $\rho_n = 32.986$.

Although complete pivoting has a much smaller worst case growth factor than partial pivoting it is more costly. Moreover, complete (as well as rook) pivoting has the drawback that it cannot be combined with the more efficient blocked methods of GE (see Sec. 7.5.3). Fortunately from decades of experience and extensive

experiments it can be concluded that substantial growth in elements using partial pivoting occurs only for a tiny proportion of matrices arising naturally. We quote Wilkinson [66, pp. 213–214].

> *It is our experience that any substantial increase in the size of elements of successive $A^{(k)}$ is extremely uncommon even with partial pivoting. No example which has arisen naturally has in my experience given an increase by a factor as large as* 16.

So far only two exceptions to the experience related by Wilkinson have been reported. One concerns linear systems arising from a class of two-point boundary value problems, when solved by the shooting method. Another is the class of linear systems arising from a quadrature method for solving a certain Volterra integral equation. These examples show that GE with partial pivoting cannot be unconditionally trusted. When in doubt some safeguard like monitoring the element growth should be incorporated. Another way of checking and improving the reliability of GE with partial pivoting is iterative refinement, which is discussed in Sec. 7.7.4.

Why large element growth rarely occurs with partial pivoting is still not fully understood. Trefethen and Schreiber [62] have shown that for certain distributions of random matrices the average element growth was close to $n^{2/3}$ for partial pivoting.

We now give a component-wise roundoff analysis for the LU factorization of A. Note that all the variants given in Sec. 7.2 for computing the LU factorization of a matrix will essentially lead to the same error bounds, since each does the same operations with the same arguments. Note also that since GE with pivoting is equivalent to GE without pivoting on a permuted matrix, we need not consider pivoting.

Theorem 7.6.3.

If the LU factorization of A runs to completion then the computed factors $\bar{L}$ and $\bar{U}$ satisfy

$$A + E = \bar{L}\bar{U}, \quad |E| \le \gamma_n |\bar{L}|\,|\bar{U}|, \tag{7.6.13}$$

where $\gamma_n = nu/(1-nu)$.

Proof. In the algorithms in Sec 7.2.6) we set $l_{ii} = 1$ and compute the other elements in L and U from the equations

$$u_{ij} = a_{ij} - \sum_{p=1}^{i-1} l_{ip}u_{pj}, \quad j \ge i;$$

$$l_{ij} = \Big(a_{ij} - \sum_{p=1}^{j-1} l_{ip}u_{pj}\Big)/u_{jj}, \quad i > j,$$

Using (7.6.11) it follows that the computed elements $\bar{l}_{ip}$ and $\bar{u}_{pj}$ satisfy

$$\Big|a_{ij} - \sum_{p=1}^{r} \bar{l}_{ip}\bar{u}_{pj}\Big| \le \gamma_r \sum_{p=1}^{r} |\bar{l}_{ip}|\,|\bar{u}_{pj}|, \quad r = \min(i,j).$$

where $\bar{l}_{ii} = l_{ii} = 1$. These inequalities may be written in matrix form □

To prove the estimate the error in a computed solution $\bar{x}$ of a linear system given in Theorem 7.6.3, we must also take into account the rounding errors performed in the solution of the two triangular systems $\bar{L}y = b$, $\bar{U}x = y$. A lower triangular system $Ly = b$ is solved by forward substitution

$$l_{kk}y_k = b_k - \sum_{i=1}^{k-1} l_{ki}y_i, \quad k = 1:n.$$

If we let $\bar{y}$ denote the computed solution, then using (7.6.9) –(7.6.10) it is straightforward to derive a bound for the backward error in solving a triangular system of equations.

Using the bound for the backward error the forward error in solving a triangular system can be estimated. It is a well known fact that the computed solution is far accurate than predicted by the normwise condition number. This has been partly explained by Stewart [60, p. 231] as follows:

> "When a matrix is decomposed by GE with partial pivoting for size, the resulting L-factor tends to be well conditioned while any ill-conditioning in the U-factor tends to be artificial."

This observation does not hold in general, for counter examples exist. However, it is true of many special kinds of triangular matrices.

Theorem 7.6.4. *If the lower triangular system $Ly = b$, $L \in \mathbf{R}^{n\times n}$ is solved by substitution with the summation order outlined above, then the computed solution $\bar{y}$ satisfies*

$$(L + \Delta L)\bar{y} = b, \quad |\Delta l_{ki}| \le \begin{cases} \gamma_2|l_{ki}|, & i = k \\ \gamma_{k+1-i}|l_{ki}|, & i = 1:k-1 \end{cases}, \quad k = 1:n. \tag{7.6.14}$$

Hence $|\Delta L| \le \gamma_n|L|$ and this inequality holds for any summation order.

An analogue result holds for the computed solution to an upper triangular systems. We conclude the backward stability of substitution for solving triangular systems. Note that it is not necessary to perturb the right hand side.

Theorem 7.6.5.

Let $\bar{x}$ be the computed solution of the system $Ax = b$, using LU factorization and substitution. Then $\bar{x}$ satisfies exactly

$$(A + \Delta A)\bar{x} = b, \tag{7.6.15}$$

where δA is a matrix, depending on both A and b, such that

$$|\Delta A| \le \gamma_n(3 + \gamma_n)|\bar{L}|\,|\bar{U}|. \tag{7.6.16}$$

Proof. From Theorem 7.6.4 it follows that the computed $\bar{y}$ and $\bar{x}$ satisfy

$$(\bar{L} + \delta\bar{L})\bar{y} = b, \qquad (\bar{U} + \delta\bar{U})\bar{x} = \bar{y},$$

where

$$|\delta\bar{L}| \le \gamma_n|\bar{L}|, \qquad |\delta\bar{U}| \le \gamma_n|\bar{U}|. \tag{7.6.17}$$

Note that $\delta\bar{L}$ and $\delta\bar{U}$ depend upon b. Combining these results, it follows that the computed solution $\bar{x}$ satisfies

$$(\bar{L} + \delta\bar{L})(\bar{U} + \delta\bar{U})\bar{x} = b,$$

and using equations (7.6.13)–(7.6.17) proves the backward error

$$|\Delta A| \le \gamma_n(3 + \gamma_n)|\bar{L}|\,|\bar{U}|. \tag{7.6.18}$$

for the computed solution $\bar{x}$ given in Theorem 7.6.3. □

Note that although the perturbation δA depends upon b the *bound* on $|\delta A|$ is independent on b. The elements in $\bar{U}$ satisfy $|\bar{u}_{ij}| \le \rho_n\|A\|_\infty$, and with partial pivoting $|\bar{l}_{ij}| \le 1$. Hence

$$\|\,|\bar{L}|\,|\bar{U}|\,\|_\infty \le \tfrac{1}{2}n(n+1)\rho_n,$$

and neglecting terms of order $O((nu)^2)$ in (7.6.18) it follows that

$$\|\delta A\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty. \tag{7.6.19}$$

By taking b to be the columns $e_1, e_2, \ldots . e_n$ of the unit matrix in succession we obtain the n computed columns of the inverse X of A. For the kth column we have

$$(A + \Delta A_r)\bar{x}_r = e_r,$$

where we have written ΔA_r to emphasize that the perturbation *is different for each column*. Therefore we cannot say that GE computes the exact inverse corresponding to some matrix $A + \Delta A$! We obtain the estimate

$$\|A\bar{X} - I\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty\|\bar{X}\|_\infty. \tag{7.6.20}$$

From $A\bar{X} - I = E$ it follows that $\bar{X} - A^{-1} = A^{-1}E$ and $\|\bar{X} - A^{-1}\|_\infty \le \|A^{-1}\|_\infty\|E\|_\infty$, which together with (7.6.20) can be used to get a bound for the error in the computed inverse. We should stress again that we recommend that computing explicit inverses is avoided.

The residual for the computed solution satisfies $\bar{r} = b - A\bar{x} = \delta A\bar{x}$, and using (7.6.19) it follows that

$$\|\bar{r}\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty\|\bar{x}\|_\infty.$$

This shows the remarkable fact that *GE will give a small relative residual even for ill-conditioned systems.* Unless the growth factor is large the quantity

$$\|b - A\bar{x}\|_\infty/(\|A\|_\infty\|\bar{x}\|_\infty)$$

will in practice be of the order nu. It is important to realize that this property of GE is not shared by most other methods for solving linear systems. For example, if we first compute the inverse A^{-1} and then $x = A^{-1}b$ the residual $\bar{r}$ may be much larger even if the accuracy in $\bar{x}$ is about the same.

The error bound in Theorem 7.6.3 is instructive in that it shows that a particularly favourable case is when $|\bar{L}|\,|\bar{U}| = |\bar{L}\bar{U}|$. This is true when $\bar{L}$ and $\bar{U}$ are nonnegative. Then

$$|\bar{L}|\,|\bar{U}| = |\bar{L}\bar{U}| = |A + \Delta A| \leq |A| + |\bar{L}|\,|\bar{U}|,$$

and neglecting terms of order $O((nu)^2)$ we find that the computed $\bar{x}$ satisfies

$$(A + \Delta A)\bar{x} = b, \quad |\Delta A| \leq 3\gamma_n |A|.$$

A class of matrices for which Gaussian elimination without pivoting gives positive factors L and U is the following.

Definition 7.6.6.
A matrix $A \in \mathbf{R}^{n\times n}$ is called **totally positive** *if the determinant of every square submatrix of A is positive.*

It is known (see de Boor and Pinkus [15]) that if A is totally positive, then it has an LU factorization with $L > 0$ and $U > 0$. Since the property of a matrix being totally positive is destroyed under row permutations, *pivoting should not be used when solving such systems.* Totally positive systems occur in spline interpolation.

In many cases there is no a priori bound for the matrix $|\bar{L}|\,|\bar{U}|$ appearing in the componentwise error analysis. It is then possible to compute its ∞-norm in $O(n^2)$ operations without forming the matrix explicitly, since

$$\| \, |\bar{L}|\,|\bar{U}| \, \|_\infty = \| \, |\bar{L}|\,|\bar{U}|e \, \|_\infty = \| \, |\bar{L}|\,(|\bar{U}|e) \, \|_\infty.$$

This useful observation is due to Chu and George [12].

An error analysis for the Cholesky factorization of a symmetric positive definite matrix $A \in \mathbf{R}^{n\times n}$ is similar to that for LU factorization.

Theorem 7.6.7.
Suppose that the Cholesky factorization of a symmetric positive definite matrix $A \in R^{n\times n}$ runs to completion and produces a computed factor $\bar{R}$ and a computed solution $\bar{x}$ to the linear system. Then it holds that

$$A + E_1 = \bar{L}\bar{U}, \quad |E_1| \leq \gamma_{n+1}|\bar{R}^T|\,|\bar{R}|, \tag{7.6.21}$$

and

$$(A + E_2)\bar{x} = b, \quad |E_2| \leq \gamma_{3n+1}|\bar{R}^T|\,|\bar{R}|. \tag{7.6.22}$$

Theorem 7.6.8. [J. H. Wilkinson [67]]

Let $A \in R^{n\times n}$ be a symmetric positive definite matrix. The Cholesky factor of A can be computed without breakdown provided that $2n^{3/2}u\kappa(A) < 0.1$. The computed $\bar{L}$ satisfies

$$\bar{L}\bar{L}^T = A + E, \quad \|E\|_2 < 2.5n^{3/2}u\|A\|_2, \tag{7.6.23}$$

and hence is the exact Cholesky factor of a matrix close to A.

This is essentially the best normwise bounds that can be obtained, although Meinguet [50] has shown that for large n the constants 2 and 2.5 in Theorem 7.6.8 can be improved to 1 and 2/3, respectively.

In practice we can usually expect much smaller backward error in the computed solutions than the bounds derived in this section. It is appropriate to recall here a remark by J. H. Wilkinson (1974):

> "All too often, too much attention is paid to the precise error bound that has been established. The main purpose of such an analysis is either to establish the essential numerical stability of an algorithm or to show why it is unstable and in doing so expose what sort of change is necessary to to make it stable. The precise error bound is not of great importance."

7.6.3 Scaling of Linear Systems

In a linear system of equations $Ax = b$ the ith equation may be multiplied by an arbitrary positive scale factor d_i, $i = 1 : n$, without changing the exact solution. In contrast, such a scaling will usually change the computed numerical solution. In this section we show that *a proper row scaling is important for GE with partial pivoting to give accurate computed solutions*, and give some rules for scaling.

We first show that *if the pivot sequence is fixed* then Gaussian elimination is unaffected by such scalings, or more precisely:

Theorem 7.6.9.

Denote by $\bar{x}$ and $\bar{x}'$ the computed solutions obtained by GE in floating point arithmetic to the two linear systems of equations

$$Ax = b, \qquad (D_r A D_c)x' = D_r b,$$

where D_r and D_c are diagonal scaling matrices. Assume that the elements of D_r and D_c are powers of the base of the number system used, so that no rounding errors are introduced by the scaling. Then if the same pivot sequence is used and no overflow or underflow occurs we have exactly $\bar{x} = D_c\bar{x}'$, i.e., the components in the solution differ only in the exponents.

Proof. The proof follows by examination of the scaling invariance of the basic step in Algorithm 7.2.2

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)} a_{kj}^{(k)})/a_{kk}^{(k)}.$$

□

This result has the important implication that scaling will affect the accuracy of a computed solution only if it leads to a change in the selection of pivots. When partial pivoting is used the row scaling may affect the choice of pivots; indeed we can always find a row scaling which leads to *any predetermined pivot sequence*. However, since only elements in the pivotal column are compared, the choice of pivots is independent of the column scaling. Since a bad choice of pivots can give rise to large errors in the computed solution, it follows that for GE with partial pivoting to give accurate solutions *a proper row scaling is important*.

Example 7.6.1. The system $Ax = b$ in Example 7.5.1 has the solution $x = (0.9999, 0.9999)^T$, correctly rounded to four decimals. Partial pivoting will here select the element a_{11} as pivot. Using three-figure floating point arithmetic, the computed solution becomes

$$\bar{x} = (0, 1.00)^T \qquad \text{(Bad!)}.$$

If GE instead is carried out on the scaled system $\hat{A}x = \hat{b}$, then a_{21} will be chosen as pivot, and the computed solution becomes

$$\bar{x} = (1.00, 1.00)^T \qquad \text{(Good!)}.$$

From the above discussion we conclude that the need for a proper scaling is of great importance for GE to yield good accuracy. An discussed in Sec, 7.5.3, an estimate of $\kappa(A)$ is often used to access the accuracy of the computed solution. If, e.g., the perturbation bound (7.1.64) is applied to the scaled system $(D_r A D_c)x' = D_r b$

$$\frac{\|D_c^{-1}\delta x\|}{\|D_c^{-1}x\|} \leq \kappa(D_r A D_c)\frac{\|D_r \delta b\|}{\|D_r b\|}. \tag{7.6.24}$$

Hence if $\kappa(D_r A D_c)$ can be made smaller than $\kappa(A)$, then it seems that we might expect a correspondingly more accurate solution. Note however that in (7.6.24) the perturbation in x is measured in the norm $\|D_c^{-1}x\|$, and we may only have found a norm in which the error *looks* better! We conclude that the column scaling D_c should be chosen in a way that reflects the importance of errors in the components of the solution. If $|x| \approx c$, and we want the same relative accuracy in all components we may take $D_c = diag(c)$.

We now discuss the choice of row scaling. A scheme which is sometimes advocated is to choose $D_r = \text{diag}\,(d_i)$ so that each row in $D_r A$ has the same l_1-norm, i.e.,

$$d_i = 1/\|a_i^T\|_1, \quad i = 1 : n. \tag{7.6.25}$$

(Sometimes the l_∞-norm, of the rows are instead made equal.) This scaling, called **row equilibration**, can be seen to avoid the bad pivot selection in Example 7.5.1. However, suppose that through an unfortunate choice of physical units the solution x has components of widely varying magnitude. Then, as shown by the following

example, row equilibration can lead to a *worse* computed solution than if no scaling is used!

Example 7.6.2. Consider the following system

$$A = \begin{pmatrix} 3\cdot 10^{-6} & 2 & 1 \\ 2 & 2 & 2 \\ 1 & 2 & -1 \end{pmatrix}, \qquad b = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6 \\ 2 \end{pmatrix} \qquad |\epsilon| \ll 1$$

which has the exact solution $x = (1,1,1)^T$. The matrix A is *well-conditioned*, $\kappa(A) \approx 3.52$, but the choice of a_{11} as pivot leads to a disastrous loss of accuracy. Assume that through an unfortunate choice of units, the system has been changed into

$$\hat{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2\cdot 10^{6} & 2 & 2 \\ 10^{6} & 2 & -1 \end{pmatrix},$$

with exact solution $\hat{x} = (10^{-6}, 1, 1)^T$. If now the rows are equilibrated, the system becomes

$$\tilde{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2\cdot 10^{-6} & 2\cdot 10^{-6} \\ 1 & 2\cdot 10^{-6} & -10^{-6} \end{pmatrix}, \qquad \tilde{b} = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6\cdot 10^{-6} \\ 2\cdot 10^{-6} \end{pmatrix}.$$

GE with column pivoting will now choose a_{11} as pivot. Using floating point arithmetic with precision $u = 0.47\cdot 10^{-9}$ we get the computed solution of $\hat{A}x = \hat{b}$

$$\bar{x} = (0.999894122\cdot 10^{-6},\ 0.999983255,\ 1.000033489)^T.$$

This has only about four correct digits, so almost six digits have been lost!

A theoretical solution to the row scaling problem in GE with partial pivoting has been given by R. D. Skeel [56, 1979]. He shows a pivoting rule in GE should depend not only on the coefficient matrix but also on the solution. Hence separating the matrix factorization from the solution of the linear system may lead to instability. His scaling rule is based on minimizing a bound on the backward error that contains the quantity

$$\frac{\max_i(|D_r A||\bar{x}|)_i}{\min_i(|D_r A||\bar{x}|)_i}.$$

Scaling Rule: (R. D. Skeel) Assume that $\min_i(|A||x|)_i > 0$. Then scale the rows of A and b by $D_r = \operatorname{diag}(d_i)$, where

$$d_i = 1/(|A||x|)_i, \quad i = 1:n. \tag{7.6.26}$$

A measure of **ill-scaling** of the system $Ax = b$ is

$$\sigma(A,x) = \max_i(|A||x|)_i / \min_i(|A||x|)_i. \tag{7.6.27}$$

This scaling rule gives infinite scale factors for rows which satisfy $(|A||x|)_i = 0$. This may occur for sparse systems, i.e., when A (and possibly also x) has many zero components. In this case a large scale factor d_i should be chosen so that the corresponding row is selected as pivot row at the first opportunity.

Unfortunately scaling according to this rule is not in general practical, since it assumes that the solution x is at least approximately known. If the components of the solution vector x are known to be of the same magnitude then we can take $|x| = (1, \ldots, 1)^T$ in (7.6.26), which corresponds to row equilibration. Note that this assumption is violated in Example 7.6.2.

7.6.4 Iterative Refinement of Solutions

So far we have considered ways of *estimating* the accuracy of computed solutions. We now consider methods for *improving* the accuracy. Let $\bar{x}$ be any approximate solution to the linear system of equations $Ax = b$ and let $r = b - A\bar{x}$ be the corresponding residual vector. Then one can attempt to improve the solution by solving the system $A\delta = r$ for a correction δ and taking $x_c = \bar{x} + \delta$ as a new approximation. If no further rounding errors are performed in the computation of δ this is the exact solution. Otherwise this refinement process can be iterated. In floating-point arithmetic with base β this process of **iterative refinement** can be described as follows:

$$
\begin{aligned}
&s := 1; \quad x^{(s)} := \bar{x};\\
&\text{repeat}\\
&\qquad r^{(s)} := b - Ax^{(s)}; \qquad (\text{inprecision } u_2 = \beta^{-t_2})\\
&\qquad \text{solve } \; A\delta^{(s)} = r^{(s)}; \qquad (\text{inprecision } u_1 = \beta^{-t_1})\\
&\qquad x^{(s+1)} := x^{(s)} + \delta^{(s)};\\
&\qquad s := s + 1;\\
&\text{end}
\end{aligned}
$$

When $\bar{x}$ has been computed by GE this approach is attractive since we can use the computed factors $\bar{L}$ and $\bar{U}$ to solve for the corrections

$$\bar{L}(\bar{U}\delta^{(s)}) = r^{(s)}, \quad s = 1, 2, \ldots.$$

The computation of $r^{(s)}$ and $\delta^{(s)}$, therefore, only takes $n^2 + 2 \cdot \frac{1}{2}n^2 = 2n^2$ flops, which is an order of magnitude less than the $n^3/3$ flops required for the initial solution.

We note the possibility of using *extended precision* $t_2 > t_1$ for computing the residuals $r^{(s)}$; these are then rounded to single precision u_1 before solving for $\delta^{(s)}$. Since $x^{(s)}$, A and b are stored in single precision, only the accumulation of the inner product terms are in precision u_2, and no multiplications in extended precision occur. This is also called *mixed precision iterative refinement* as opposed to *fixed precision iterative refinement* when $t_2 = t_1$.

Since the product of two t digit floating point numbers can be exactly represented with at most $2t$ digits inner products can be computed in extended precision

without much extra cost. If fl_e denotes computation with extended precision and u_e the corresponding unit roundoff then the forward error bound for an inner product becomes

$$|fl\,(fl_e((x^Ty)) - x^Ty| < u|x^Ty| + \frac{nu_e}{1-nu_e/2}(1+u)|x^T||y|, \tag{7.6.28}$$

where the first term comes form the final rounding. If $|x^T||y| \leq u|x^Ty|$ then the computed inner product is almost as accurate as the correctly rounded exact result. However, since computations in extended precision are machine dependent it has been difficult to make such programs portable.[21] The recent development of Extended and Mixed Precision BLAS (Basic Linear Algebra Subroutines) (see [49]) may now make this more feasible!

In the ideal case that the rounding errors committed in computing the corrections can be neglected we have

$$x^{(s+1)} - x = (I - (\bar{L}\bar{U})^{-1}A)^s(\bar{x} - x).$$

where $\bar{L}$ and $\bar{U}$ denote the computed LU factors of A. Hence the process converges if

$$\rho = \|(I - (\bar{L}\bar{U})^{-1}A\| < 1.$$

This roughly describes how the refinement behaves in the *early stages*, if extended precision is used for the residuals. If $\bar{L}$ and $\bar{U}$ have been computed by GE using precision u_1, then by Theorem 7.2.3 we have

$$\bar{L}\bar{U} = A + E, \quad \|E\|_\infty \leq 1.5n^2\rho_n u_1\|A\|_\infty,$$

and ρ_n is the growth factor. It follows that an upper bound for the initial rate of convergence is given by

$$\rho = \|(\bar{L}\bar{U})^{-1}E\|_\infty \leq n^2\rho_n u_1\kappa(A).$$

When also rounding errors in computing the residuals $r^{(s)}$ and the corrections $\delta^{(s)}$ are taken into account, the analysis becomes much more complicated. The behavior of iterative refinement, using t_1-digits for the factorization and $t_2 = 2t_1$ digits when computing the residuals, can be summed up as follows:

1. Assume that A is not too ill-conditioned so that the first solution has some accuracy, $\|x - \bar{x}\|/\|x\| \approx \beta^{-k} < 1$ in some norm. Then the relative error diminishes by a factor of roughly β^{-k} with each step of refinement until we reach a stage at which $\|\delta_c\|/\|x_c\| < \beta^{-t_1}$, when we may say that the solution is correct to working precision.

2. In general the attainable accuracy is limited to $\min(k + t_2 - t_1, t_1)$ digits, which gives the case above when $t_2 \geq 2t_1$. Note that although the computed solution improves progressively with each iteration this is *not* reflected in a corresponding decrease in the norm of the residual, which stays about the same.

[21] It was suggested that the IEEE 754 standard should require inner products to be precisely specified, but that did not happen.

Iterative refinement can be used to compute a more accurate solution, in case A is ill-conditioned. However, unless A and b are exactly known this may not make much sense. The exact answer to a poorly conditioned problem may be no more appropriate than one which is correct to only a few places.

In many descriptions of iterative refinement it is stressed that it is essential that the residuals are computed with a higher precision than the rest of the computation, for the process to yield a more accurate solution. This is true if the initial solution has been computed by a backward stable method, such as GE with partial pivoting, and provided that the system is well scaled. However, iterative refinement using single precision residuals, *can considerably improve the quality of the solution, for example, when the system is ill-scaled*, i.e., when $\sigma(A, x)$ defined by (7.6.27) is large, or if the pivot strategy has been chosen for the preservation of sparsity, see Section 7.6.

Example 7.6.3. As an illustration consider again the badly scaled system in Example 7.6.1

$$\tilde{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2\cdot 10^{-6} & 2\cdot 10^{-6} \\ 1 & 2\cdot 10^{-6} & -10^{-6} \end{pmatrix}, \qquad \tilde{b} = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6\cdot 10^{-6} \\ 2\cdot 10^{-6} \end{pmatrix},$$

with exact solution $\tilde{x} = (10^{-6}, 1, 1)^T$. Using floating point arithmetic with unit roundoff $u = 0.47 \cdot 10^{-9}$ the solution computed by GE with partial pivoting has only about four correct digits. From the residual $r = \tilde{b} - \tilde{A}\bar{x}$ we compute the Oettli–Prager backward error $\omega = 0.28810 \cdot 10^{-4}$. The condition estimate computed by (7.5.17) is $3.00 \cdot 10^{6}$, and wrongly indicates that the loss of accuracy should be blamed on ill-conditioning.

With one step of iterative refinement using a single precision residual we get

$$\tilde{x} = \bar{x} + d = (\, 0.999999997\cdot 10^{-6} \quad 1.000000000 \quad 1.000000000\,)^T.$$

This is almost as good as for GE with column pivoting applied to the system $Ax = b$. The Oettli–Prager error bound for $\tilde{x}$ is $\omega = 0.54328 \cdot 10^{-9}$, which is close to the machine precision. Hence one step of iterative refinement sufficed to correct for the bad scaling. If the ill-scaling is worse or the system is also ill-conditioned then several steps of refinement may be needed.

The following theorem states that if GE with partial pivoting is combined with iterative refinement in single precision then the resulting method will give a small relative backward error provided that the system is not too ill-conditioned or ill-scaled.

Theorem 7.6.10. (R. D. Skeel.)

As long as the product of $\operatorname{cond}(A^{-1}) = \||A||A^{-1}|\|_\infty$ *and* $\sigma(A, x)$ *is sufficiently less than* $1/u$*, where* u *is the machine unit, it holds that*

$$(A + \delta A)x^{(s)} = b + \delta b, \quad |\delta a_{ij}| < 4n\epsilon_1|a_{ij}|, \quad |\delta b_i| < 4n\epsilon_1|b_i|, \tag{7.6.29}$$

for s large enough. Moreover, the result is often true already for $s = 2$, i.e., after only one improvement.

Proof. For exact conditions under which this theorem holds, see Skeel [57, 1980]. □

As illustrated above, GE with partial or complete pivoting may not provide all the accuracy that the data deserves. How often this happens in practice is not known. In cases where accuracy is important the following scheme, which offers improved reliability for a small cost is recommended.

1. Compute the Oettli–Prager backward error ω using (7.5.10) with $E = |A|$, $f = |b|$, by simultaneously accumulating $r = b - A\bar{x}$ and $|A||\bar{x}| + |b|$. If ω is not sufficiently small go to 2.

2. Perform one step of iterative refinement using the single precision residual r computed in step 1 to obtain the improved solution $\tilde{x}$. Compute the backward error $\tilde{\omega}$ of $\tilde{x}$. Repeat until the test on $\tilde{\omega}$ is passed.

7.6.5 Interval Matrix Computations

In order to treat multidimensional problems interval vectors $[x] = ([x_i])$ with interval components $[x_i] = [\underline{x_i}, \overline{x_i}])$, $i = 1 : n$ and interval matrices $[A] = ([a_{ij}])$ with interval elements $[a_{ij}] = [\underline{a}_{ij}, \overline{a}_{ij}]$, $i = 1 : m$, $j = 1 : n$, are introduced.

Operations between interval matrices and interval vectors are defined in an obvious manner. The interval matrix-vector product $[A][x]$ is the smallest interval vector, which contains the set $\{Ax \mid A \in [A],\ x \in [x]\}$, but normally does not coincide with it. By the inclusion property it holds that

$$\{Ax \mid A \in [A],\ x \in [x]\} \subseteq [A][x] = \left(\sum_{j=1}^{n} [a_{ij}][x_j] \right). \tag{7.6.30}$$

In general there will be an overestimation in enclosing the image with an interval vector caused by the fact that the image of an interval vector under a transformation in general is not an interval vector. This phenomenon, intrinsic to interval computations, is called the **wrapping effect**.

Example 7.6.4.

We have

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad [x] = \begin{pmatrix} [0,1] \\ [0,1] \end{pmatrix}, \quad \Rightarrow \quad A[x] = \begin{pmatrix} [0,2] \\ [-1,1] \end{pmatrix}.$$

Hence $b = (\,2 \quad -1\,)^T \in A[x]$, but there is no $x \in [x]$ such that $Ax = b$. (The solution to $Ax = b$ is $x = (\,3/2 \quad 1/2\,)^T$.)

The magnitude of an interval vector or matrix is interpreted component-wise and defined by

$$|\,[x]\,| = (|\,[x_1]\,|, |\,[x_2]\,|, \ldots, |\,[x_n]\,|)^T,$$

where the magnitude of the components are defined by

$$|\,[a,b]\,| = \max\{|x| \mid x \in [a,b]\}, \tag{7.6.31}$$

The ∞-norm of an interval vector or matrix is defined as the ∞-norm of their magnitude,

$$\|\,[x]\,\|_\infty = \|\,|\,[x]\,|\,\|_\infty, \qquad \|\,[A]\,\|_\infty = \|\,|\,[A]\,|\,\|_\infty. \tag{7.6.32}$$

In implementing matrix multiplication it is important to avoid case distinctions in the inner loops, because that would make it impossible to use fast vector and matrix operations. Using interval arithmetic it is possible to compute strict enclosures of the product of two matrices. Note that this is needed also in the case of the product of two point matrices since rounding errors will in general occur. In this case we want to compute an interval matrix $[C]$ such that $fl(A \cdot B) \subset [C] = [C_{\inf}, C_{\sup}]$. The following simple code does that using two matrix multiplications:

$$\begin{aligned} &\text{setround}(-1); \quad C_{\inf} = A \cdot B; \\ &\text{setround}(1); \quad\;\; C_{\sup} = A \cdot B; \end{aligned}$$

Here and in the following we assume that the command setround(i), $i = -1, 0, 1$ sets the rounding mode to $-\infty$, to nearest, and to $+\infty$, respectively.

We next consider the product of a point matrix A and an interval matrix $[B] = [B_{\inf}, B_{\sup}]$. The following code, suggested by A. Neumeier, performs this using four matrix multiplications:

$$\begin{aligned} &A_- = \min(A,\ 0); \quad A_+ = \max(A,\ 0); \\ &\text{setround}(-1); \\ &C_{\inf} = A_+ \cdot B_{\inf} + A_- \cdot B_{\sup}; \\ &\text{setround}(1); \\ &C_{\sup} = A_- \cdot B_{\inf} + A_+ \cdot B_{\sup}; \end{aligned}$$

(Note that the commands $A_- = \min(A,\ 0)$ and $A_+ = \max(A,\ 0)$ acts componentwise.) Rump [55] gives an algorithm for computing the product of two interval matrices using eight matrix multiplications. He also gives several faster implementations, provided a certain overestimation can be allowed.

A square interval matrix $[A]$ is called nonsingular if it does not contain a singular matrix. An interval linear system is a system of the form $[A]\,x = [b]$, where A is a nonsingular interval matrix and b an interval vector. The solution set of such an interval linear system is the set

$$\mathcal{X} = \{x \mid Ax = b,\ A \in [A],\ b \in [b]\}. \tag{7.6.33}$$

Computing this solution set can be shown to be an intractable problem (NP-complete). Even for a 2×2 linear system this set may not be easy to represent.

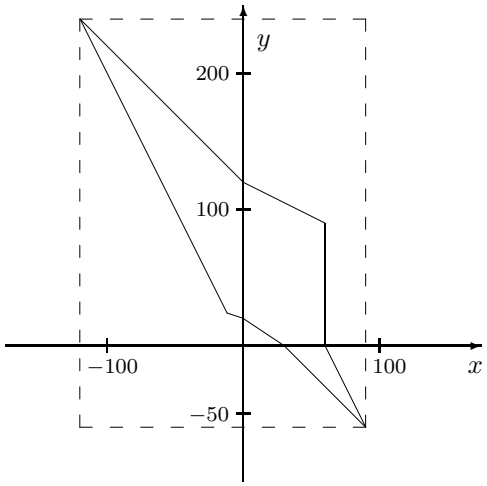

Figure 7.6.1. *The solution set (solid line) and its enclosing hull (dashed line) for the linear system in Example* 7.6.5.

Example 7.6.5. (Hansen [37, Chapter 4])
Consider a linear system with

$$[A] = \begin{pmatrix} [2,3] & [0,1] \\ [1,2] & [2,3] \end{pmatrix}, \quad [b] = \begin{pmatrix} [0,120] \\ [60,240] \end{pmatrix}. \tag{7.6.34}$$

The solution set $\mathcal{X}$ in (7.6.33) is the star shaped region in Fig. 7.6.1.

An enclosure of the solution set of an interval linear system can be computed by a generalization of Gaussian elimination adopted to interval coefficients. The solution of the resulting interval triangular system will give an inclusion of the solution set. Realistic bounds can be obtained in this way only for special classes of matrices, e.g., for diagonally dominant matrices and tridiagonal matrices; see Hargreaves [38]. For general systems this approach unfortunately tends to give interval sizes which grow exponentially during the elimination. For example, if $[x]$ and $[y]$ are intervals then in the 2×2 reduction

$$\begin{pmatrix} 1 & [x] \\ 1 & [y] \end{pmatrix} \sim \begin{pmatrix} 1 & [x] \\ 0 & [y]-[x] \end{pmatrix}.$$

If $[x] \approx [y]$ the size of the interval $[y]-[x]$ will be twice the size of $[x]$. This growth is very likely to happen. Even for well-conditioned linear systems the elimination can break down prematurely, because all remaining possible pivot elements contain zero.

A better way to compute verified bounds on a point or interval linear system uses an idea that goes back to E. Hansen [1965]. In this an approximate inverse C is used to precondition the system. Assuming that an initial interval vector $[x^{(0)}]$

is known, such that $[x^{(0)}] \supseteq \mathcal{X}$ where $\mathcal{X}$ is the solution set (7.6.33). An improved enclosure can then be obtained as follows:

By the inclusion property of interval arithmetic, for all $\tilde{A} \in [A]$ and $\tilde{b} \in [b]$ it holds that

$$\tilde{A}^{-1}\tilde{b} = C\tilde{b} + (I - C\tilde{A})\tilde{A}^{-1}\tilde{b} \in C\,[b] + (I - C\,[A])[x^{(0)}] =: [x^{(1)}].$$

This suggests the iteration known as **Krawczyck's method**

$$[x^{(i+1)}] = \Big(C\,[b] + (I - C[A])[x^{(i)}]\Big) \cap [x^{(i)}], \quad i = 0, 1, 2, \ldots, \tag{7.6.35}$$

for computing a sequence of interval enclosures $[x^{(i)}]$ of the solution. Here the interval vector $[c] = C\,[b]$ and interval matrix $[E] = I - C\,[A]$ need only be computed once. The dominating cost per iteration is one interval matrix-vector multiplication.

As approximate inverse we can take the inverse of the midpoint matrix $C = (\mathrm{mid}\,[A])^{-1}$. An initial interval can be chosen of the form

$$[x^{(0)}] = C\mathrm{mid}\,[b] + [-\beta, \beta]e, \quad e = (1, 1, \ldots, 1),$$

with β sufficiently large. The iterations are terminated when the bounds are no longer improving. A measure of convergence can be computed as $\rho = \|[E]\|_\infty$.

Rump [55, 54] has developed a MATLAB toolbox INTLAB (INTerval LABoratory). This is very efficient and easy to use and includes many useful subroutines. INTLAB uses a variant of Krawczyck's method, applied to a residual system, to compute an enclosure of the difference between the solution and an approximate solution $x_m = C\mathrm{mid}\,[b]$; see Rump [55].

Example 7.6.6.

A method for computing an enclosure of the inverse of an interval matrix can be obtained by taking $[b]$ equal to the identity matrix in the iteration (7.6.35) and solving the system $[A][X] = I$. For the symmetric interval matrix

$$[A] = \begin{pmatrix} [0.999, 1.01] & [-0.001, 0.001] \\ [-0.001, 0.001] & [0.999, 1.01] \end{pmatrix}$$

the identity $C = \mathrm{mid}\,[A] = I$ is an approximate point inverse. We find

$$[E] = I - C[A] = \begin{pmatrix} [-0.01, 0.001] & [-0.001, 0.001] \\ [-0.001, 0.001] & [-0.01, 1.001] \end{pmatrix},$$

and as an enclosure for the inverse matrix we can take

$$[X^{(0)}] = \begin{pmatrix} [0.98, 1.02] & [-0.002, 0.002] \\ [-0.002, 0.002] & [0.98, 1.02] \end{pmatrix}.$$

The iteration

$$[X^{(i+1)}] = \Big(I + E[X^{(i)}]\Big) \cap [X^{(i)}], \quad i = 0, 1, 2, \ldots.$$

converges rapidly in this case.

Review Questions

1. The result of a roundoff error analysis of Gaussian elimination can be stated in the form of a backward error analysis. Formulate this result. (You don't need to know the precise expression of the constants involved.)

2. (a) Describe the main steps in iterative refinement with extended precision for computing more accurate solutions of linear system.

(b) Sometimes it is worthwhile to do a step of iterative refinement in using fixed precision. When is that?

Problems

1. Compute the LU factors of the matrix in (7.6.12).

7.7 Block Algorithms for Gaussian Elimination

7.7.1 Block and Blocked Algorithms

In block matrix algorithms most part of the arithmetic operations consists of matrix multiplications. Because of this these algorithms can achieve high performance on modern computers. The following distinction between two different classes of algorithms is important to emphasize, since they have different stability properties.

As a first example, consider the inverse of a block lower triangular matrix

$$L = \begin{pmatrix} L_{11} & & & \\ L_{21} & L_{22} & & \\ \vdots & \vdots & \ddots & \\ L_{n,1} & L_{n,2} & \cdots & L_{nn} \end{pmatrix}, \tag{7.7.1}$$

If the diagonal blocks L_{ii}, $i = 1:2$, are nonsingular, it is easily verified that the inverse also will be block lower triangular,

$$L^{-1} = \begin{pmatrix} Y_{11} & & & \\ Y_{21} & Y_{22} & & \\ \vdots & \vdots & \ddots & \\ Y_{n,1} & Y_{n,2} & \cdots & Y_{nn} \end{pmatrix}, \tag{7.7.2}$$

In Sec. 7.1.5 we showed that the inverse in the 2×2 case is

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}.$$

Note that we do not assume that the diagonal blocks are lower triangular.

In the general case the blocks in the inverse can be computed a block column at a time from a straightforward extension of the scalar algorithm (7.2.40). Identifying blocks in the jth block column, of the equation $LY = I$, we for $j = 1 : n$,

$$L_{jj}Y_{jj} = I, \qquad L_{ii}Y_{ij} = -\sum_{k=j}^{i-1} L_{ik}Y_{kj}, \quad i = j+1 : n. \tag{7.7.3}$$

These equations can be solved for $Y_{jj}, \ldots, Y_{nj}$, by the scalar algorithms described in Sec. 7.2. The main arithmetic work will take place in the matrix-matrix multiplications $L_{ik}Y_{kj}$. This is an example of a true **block algorithm**, which is a obtained by substituting in a scalar algorithm operations on blocks of partitioned matrices regarded as non-commuting scalars.

In the special case that L is a lower triangular matrix this implies that all diagonal blocks L_{ii} and Y_{ii}, $i = 1 : n$, are lower triangular. In this case the equations in (7.7.3) can be solved by back-substitution. The resulting algorithm is then just a scalar algorithm in which the operations have been grouped and reordered into matrix operations. Such an algorithm is called a **blocked algorithm**. Blocked algorithms have the same stability properties as their scalar counterparts. This is not true for general block algorithms, which is why the distinction is important to make.

In Sec. 7.1.5 we gave, using slightly different notations, the block LU factorization

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & S \end{pmatrix}, \tag{7.7.4}$$

for a 2×2 block matrix, with square diagonal blocks. Here $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement. Note that the diagonal blocks in the block lower triangular factor in (7.7.4) are the identity matrix. Hence, this is a true block algorithm.

In a blocked LU factorization algorithm, the LU factors should have the form

$$A = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

where L_{11}, L_{22} are unit lower triangular and U_{11} , U_{22} are upper triangular. Such a factorization can be computes as follows. We first compute the scalar LU factorization $A_{11} = L_{11}U_{11}$, and then compute

$$L_{21} = A_{21}U_{11}^{-1}, \qquad U_{12} = L_{11}^{-1}A_{12}, \qquad S_{22} = A_{22} - L_{21}U_{12}.$$

Finally compute the scalar factorization $S_{22} = L_{22}U_{22}$.

In the general case a blocked algorithm for the LU factorization of a block matrix

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1N} \\ A_{21} & A_{22} & \ldots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \ldots & A_{NN} \end{pmatrix}, \tag{7.7.5}$$

having square diagonal blocks. Let L and U be partitioned conformally with A. Equating blocks in the product $A = LU$, we obtain, assuming that all inverses exist, the following block LU algorithm:

Algorithm 7.7.1 Blocked LU Factorization.

for $k = 1 : N$

$$S_{kk} = A_{kk} - \sum_{p=1}^{k-1} L_{kp}U_{pk};$$

$$S_{kk} = L_{kk}U_{kk}$$

for $j = k + 1 : N$

$$L_{jk} = \Big(A_{jk} - \sum_{p=1}^{k-1} L_{jp}U_{pk}\Big)U_{kk}^{-1};$$

end

for $j = 1 : k - 1$

$$U_{jk} = L_{kk}^{-1}\Big(A_{jk} - \sum_{p=1}^{k-1} L_{jp}U_{pj}\Big);$$

end

end

Here the LU-decompositions $S_{kk} = L_{kk}U_{kK}$ of the modified diagonal blocks are computed by a scalar LU factorization algorithm. However, the dominating part of the work is performed in matrix-matrix multiplications. The inverse of the triangular matrices L_{kk}^{-1} and U_{kk}^{-1} are *not* formed but the off-diagonal blocks U_{kj} and L_{jk} (which in general are full matrices) are computed by triangular solves. Pivoting can be used in the factorization of the diagonal blocks. As described the algorithm does not allow for row interchanges between blocks. This point is addressed In the next section.

As with the scalar algorithms there are many possible ways of sequencing the block factorization. The block algorithm above computes in the kth major step the kth block column of L and U. In this variant at step k only the kth block column of A is accessed, which is advantageous from the standpoint of data access.

A block LU factorization algorithm differs from the blocked algorithm above in that the lower block triangular matrix L has diagonal blocks equal to unity. Although such a block algorithm may have good numerical stability properties this cannot be taken for granted, since in general they do not perform the same arithmetic operations as in the corresponding scalar algorithms. It has been shown that block LU factorization can fail even for symmetric positive definite and row diagonally dominant matrices.

One class of matrices for which the block LU algorithm is known to be stable is block tridiagonal matrices that are **block diagonally dominant**. .

Definition 7.7.1. *(see Demmel et al. [16])*

A general matrix $A \in \mathbf{R}^{n\times n}$ is said to be block diagonally dominant by columns, with respect to a given partitioning, if it holds i.e.

$$\|A_{jj}^{-1}\|^{-1} \geq \sum_{i\neq j} \|A_{ij}\|, \quad j = 1 : n. \tag{7.7.6}$$

A is block **diagonally dominant** *by rows, if A^T is (strictly) diagonally dominant by columns.*

Note that for block size 1 the usual property of (point) diagonal dominance is obtained. For the 1 and ∞-norms diagonal dominance does not imply block diagonal dominance Neither does and the reverse implications hold.

Analogous to the Block LU Algorithm in Section 7.7.1 block versions of the Cholesky algorithm can be developed. If we assume that A has been partitioned into $N \times N$ blocks with square diagonal blocks we get using a block column-wise order:

Algorithm 7.7.2 Blocked Cholesky Algorithm.

$$
\begin{aligned}
&\textbf{for } j = 1 : N \\
&\quad S_{jj} = A_{jj} - \sum_{k=1}^{j-1} R_{jk}^T R_{jk}; \\
&\quad S_{jj} = R_{jj}^T R_{jj} \\
&\quad \textbf{for } i = j+1 : N \\
&\qquad R_{ij}^T = \Big(A_{ij} - \sum_{k=1}^{j-1} R_{ik}^T R_{jk} \Big) (R_{jj})^{-1}; \\
&\quad \textbf{end} \\
&\textbf{end}
\end{aligned}
$$

Note that the diagonal blocks R_{jj} are obtained by computing the Cholesky factorizations of matrices of smaller dimensions. The right multiplication with $(R_{jj})^{-1}$ in the computation of R_{jk}^T is performed by solving the triangular equations of the form $R_{jj}^T R_{ij} = S^T$. The matrix multiplications dominate the arithmetic work in the block Cholesky algorithm.

In deriving the block LU and Cholesky algorithms we assumed that the block sizes were determined in advance. However, this is by no means necessary. A more flexible way is to advance the computation by deciding at each step the size of the current pivot block. The corresponding blocked formulation then uses a 3×3 block structure, but the partitioning changes after each step.

Suppose that a partial LU factorization so that

$$
P_1 A = \begin{pmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & \tilde{A}_{22} & \tilde{A}_{23} \\ & \tilde{A}_{32} & \tilde{A}_{33} \end{pmatrix}.
$$

has been obtained, where P_1 is a permutation matrix and L_{11}, $U_{11} \in \mathbf{R}^{n_1 \times n_1}$. To advance the factorization compute the LU factorization with row pivoting

$$
P_2 \begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22},
$$

where L_{22}, $U_{22} \in \mathbf{R}^{n_2 \times n_2}$. The permutation matrix P_2 has to be applied also to

$$\begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix} := P_2 \begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix}, \qquad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} := P_2 \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix}.$$

We then solve for U_{23} and update A_{33} using

$$L_{22}U_{23} = \tilde{A}_{23}, \qquad \tilde{A}_{33} = A_{33} - L_{32}U_{23}.$$

The factorization has now been advanced one step to become

$$P_2PA = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & A_{33} \end{pmatrix}.$$

We can now repartition so that the first two block-columns in L are joined into a block of $n_1 + n_2$ columns and similarly the first two block-rows in U joined into one block of $n_1 + n_2$ rows. The blocks I and A_{33} in L and U are partitioned into 2×2 block matrices and we advance to the next block-step. This describes the complete algorithm since we can start the algorithm by taking $n_1 = 0$.

The above algorithm is sometimes called **right-looking**, referring to the way in which the data is accessed. The corresponding **left-looking** algorithm goes as follows. Assume that we have computed

$$PA = \begin{pmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} U_{11} & A_{12} & A_{13} \\ & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{pmatrix}.$$

To advance the factorization we solve first a triangular system $L_{11}U_{12} = A_{12}$ to obtain U_{12} and then compute

$$\begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{12},$$

We then compute the LU factorization with row pivoting

$$P_2 \begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22},$$

and replace

$$\begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix} = P_2 \begin{pmatrix} A_{23} \\ A_{33} \end{pmatrix}, \qquad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} := P_2 \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix}.$$

The factorization has now been advanced one step to become

$$P_2PA = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & A_{13} \\ & U_{22} & A_{23} \\ & & A_{33} \end{pmatrix}.$$

Note that in this version the blocks in the last block column of A are referenced only in the pivoting operation, but this can be postponed.

Block LU factorizations appears to have been first proposed for **block tridiagonal** matrices, which often arise from the discretization of partial differential equations. For a symmetric positive definite matrix the recursion (7.4.11) is easily generalized to compute the following block-factorization:

$$A = U^T D^{-1} U, \quad D = \operatorname{diag}(\Sigma_1, \ldots, \Sigma_n),$$

of a symmetric positive definite **block-tridiagonal** matrix with square diagonal blocks. We obtain

$$A = \begin{pmatrix} D_1 & A_2^T & & & \\ A_2 & D_2 & A_3^T & & \\ & A_3 & \ddots & \ddots & \\ & & \ddots & \ddots & A_N^T \\ & & & A_N & D_N \end{pmatrix}, \qquad U^T = \begin{pmatrix} \Sigma_1 & & & & \\ A_2 & \Sigma_2 & & & \\ & A_3 & \ddots & & \\ & & \ddots & \ddots & \\ & & & A_N & \Sigma_N \end{pmatrix},$$

where

$$\Sigma_1 = D_1, \qquad \Sigma_k = D_k - A_k \Sigma_{k-1}^{-1} A_k^T, \quad k = 2 : N. \tag{7.7.7}$$

To perform the operations with Σ_k^{-1}, $k = 1 : N$ the Cholesky factorization of these matrices are computed by a scalar algorithm. After this factorization has been computed the solution of the system

$$Ax = U^T D^{-1} U x = b$$

can be obtained by block forward- and back-substitution $U^T z = b$, $Ux = Dz$.

Note that the blocks of the matrix A may again have band-structure, which should be taken advantage of! A similar algorithm can be developed for the unsymmetric block-tridiagonal case.

For block tridiagonal matrices the following result is known:

Theorem 7.7.2. *(*Varah [64]*)*

Let the matrix $A \in \mathbf{R}^{n\times n}$ be block tridiagonal and have the block LU factorization $A = LU$, where L and U are block bidiagonal, and normalized so that $U_{i,i+1} = A_{i,i+1}$. Then if A is block diagonally dominant by columns

$$\|L_{i,i-1}\| \le 1, \qquad \|U_{i,i}\| \le \|A_{i,i}\| + \|A_{i-1,i}\|. \tag{7.7.8}$$

If A is block diagonally dominant by rows

$$\|L_{i,i-1}\| \le \frac{\|A_{i-1,i}\|}{\|A_{i,i-1}\|}, \qquad \|U_{i,i}\| \le \|A_{i,i}\| + \|A_{i-1,i}\|. \tag{7.7.9}$$

These results can be extended to full block diagonally dominant matrices, by using the key property that block diagonal dominance is inherited by the Schur complements obtained in the factorizations.

7.7.2 Recursive Algorithms

In the 2×2 case the block LU factorization Algorithm 7.7.1 is obtained by equating

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}. \tag{7.7.10}$$

We get

$$\begin{aligned} &L_{11}U_{11} = A_{11}, \\ &L_{21} = A_{21}U_{11}^{-1}, \qquad U_{12} = L_{11}^{-1}A_{12}, \\ &\tilde{A}_{22} = A_{22} - L_{21}U_{12}, \\ &L_{22}U_{22} = \tilde{A}_{22}. \end{aligned}$$

Hence the LU factorization of A can be reduced to the LU factorization of two smaller matrices A_{11} and $\tilde{A}_{22}$, two triangular solves with matrix right hand sides, and one matrix update $A_{22} - L_{21}U_{12}$.

Similarly for the Cholesky factorization equating

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}, \tag{7.7.11}$$

gives

$$\begin{aligned} &L_{11}L_{11}^T = A_{11}, \\ &L_{21}^T = L_{11}^{-1}A_{21}^T, \\ &L_{22}L_{22}^T = A_{22} - L_{21}L_{21}^T. \end{aligned}$$

It is possible to derive "divide and conquer" algorithms for the LU and Cholesky algorithms, by using the 2×2 block versions recursively. see [24].

Algorithm 7.7.3 Recursive Cholesky Factorization.

Let $A \in \mathbf{R}^{n \times n}$ be a symmetric positive definite matrix. The following recursive algorithm computes the Cholesky factorization of A.

```
function L = rchol(A);
[n, n] = size(A);
if n ≠ 1
%RecursiveCholesky
   k = floor(n/2)
   L11 = rchol(A(1 : k, 1 : k));
   L21 = (L11^{-1} A(1 : k, k + 1 : n))';
   L22 = rchol(A(1 : k, 1 : k) − L21 ∗ L21');
   L = [L11 zeros(k, n − k); L21 L22];
```

else
$L = sqrt(A)$;
end;

This is not a toy algorithm, but can be developed into an efficient algorithm for parallel high performance computers!

An intriguing question is whether it is possible to multiply two matrices $A \in \mathbf{R}^{m\times n}$, and $B \in \mathbf{R}^{n\times p}$ in less than mnp (scalar) multiplications. The answer is yes! Strassen [1969] developed a fast algorithm for matrix multiplication based on the following method for multiplying 2×2 block matrices. Assume that m, n, p are even and partition each of A, B and the product $C \in \mathbf{R}^{m\times p}$, into four equally sized blocks. Then, as can be verified by substitution, the product $C = AB$ can be computed using the following formulas:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 + P_3 - P_2 + P_6 \end{pmatrix},$$

where

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \qquad P_2 = (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), \qquad P_4 = A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, \qquad P_6 = (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

The key property of **Strassen's algorithm** is that only *seven* matrix multiplications and eighteen matrix additions are needed, instead of the *eight* matrix multiplications and four matrix additions required using conventional block matrix multiplications. Since for large dimensions multiplication of two matrices is more expensive (n^3) than addition (n^2) this will lead to a saving in operations.

Strassen's algorithm can be used recursively. to multiply two square matrices of dimension $n = 2^k$. The number of multiplications is then reduced from n^3 to $n^{\log_2 7} = n^{2.807\cdots}$. (The number of additions is of the same order.) Even with just one level of recursion Strassen's method is faster in practice when n is larger than about 100, see Problem 2. However, there is some loss of numerical stability compared to conventional matrix multiplication, see Higham [41, Ch. 23].

By using the block formulation recursively, and Strassen's method for the matrix multiplication it is possible to perform the LU factorization in $O(n^{\log_2 7})$ operations.

7.7.3 Kronecker Systems

Linear systems where the matrix is a **Kronecker product**[22] arise in several application areas such as signal and image processing, photogrammetry, multidimensional data fitting, etc. Such systems can be solved with great savings in storage

[22] Leopold Kronecker (1823–1891) German mathematician. He is known also for his remark "God created the integers, all else is the work of man".

and operations. Since often the size of the matrices A and B is large, resulting in models involving several hundred thousand equations and unknowns, such savings may be essential.

Definition 7.7.3.
Let $A \in \mathbf{C}^{m\times n}$ and $B \in \mathbf{C}^{p\times q}$ be two matrices. Then the **Kronecker product** *of A and B is the $mp \times nq$ block matrix*

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}. \tag{7.7.12}$$

We now state without proofs some elementary facts about Kronecker products. From the definition (7.7.12) it follows that

$$\begin{aligned} (A+B)\otimes C &= (A\otimes C)+(B\otimes C),\\ A\otimes(B+C) &= (A\otimes B)+(A\otimes C),\\ A\otimes(B\otimes C) &= (A\otimes B)\otimes C,\\ (A\otimes B)^T &= A^T\otimes B^T. \end{aligned}$$

Further we have the important mixed-product relation, which is not so obvious:

Lemma 7.7.4.
Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{p\times q}$, $C \in \mathbf{R}^{n\times k}$, and $D \in \mathbf{R}^{q\times r}$. Then the ordinary matrix products AC and BD are defined, and

$$(A\otimes B)(C\otimes D) = AC\otimes BD. \tag{7.7.13}$$

Proof. Let $A = (a_{ik})$ and $C = (c_{kj})$. Partitioning according to the sizes of B and D, $A\otimes B = (a_{ik}B)$ and $C\otimes D = (c_{kj}D)$. Hence, the (i,j)th block of $(A\otimes B)(C\otimes D)$ equals

$$\sum_{k=1}^{n} a_{ik}Bc_{kj}D = \Big(\sum_{k=1}^{n} a_{ik}c_{kj}\Big)BD,$$

which is the (i,j)th element of AC times BD, which is the (i,j)th block of $(A\otimes B)(C\otimes D)$. ∎

If $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{p\times p}$ are non-singular, then then by Lemma 7.7.4

$$(A^{-1}\otimes B^{-1})(A\otimes B) = I_n \otimes I_p = I_{np}.$$

It follows that $A\otimes B$ is nonsingular and

$$(A\otimes B)^{-1} = A^{-1}\otimes B^{-1}. \tag{7.7.14}$$

We now introduce an operator closely related to the Kronecker product, which converts a matrix into a vector.

Definition 7.7.5. *Given a matrix* $C = (c_1, c_2, \ldots, c_n) \in \mathbf{R}^{m\times n}$ *we define*

$$\operatorname{vec}(C) = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}, \tag{7.7.15}$$

that is, the vector formed by **stacking** *the columns of* C *into one long vector.*

We now state an important result which shows how the vec-function is related to the Kronecker product.

Lemma 7.7.6.
If $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{p\times q}$, *and* $C \in \mathbf{R}^{q\times n}$, *then*

$$(A \otimes B)\operatorname{vec} C = \operatorname{vec} X, \quad X = BCA^T. \tag{7.7.16}$$

Proof. Denote the kth column of a matrix M by M_k. Then

$$\begin{aligned}(BCA^T)_k = BC(A^T)_k &= B\sum_{i=1}^{n} a_{ki}C_i \\ &= (\, a_{k1}B \quad a_{k2}B \quad \ldots a_{kn}B\,)\operatorname{vec} C,\end{aligned}$$

where Let $A = (a_{ij})$. But this means that $\operatorname{vec}(BCA^T) = (A \otimes B)\operatorname{vec} C$. □

Let $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{p\times p}$ be non-singular, and $C \in \mathbf{R}^{p\times n}$. Consider the Kronecker linear system

$$(A \otimes B)x = \operatorname{vec} C, \tag{7.7.17}$$

which is of order np. Then by (7.7.14) the solution can be written

$$x = (A^{-1} \otimes B^{-1})\operatorname{vec} C = \operatorname{vec}(X), \quad X = B^{-1}CA^{-T}. \tag{7.7.18}$$

where C is the matrix such that $c = \operatorname{vec}(C)$. This reduces the operation count for solving (7.7.17) from $O(n^3p^3)$ to $O(n^2p + np^2)$.

7.7.4 Linear Algebra Software

The first collection of high quality software was a series of algorithms written in Algol 60 that appeared in a handbook edited by Wilkinson and Reinsch [68, 1971]. This contains 11 subroutines for linear systems, least squares, and linear programming and 18 routines for the algebraic eigenvalue problem.

The collection LINPACK of Fortran subroutines for linear systems that followed contained several important innovations; see Dongarra et al. [18, 1979]. As much as possible of the computations were performed by calls to so called Basic Linear Algebra Subprograms (BLAS) [48]. These identified frequently occurring

vector operations in linear algebra such as scalar product, adding of a multiple of one vector to another. For example, the operations

$$y := \alpha x + y, \qquad \alpha := \alpha + x^T y$$

in single precision was named SAXPY. By carefully optimizing these BLAS for each specific computer performance could be enhanced without sacrificing portability. LINPACK was followed by EISPACK, a collection of routines for the algebraic eigenvalue problem; see Smith et al. [58, 1976], B. S. Garbow et al. [29, 1977].

The original BLAS, now known as Level 1 BLAS, were found to be unsatisfactory for vector computers, the level 2 BLAS for matrix-matrix operations were introduced in 1988 [20]. These operations involve one matrix A and one or several vectors x and y, e.g., the real matrix-vector products

$$y := \alpha Ax + \beta y, \qquad y := \alpha A^T x + \beta y,$$

and

$$x := Tx, \qquad x := T^{-1}x, \qquad x := T^T x,$$

where α and β are scalars, x and y are vectors, A a matrix and T an upper or lower triangular matrix. The corresponding operations on complex data are also provided.

In most computers in use today the key to high efficiency is to avoid as much as possible data transfers between memory, registers and functional units, since these can be more costly than arithmetic operations on the data. This means that the operations have to be carefully structured. The LAPACK collection of subroutines [2] was initially released in 1992 to address these questions. LAPACK was designed to supersede and integrate the algorithms in both LINPACK and EISPACK. The subroutines are restructured to achieve much greater efficiency on modern high-performance computers. This is achieved by performing as much as possible of the computations by calls to so called Level 2 and 3 BLAS. These enables the LAPACK routines to combine high performance with portable code and is also an aid to clarity, portability and modularity.

Level 2 BLAS involve $O(n^2)$ data, where n is the dimension of the matrix involved, and the same number of arithmetic operations. However, on computers with hierarchical memories, as is now the rule, they failed to obtain adequate performance. Therefore level 3 BLAS were finally introduced in 1990 [19]. These were derived in a fairly obvious manner from some level 2 BLAS, by replacing the vectors x and y by matrices B and C,

$$C := \alpha AB + \beta C, \qquad C := \alpha A^T B + \beta C, \qquad C := \alpha AB^T + \beta C,$$

and

$$B := TB, \qquad B := T^{-1}B, \qquad B := T^T B,$$

Since level 3 BLAS use $O(n^2)$ data but perform $O(n^3)$ arithmetic operations and gives a surface-to-volume effect for the ratio of data movement to operations. This avoids excessive data movements between different parts of memory hierarchy. Level 3 BLAS are used in LAPACK, the linear algebra package that is the successor

of LINPACK, which achieves close to optimal performance on a large variety of computer architectures.

LAPACK is continually improved and updated and is available for free from `http://www.netlib.org/lapack95/`. Several specila forms of matrices are supported by LAPACK: General

General band
Positive definite
Positive definite packed
Positive definite band
Symmetric (Hermitian) indefinite
Symmetric (Hermitian) indefinite packed
Triangular
General tridiagonal
Positive definite tridiagonal

The LAPACK subroutines form the backbone of Cleve Moler's MATLAB system, which has simplified matrix computations tremendously.

LAPACK95 is a Fortran 95 interface to the Fortran 77 LAPACK library. It is relevant for anyone who writes in the Fortran 95 language and needs reliable software for basic numerical linear algebra. It improves upon the original user-interface to the LAPACK package, taking advantage of the considerable simplifications that Fortran 95 allows. LAPACK95 Users' Guide provides an introduction to the design of the LAPACK95 package, a detailed description of its contents, reference manuals for the leading comments of the routines, and example programs.

Review Questions

1. How many operations are needed (approximately) for
   (a) The LU factorization of a square matrix?
   (b) The solution of $Ax = b$, when the triangular factorization of A is known?

2 To compute the matrix product $C = AB \in \mathbf{R}^{m\times p}$ we can either use an outer product or an inner product formulation. Discuss the merits of the two resulting algorithms when A and B have relatively few nonzero elements.

Problems

1. Assume that for the nonsingular matrix $A_{n-1} \in \mathbf{R}^{(n-1)\times(n-1)}$ we know the LU factorization $A_{n-1} = L_{n-1}U_{n-1}$. Determine the LU factorization of the **bordered matrix** $A_n \in \mathbf{R}^{n\times n}$,

$$A_n = \begin{pmatrix} A_{n-1} & b \\ c^T & a_{nn} \end{pmatrix} = \begin{pmatrix} L_{n-1} & 0 \\ l^T & 1 \end{pmatrix} \begin{pmatrix} U_{n-1} & u \\ 0 & u_{nn} \end{pmatrix}.$$

Here $b, c \in \mathbf{R}^{n-1}$ and a_{nn} are given and $l, u \in \mathbf{R}^{n-1}$ and u_{nn} are to be determined.

2. The methods of forwards- and back-substitution extend to block triangular systems. Show that the 2×2 block upper triangular system

$$\begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

can be solved by block back-substitution provided that the diagonal blocks U_{11} and U_{22} are square and nonsingular.

3. Write a recursive LU Factorization algorithm based on the 2×2 block LU algorithm.

4. (a) Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, with m and n even. Show that, whereas conventional matrix multiplication requires mnp multiplications (M) and $m(n-1)p$ additions (A) to form the product $C = AB \in \mathbf{R}^{m\times p}$, Strassen's algorithm, using conventional matrix multiplication at the block level, requires

$$\frac{7}{8}mnp \text{ M } + \frac{7}{8}m(n-2)p + \frac{5}{4}n(m+p) + 2mp \text{ A}.$$

(b) Show, using the result in (a), that if we assume that "M $\approx$ A", Strassen's algorithm is cheaper than conventional multiplication when $mnp \leq 5(mn + np + mp)$.

5. Show the equality

$$\text{vec}\,(A)^T \text{vec}\,(B) = \text{trace}\,(A^T B). \tag{7.7.19}$$

7.8 Sparse Linear Systems

7.8.1 Introduction

A matrix $A \in \mathbf{R}^{n\times n}$ is called **sparse** if only a small fraction of its elements are nonzero. Similarly, a linear systems $Ax = b$ is called sparse if its matrix A is sparse. The simplest class of sparse matrices is the class of banded matrices treated in Sec. 7.4. These have the property that in each row all nonzero elements are contained in a relatively narrow band centered around the main diagonal. Matrices of small bandwidth occur naturally, since they correspond to a situation where only variables "close" to each other are coupled by observations.

Large sparse linear systems of more general structure arise in numerous areas of application such as the numerical solution of partial differential equations, mathematical programming, structural analysis, chemical engineering, electrical circuits and networks, etc. Large could imply a value of n in the range 1,000–1,000,000. Typically, A will have only a few (say, 5–30) nonzero elements in each row, regardless of the value of n. In Fig. 7.8.1 we show a sparse matrix of order 479 with 1887 nonzero elements and its LU factorization. It is a matrix W called west0479 in the Harwell–Boeing sparse matrix test collection, see Duff, Grimes and Lewis [23]. It comes from a model due to Westerberg of an eight stage chemical distillation column. Other applications may give pattern with quite different characteristics.

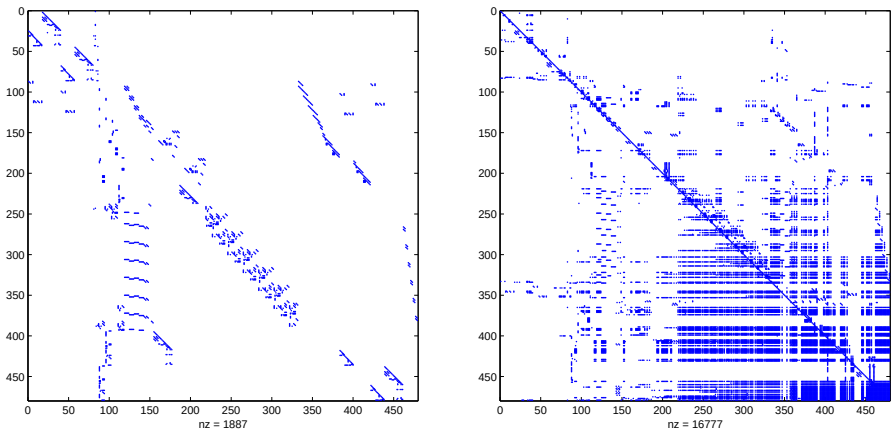

Figure 7.8.1. *Nonzero pattern of a matrix W and its LU factors.*

For many sparse linear systems iterative methods (see Chapter 11) may be preferable to use. This is particularly true of linear systems derived by finite difference methods for partial differential equations in two and three dimensions. In this section we will study elimination methods for sparse systems. These are easier to develop as black box algorithms. Iterative methods, on the other hand, often have to be specially designed for a particular class of problems.

When solving sparse linear systems by direct methods it is important to avoid storing and operating on the elements which are known to be zero. One should also try to minimize **fill-in** as the computation proceeds, which is the term used to denote the creation of new nonzeros during the elimination. For example, as shown in Fig. 7.8.1, the LU factors of W contain 16777 nonzero elements about nine times as many as in the original matrix. The object is to reduce storage and the number of arithmetic operations. Indeed, without exploitation of sparsity, many large problems would be totally intractable.

7.8.2 Storage Schemes for Sparse Matrices

A simple scheme to store a sparse matrix is to store the nonzero elements in an unordered one-dimensional array AC together with two integer vectors ix and jx containing the corresponding row and column indices.

$$ac(k) = a_{i,j}, \quad i = ix(k), \quad j = jx(k), \quad k = 1 : nz.$$

Hence A is stored in "coordinate form" as an unordered set of triples consisting of a numerical value and two indices. This scheme is very convenient for the initial representation of a general sparse matrix. Note that further nonzero elements are easily added to the structure. This coordinate form is very convenient for the original input of a sparse matrix. A drawback is that using this storage structure it is difficult to access the matrix A by rows or by columns, which is needed for the implementation of Gaussian elimination.

Example 7.8.1. The matrix

$$A = \begin{pmatrix} a_{11} & 0 & a_{13} & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{24} & 0 \\ 0 & a_{32} & a_{33} & 0 & a_{35} \\ 0 & a_{42} & 0 & a_{44} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix},$$

is stored in coordinate form as

$$\begin{aligned} AC &= (a_{13}, a_{22}, a_{21}, a_{33}, a_{35}, a_{24}, a_{32}, a_{42}, a_{44}, a_{55}, a_{54}, a_{11}) \\ i &= (1, 2, 2, 3, 3, 2, 3, 4, 4, 5, 5, 1) \\ j &= (3, 2, 1, 3, 5, 4, 2, 2, 4, 5, 4, 1) \end{aligned}$$

In some applications, one encounters matrices of banded structure, where the bandwidth differs from row to row. For this class of matrices, called **variable-band matrices**, we define

$$f_i = f_i(A) = \min\{j \mid a_{ij} \neq 0\}, \quad l_j = l_j(A) = \min\{i \mid a_{ij} \neq 0\}. \tag{7.8.1}$$

Here f_i is the column subscript of the first nonzero in the i-th *row* of A, and similarly l_j the row subscript of the first nonzero in the jth *column* of A. We assume here and in the following that A has a zero free diagonal. From the definition it follows that $f_i(A) = l_i(A^T)$. Hence for a symmetric matrix A we have $f_i(A) = l_i(A)$, $i = 1 : n$.

Definition 7.8.1.
The **envelope** *(or profile) of A is the index set*

$$\text{Env}\,(A) = \{(i, j) \mid f_i \leq j \leq i; \text{ or } l_j \leq i < j;\}. \tag{7.8.2}$$

The envelope of a symmetric matrix is defined by the envelope of its lower (or upper) triangular part including the main diagonal.

For a variable band matrix it is convenient to use a storage scheme, in which every element a_{ij}, $(i, j) \in \text{Env}\,(A)$ is stored. This means that zeros outside the envelope are exploited, but those inside the envelope are stored. This storage scheme is useful because of the important fact that only zeros inside the envelope will suffer fill-in during Gaussian elimination.

The proof of the following theorem is left as an exercise.

Theorem 7.8.2.
Assume that the triangular factors L and U of A exist. Then it holds that

$$\text{Env}\,(L + U) = \text{Env}\,(A),$$

i.e., the nonzero elements in L and U are contained in the envelope of A.

One of the main objectives of a sparse matrix data structure is to economize on storage while at the same time facilitating subsequent operations on the matrix. We now consider storage schemes that permits rapid execution of the elimination steps when solving general sparse linear systems. Usually the pattern of nonzero elements is very irregular, as illustrated in Fig. 7.8.1. We first consider a storage scheme for a sparse vector x. The nonzero elements of x can be stored in **compressed form** in a vector xc with dimension nnz, where nnz is the number of nonzero elements in x. Further, we store in an integer vector ix the indices of the corresponding nonzero elements in xc. Hence the sparse vector x is represented by the triple (nnz, xc, ix), where

$$xc_k = x_{ix(k)}, \quad k = 1 : nnz.$$

Example 7.8.2. The vector $x = (0, 4, 0, 0, 1, 0, 0, 0, 6, 0)$ can be stored in compressed form as

$$xc = (1, 4, 6), \qquad ix = (5, 2, 9), \qquad nnz = 3$$

Operations on sparse vectors are simplified if *one* of the vectors is first **uncompressed**, i.e, stored in a full vector of dimesnion n. Clearly this operation can be done in time proportional to the number of nonzeros, and allows direct random access to specified element in the vector. Vector operations, e.g., adding a multiple a of a sparse vector x to an uncompressed sparse vector y, or computing the inner product $x^T y$ can then be performed in *constant time per nonzero element*. Assume, for example, that the vector x is held in compressed form as nnz pairs of values and indices, and y is held in a full length array. Then the operation $y := a * x + y$ may be expressed as

$$\text{for } k = 1 : nnz, \quad y(ix(k)) := a * xc(k) + y(ix(k));$$

A matrix can be stored as a collection of sparse row vectors, where each row vector is stored in AC in compressed form. The corresponding column subscripts are stored in the integer vector jx, i.e., the column subscript of the element ac_k is given in $jx(k)$. Finally we need a third vector $ia(i)$, which gives the position in the array AC of the first element in the ith row of A. For example, the matrix in Example 7.8.1 is stored as

$$\begin{aligned} AC &= (a_{11}, a_{13} \mid a_{21}, a_{22}, a_{24} \mid a_{32}, a_{33}, a_{35} \mid a_{42}, a_{44} \mid a_{54}, a_{55}), \\ ia &= (1, 3, 6, 9, 11, 13), \\ jx &= (1, 3, 1, 2, 4, 2, 3, 5, 2, 4, 4, 5). \end{aligned}$$

Alternatively a similar scheme storing A as a collection of column vectors may be used. A drawback with these schemes is that it is expensive to insert new nonzero elements in the structure when fill-in occurs.

The components in each row need not be ordered; indeed there is often little advantage in ordering them. To access a nonzero a_{ij} there is no direct method

of calculating the corresponding index in the vector AC. Some testing on the subscripts in jx has to be done. However, more usual is that a complete row of A has to be retrieved, and this can be done quite efficiently. This scheme can be used unchanged for storing the lower triangular part of a symmetric positive definite matrix.

If the matrix is stored as a collection of sparse row vectors, the entries in a particular column cannot be retrieved without a search of nearly all elements. This is needed, for instance, to find the rows which are involved in a stage of Gaussian elimination. A solution is then to store also the structure of the matrix as a set of column vectors. If a matrix is input in coordinate form it the conversion to this storage form requires a sorting of the elements, since they may be in arbitrary order. Such a sortin can be done very efficiently in $O(n) + O(\tau)$ time.

Another way to avoid extensive searches in data structures is to use a linked list to store the nonzero elements. Associated with each element is a pointer to the location of the next element in its row and a pointer to the location of the next element in its column. If also pointer to the first nonzero in each row and column are stored there is a total overhead of integer storage of $2(\tau + n)$, where τ is the number of nonzero elements in the factors and n is the order of the matrix. This allows fill-ins to be added to the data structure with only two pointers being altered. Also the fill-in can be placed anywhere in storage so no reorderings are necessary. Disadvantages are that indirect addressing must be used when scanning a row or columnand that the elements in one row or column can be scattered over a wide range of memory.

An important distinction is between **static** storage structures that remain fixed and **dynamic** structures that can accommodate fill-in. If only nonzeros are to be stored, the data structure for the factors must dynamically allocate space for the fill-in during the elimination. A static structure can be used when the location of the nonzeros in the factors can be predicted in advance, as is the case for the Cholesky factorization.

7.8.3 Graph representation of sparse matrices.

In the method of normal equations for solving sparse linear least squares problems an important step is to determine a column permutation P such that the matrix $P^T A^T A P$ has a sparse Cholesky factor R, and to then generate a storage structure for R. This should be done symbolically using only the nonzero structure of A (or $A^T A$) as input. To perform such tasks the representation of the structure of a sparse matrix as a directed or undirected graph is a powerful tool.

A useful way to represent the structure of a symmetric matrix is by an **undirected graph** $G = (X, E)$, consisting of a set of **nodes** X and a set of **edges** E (unordered pairs of nodes). A graph is **ordered** (labeled) if its nodes are labeled. The ordered graph $G(A) = (X, E)$, representing the structure of a symmetric matrix $A \in \mathbf{R}^{n\times n}$, consists of nodes labeled $1, \ldots, n$ and edges $(x_i, x_j) \in E$ if and only if $a_{ij} = a_{ji} \neq 0$. Thus there is a direct correspondence between nonzero elements and edges in its graph; see Figure 6.4.1.

$$A = \begin{pmatrix} \times & \times & & \times & \times & & \\ \times & \times & \times & & & & \\ & \times & \times & & & \times & \times \\ \times & & & \times & & & \\ \times & & & & \times & & \\ & & \times & & & \times & \\ & & \times & & & & \times \end{pmatrix}$$

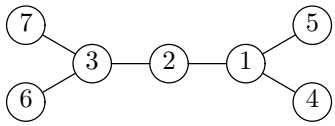

Figure 7.8.2. *The matrix A and its labeled graph.*

Two nodes, x and y, are said to be **adjacent** if there is an edge $(x, y) \in E$. The adjacency set of x in G is defined by

$$\mathrm{Adj}_G(x) = \{y \in X \mid x \text{ and } y \text{ are adjacent}\}.$$

The number of nodes adjacent to x is denoted by $|\mathrm{Adj}_G(x)|$, and is called the **degree** of x. A **path** of length $l \geq 1$ between two nodes, u_1 and u_{l+1}, is an ordered set of distinct nodes $u_1, \ldots, u_{l+1}$, such that

$$(u_i, u_{i+1}) \in E, \quad i = 1, \ldots, l.$$

If there is such a chain of edges between two nodes, then they are said to be **connected**. If there is a path between every pair of distinct nodes, then the graph is connected. A disconnected graph consists of at least two separate connected subgraphs. ($\bar{G} = (\bar{X}, \bar{E})$ is a subgraph of $G = (X, E)$ if $\bar{X} \subset X$ and $\bar{E} \subset E$.) If $G = (X, E)$ is a connected graph, then $Y \subset X$ is called a separator if G becomes disconnected after the removal and the nodes Y.

A symmetric matrix A is said to be **reducible** if there is a permutation matrix P such that $P^T AP$ is block diagonal. Such a symmetric permutation $P^T AP$ of A corresponds to a reordering of the nodes in $G(A)$ without changing the graph. It follows that the graph $G(P^T AP)$ is connected if and only if $G(A)$ is connected. It is then easy to prove that A is reducible if and only if its graph $G(A)$ is disconnected.

The structure of an unsymmetric matrix can similarly be represented by a **directed graph** $G = (X, E)$, where the edges now are ordered pairs of nodes. A directed graph is **strongly connected** if there is a path between every pair of distinct nodes along directed edges.

The structure of a symmetric matrix A can be represented by the **undirected graph of** A.

Definition 7.8.3.

The ordered undirected graph $G(A) = (X, E)$ of a symmetric matrix $A \in \mathbf{R}^{n\times n}$ consists of a set of n nodes *X together with a set E of* edges*, which are unordered pairs of nodes. The nodes are labeled $1, 2 : n$ where n, and nodes i and j are joined by an edge if and only if $a_{ij} = a_{ji} \neq 0$, $i \neq j$. We then say that the nodes i and j are* adjacent*. The number of edges incident to a node is called the* degree *of the node.*

The important observation is that for any permutation matrix $P \in \mathbf{R}^{n\times n}$ *the graphs $G(A)$ and $G(PAP^T)$ are the same except that the labelling of the nodes are different.* Hence the unlabeled graph represents the structure of A without any particular ordering. Finding a good permutation for A is equivalent to finding a good labeling for its graph.

7.8.4 Nonzero Diagonal and Block Triangular Form

Before performing a factorization of a sparse matrix it is often advantageous to perform some pre-processing. An arbitrary square nonsingular matrix $A \in \mathbf{R}^{n\times n}$ there always is a row permutation P such that PA has nonzero elements on it diagonal. Further, there is a row permutation P and column permutation Q such that PAQ has a nonzero diagonal and **block triangular structure**

$$PAQ = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1,t} \\ & A_{22} & \dots & A_{2,t} \\ & & \ddots & \vdots \\ & & & A_{tt} \end{pmatrix} \tag{7.8.3}$$

with square nonsingular diagonal blocks $A_{11}, \dots, A_{tt}$. The off-diagonal blocks are possibly nonzero matrices of appropriate dimensions. Using this structure a linear system $Ax = b$ or $PAQy = c$, where $y = Q^T x$, $c = Pb$, reduces to

$$A_{ii}y_i = c_i - \sum_{j=i+1}^{n} A_{ij}x_j, \quad j = n : -1 : 1. \tag{7.8.4}$$

Hence we only need to factorize the diagonal blocks. This block back-subsitution can lead to significant savings.

If we require that the diagonal blocks are irreducible, then the block triangular form (7.8.3) can be shown to be essentially unique. Any one block triangular form can be obtained from any other by applying row permutations that involve the rows of a single block row, column permutations that involve the columns of a single block column, and symmetric permutations that reorder the blocks. A square matrix which can be permuted to the form (7.8.3), with $t > 1$, is said to be **reducible**; otherwise it is called **irreducible**.

In the symmetric positive definite case a similar reduction to block upper triangular form can be considered, where $Q = P^T$. Some authors reserve the terms reducible for the case, and use the terms bi-reducible and bi-irreducible for the general case.

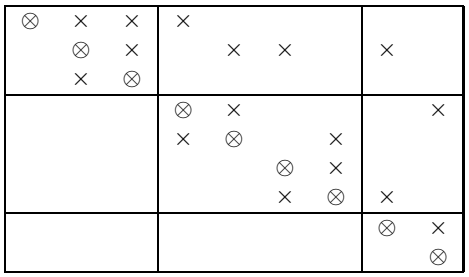

Figure 7.8.3. *The block triangular decomposition of A.*

An arbitrary rectangular matrix $A \in \mathbf{R}^{m\times n}$ has a block triangular form called the **Dulmage–Mendelsohn form**. If A is square and nonsingular this is the form (7.8.3). The general case is based on a canonical decomposition of bipartite graphs discovered by Dulmage and Mendelsohn. In the general case the first diagonal block may have more columns than rows, the last diagonal block more rows than column. All the other diagonal blocks are square and nonzero diagonal entries.This block form can be used for solving least squares problems by a method analogous to back-substitution.

The **bipartite graph** associated with A is denoted by $G(A) = \{R, C, E\}$, where $R = (r_1, \ldots, r_m)$ is a set of vertices corresponding to the rows of A and $C = (c_1, \ldots, c_m)$ a set of vertices corresponding to the columns of A. E is the set of edges, and $\{r_i, c_j\} \in E$ if and only if $a_{ij} \neq 0$. A **matching** in $G(A)$ is a subset of its edges with no common end points. In the matrix A this corresponds to a subset of nonzeros, no two of which belong to the same row or column. A **maximum matching** is a matching with a maximum number $r(A)$ of edges. The **structural rank** of A equals $r(A)$. Note that the mathematical rank is always less than or equal to its structural rank. For example, the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

has structural rank 2 but numerical rank 1.

For the case when A is structurally nonsingular matrix there is a two-stage algorithm for permuting A to block upper triangular form. In the first stage a maximum matching in the bipartite graph $G(A)$ with row set R and column set C is found. In the second stage the block upper triangular form of each submatrix determined from the strongly connected components in the graph $G(A)$, with edges directed from columns to rows.

If A has structural rank n but is *numerically* rank deficient it will not be possible to factorize all the diagonal blocks in (7.8.3). In this case the block triangular structure given by the Dulmage–Mendelsohn form cannot be preserved, or some blocks may become severely ill-conditioned.

Note that for some applications, e.g., for matrices arising from discretizations of partial differential equations, it may be known a priori that the matrix is irre-

ducible. In other applications the block triangular decomposition may be known in advance from the underlying physical structure. In both these cases the algorithm discussed above is not useful.

7.8.5 LU Factorization of Sparse Matrices

Hence the first task in solving a sparse system is to order the rows and columns so that Gaussian elimination applied to the permuted matrix PAQ does not introduce too much fill-in. To find the *optimal* ordering, which minimizes the number of nonzero in L and U is unfortunately a hard problem. This is because the number of possible orderings of rows and columns is very large, $(n!)^2$, wheras solving a linear system only takes $O(n^3)$ operations. Fortunately, there are heuristic ordering algorithms which do a god job at approximately minimizing fill-in. These orderinga usually also nearly minimize the arithmetic operation count.

Example 7.8.3.

The ordering of rows and columns in Gaussian elimination may greatly affect storage and number of arithmewtic operations as shown by the following example. Let

$$A = \begin{pmatrix} \times & \times & \times & \dots & \times \\ \times & \times & & & \\ \times & & \times & & \\ \vdots & & & \ddots & \\ \times & & & & \times \end{pmatrix}, \qquad PAP^T = \begin{pmatrix} \times & & & & \times \\ & \ddots & & & \vdots \\ & & \times & & \times \\ & & & \times & \times \\ \times & \dots & \times & \times & \times \end{pmatrix}.$$

Matrices, or block matrices of this structure are called **arrowhead matrices** and occur in many applications.

If the $(1,1)$ element in A is chosen as the first pivot the fill in will be total and $n^3/3$ operations required for the LU factorization. In PAP^T the orderings of rows and columns have been reversed. Now there is no fill-in except in the last step of, when pivots are chosen in natural order.Only about $2n$ flops are required to perform the factorization.

For variable-band matrices no fill-in occurs in L and U outside the envelope. One strategy therefore is to choose P and Q to approximately minimize the envelope of PAQ. (Note that the reordered matrix PAP^T in Example 7.8.3 has a small envelope but A has a full envelope!) For symmetric matrices the reverse Cuthill–McKee ordering is often used. In the unsymmetirc case one can determine a reordering of the columns by applying this algorithm to the symmetric structure of $A + A^T$.

Perhaps surprisingly, the orderings that approximately minimize the total fill-in in LU factorization tend *not* to give a small bandwidth. Typically, the factors L and U instead have their nonzeros scattered throughout their triangular parts. A simple column reordering is to sort the columns by increasing column count, i.e. by the number of nonzeros in each column. This can often give a substantial reduction of the fill-in in Gaussian elimination. In Figure 7.8.5 we show the LU factorization

of the matrix W reordered after column count and its LU factors. The number of nonzeros in L and U now are 6604, which is a substantial reduction.

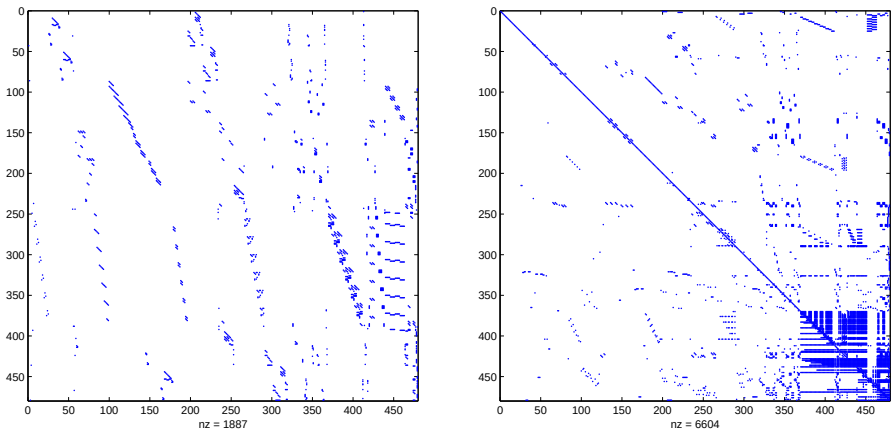

Figure 7.8.4. *Nonzero pattern of a matrix and its LU factors after reordering by increasing column count.*

An ordering that often performs even better is the so called column minimum degree ordering shown in Figure 7.8.5. The LU factors of the reordered matrix now containg 5904 nonzeros. This column ordering is obtained by using the symmetric minimum degree described in the next section on the matrix W^TW. MATLAB uses an implementation of this ordering algorithm that does not actually form the matrix W^TW. For the origin and details of this code we refer to Gilbert, Moler, and Schreiber [33].

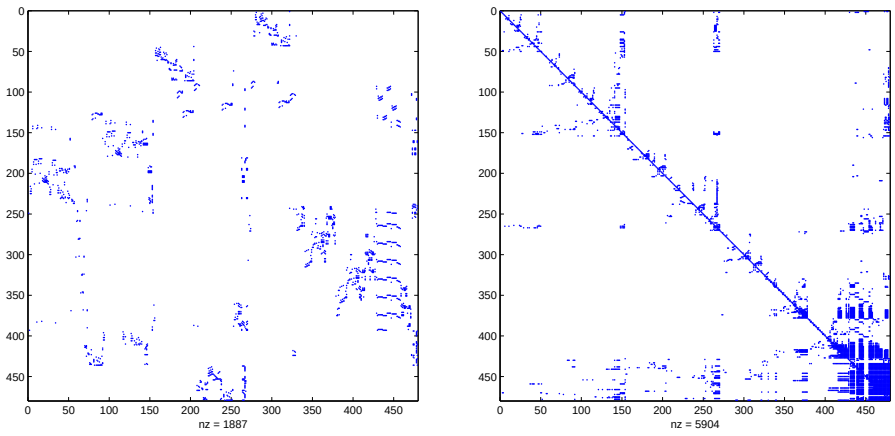

Figure 7.8.5. *Nonzero pattern of a matrix and its LU factors after minimum degree ordering.*

For unsymmetric systems some kind of stability check on the pivot elements must be performed during the numerical factorization. Therefore the storage structure for L and U cannot be predicted from the structure of A only. but must be determined dynamically during the numerical elimination phase.

MATLAB uses the column sweep method with partial pivoting due to Gilbert and Peierls [34] for computing the LU factorization a column of L and U at a time. In this the basic operation is to solve a series of sparse triangular system involving the already computed part of L. The column-oriented storage structure is set up dynamically as the factorization progresses. Note that the size of storage needed can not be predicted in advance. The total time for this LU factorization algorithm can be shown to be proportional to the number of arithmetic operations plus the size of the result.

Other sparse LU algorithms reorders both rows and columns before the numerical factorization. One of the most used ordering algorithm is the **Markowitz algorithm**. To motivate this suppose that Gaussian elimination has proceeded through k stages and let $A^{(k)}$ be the remaining active submatrix. Denote by r_i is the number of nonzero elements in the ith row and c_j is the number of nonzero elements in the jth column of $A^{(k)}$. In the Markowitz algorithm one performs a row and column interchange so that the product

$$(r_i - 1)(c_j - 1),$$

is minimized. (Some rules for tie-breaking are also needed.) This is equivalent to a *local minimization* of the fill-in at the next stage, assuming that all entries modified were zero beforehand. This choice also minimizes the number of multplications required for this stage.

With such an unsymmetric reordering there is a conflict with ordering for sparsity and for stability. The ordering for sparsity may not give pivotal elements which are acceptable from the point of numerical stability. Usually a **threshold pivoting** scheme is used to minimize the reorderings. This means that the chosen pivot is restricted by an inequality

$$|a_{ij}^{(k)}| \geq \tau \max_r |a_{rj}^{(k)}|, \tag{7.8.5}$$

where τ, $0 < \tau \leq 1$, is a predetermined threshold value. A value of $\tau = 0.1$ is usually recommended as a good compromise between sparsity and stability. (Note that the usual partial pivoting strategy is obtained for $\tau = 1$.) The condition (7.8.5) ensures that in any column that is modified in an elimination step the maximum element increases in size by at most a factor of $(1 + 1/\tau)$. Note that a column is only modified if the pivotal row has a nonzero element in that column. The total number of times a particular column is modified during the complete elimination is often quite small if the matrix is sparse. Furthermore, it is possible to monitor stability by, for example, computing the relative backward error, see Sec. 7.5.2.

7.8.6 Cholesky Factorization of Sparse Matrices

If A is symmetric and positive definite, then the Cholesky factorization is numerically stable for any choice of pivots along the diagonal. We need only consider

symmetric permutations PAP^T, where P can be chosen with regard only to sparsity. This, leads to a substantial increase in the efficiency of the sparse Cholesky algorithm since a static storage structure can be used.

We remark that the structure predicted for R from that of P^TAP by performing the Cholesky factor symbolically, is such that $R + R^T$ will be at least as full as PAP^T. In Figure 7.8.6 we show the nonzero pattern of the matrix $S = WW^T$, where W is the matrix west0479, and its Cholesky factor.

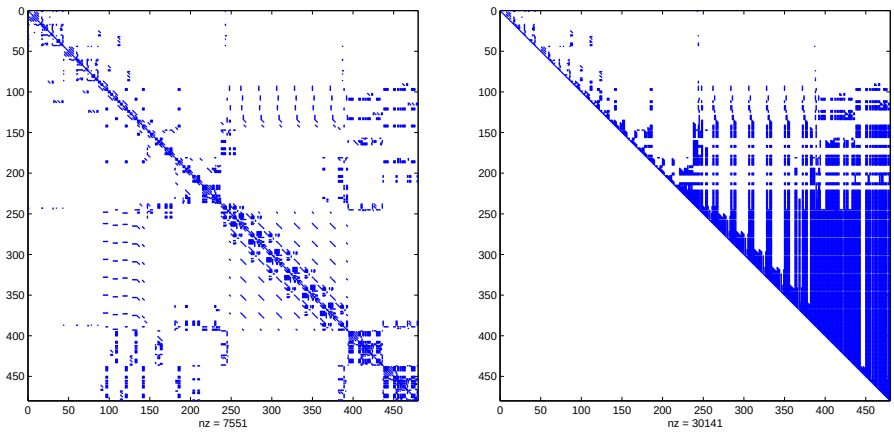

Figure 7.8.6. *Nonzero pattern of a matrix and its Cholesky factor.*

The Cholesky factorization of a sparse symmetric positive definite matrix A can be divided into four separate steps:

1. Determine a permutation P such that P^TAP has a sparse Cholesky factor L.

2. Perform a symbolic Cholesky factorization of PAP^T and generate a storage structure for R.

3. Form P^TAP and store in data structure for R.

4. Compute numerically the Cholesky factor R such that $P^TAP = R^TR$.

We stress that steps 1 and 2 are done symbolically, only working on the structure of A. The numerical computations take place in steps 3 and 4 a static storage scheme can be used.

Example 7.8.4.
To illustrate the symbolic factorization we use the sparse symmetric matrix A

with Cholesky factor R

$$A = \begin{pmatrix} \times & \times & & \times & \times & & \\ \times & \times & \times & & & & \\ & \times & \times & & & \times & \times \\ \times & & & \times & & & \\ \times & & & & \times & & \\ & & \times & & & \times & \\ & & \times & & & & \times \end{pmatrix}, \qquad R = \begin{pmatrix} \times & \times & & \times & \times & & \\ & \times & \times & + & + & & \\ & & \times & + & + & \times & \times \\ & & & \times & + & & \\ & & & & \times & & \\ & & & & & \times & \\ & & & & & & \times \end{pmatrix},$$

where $\times$ and $+$ denote a nonzero element. We show only the nonzero structure of A and R, not any numerical values. The five elements marked $+$ are the fill-in that occur in the Cholesky factorization.

Graph theory provides a powerful tool for the analysis and implementation of ordering algorithms. In the following we restrict ourselves to the case of a symmetric structure. Below is the ordered graph $G(A)$, of the matrix in Example 7.8.4.

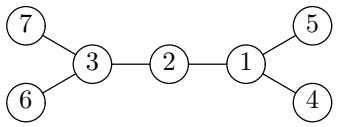

Figure 7.8.7. *The labeled graph of the matrix A.*

Example 7.8.5.

The labelled graph suggest that row and columns of the matrix in Example 7.8.5 is rearranged in order $4, 5, 7, 6, 3, 1, 2$. With this ordering the Cholesky factor of the matrix PAP^T will have no fill-in!

$$PAP^T = \begin{pmatrix} \times & & & & & \times & \\ & \times & & & & \times & \\ & & \times & & \times & & \\ & & & \times & \times & & \\ & & \times & \times & \times & & \times \\ \times & \times & & & & \times & \times \\ & & & & \times & \times & \times \end{pmatrix}, \qquad R = \begin{pmatrix} \times & & & & & \times & \\ & \times & & & & \times & \\ & & \times & & \times & & \\ & & & \times & \times & & \\ & & & & \times & & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{pmatrix},$$

From the graph $G(A^TA)$ the structure of the Cholesky factor R can be predicted by using a graph model of Gaussian elimination. The fill-in under the factorization process can be analyzed by considering a sequence of **elimination graphs** that can be recursively formed as follows. We take $G_0 = G(A)$, and form G_i from $G_{(i-1)}$ by removing the node i and its incident edges and adding fill edges. The fill edges in eliminating node v in the graph G are

$$\{(j,k) \mid (j,k) \in \mathrm{Adj}_G(v), j \neq k\}.$$

Thus the fill edges correspond to the set of edges required to make the adjacent nodes of v pairwise adjacent. The filled graph $G_F(A)$ of A is a graph with n vertices and edges corresponding to all the elimination graphs G_i, $i = 0, \ldots, n-1$. The filled graph bounds the structure of the Cholesky factor R,

$$G(R^T + R) \subset G_F(A). \tag{7.8.6}$$

Under a no-cancellation assumption, the relation (7.8.6) holds with equality.

The following characterization of the filled graph describes how it can be computed directly from $G(A)$.

Theorem 7.8.4. *Let $G(A) = (X, E)$ be the undirected graph of A. Then (x_i, x_j) is an edge of the filled graph $G_F(A)$ if and only if $(x_i, x_j) \in E$, or there is a path in $G(A)$ from node i to node j passing only through nodes with numbers less than* $\min(i, j)$.

Consider the structure of the Cholesky factor $R = (r_{ij}$. For each row $i \leq n$ we define $\gamma(i)$ by

$$\gamma(i) = \min\{j > i \mid r_{ij} \neq 0\}, \tag{7.8.7}$$

that is $\gamma(i)$ is the column subscript of the first off-diagonal nonzero element in row i of R. If row i has no off-diagonal nonzero, then $\gamma(i) = i$. Clearly $\gamma(n) = n$. The quantities $\gamma(i)$, $i = 1 : n$ can be used to represent the structure of the sparse Cholesky factor R. For the matrix R in Example 7.8.4 we have

i	1	2	3	4	5	6	7
$\gamma(i)$	2	3	6	4	5	6	7

We now introduce the **elimination tree** corresponding to the structure of the Cholesky factor. The tree has n nodes, labelled form 1 to n. For each i if $\gamma(i) > j$, then node $\gamma(i)$ is the **parent** of node i in the elimination tree and node j is one of possible several **child** nodes of node $\gamma(i)$. If the matrix is irreducible then n is the only node with $\gamma(n) = n$ and is the **root** of the tree. There is exactly one path from node i to the root. If node j lies on the pathe from node i to the root, then node j is an ancestor to node i and node j is a descendant of node i.

The most widely used algorithm for envelope reduction for symmetric matrices is the **reverse Cuthill–McKee ordering**. This works on the graph $G(A)$ as follows:

1. Determine a starting node and label this 1.
2. For $i = 1 : n - 1$ find all unnumbered nodes adjacent to the node with label i, and number them in increasing order of degree.
3. The reverse ordering is obtained by reversing the ordering just determined.

The reversal of the Cuthill–McKee ordering in step 3 was suggested by Alan George, who noticed that it often was much superior to the original ordering produced by steps 1 and 2 above. In order for the algorithm to perform well it is

necessary to choose a good starting node; see George and Liu [31, Section 4.3.3]. In Fig. 7.8.2 we show the structure of the matrix from Fig. 7.8.1 and its Cholesky factor after reverse Cuthill–McKee reordering. The number of non-zero elements in the Cholesky factor is 23,866.

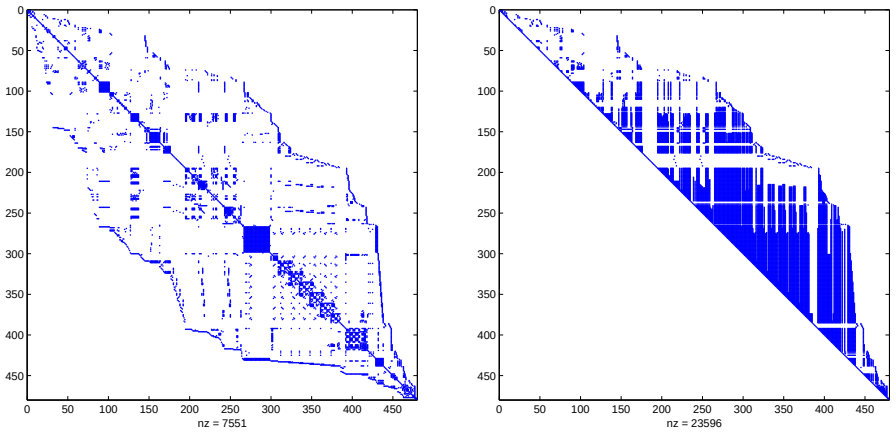

Figure 7.8.8. *Matrix and its Cholesky factor after reverse Cuthill–McKee reordering.*

As for unsymmetric matrices, the orderings that approximately minimize the total fill-in in the Cholesky factor tend to have their nonzeros scattered throughout the matrix. For some problems, such orderings can reduce fill-in by one or more orders of magnitude over the corresponding minimum bandwidth ordering.

In the symmetric case $r_i = c_i$ for the Markowitz ordering. It is then equivalent to minimizing r_i, and the resulting algorithm is called the **minimum-degree algorithm**. The minimum degree ordering can be determined using a graph model of the Cholesky factorization. At the same time the nonzero structure of the Cholesky factor R can be determined and a storage structure for R generated. The minimum-degree algorithm ordering algorithm has been subject to an extensive development. Very efficient implementations now exist. For details we refer to George and Liu [31, *Chapter* 5] and [32].

Figure 7.8.3 shows the structure of the matrix from Fig. 7.8.1 and its Cholesky factor after minimum-degree reordering. The number of non-zero elements in the Cholesky factor is reduced to 12,064. For nested dissection orderings, see George and Liu [31, Chapter 8].

Review Questions

1. Describe the coordinate form of storing a sparse matrix. Why is this not suitable for performing the numerical LU factorization?

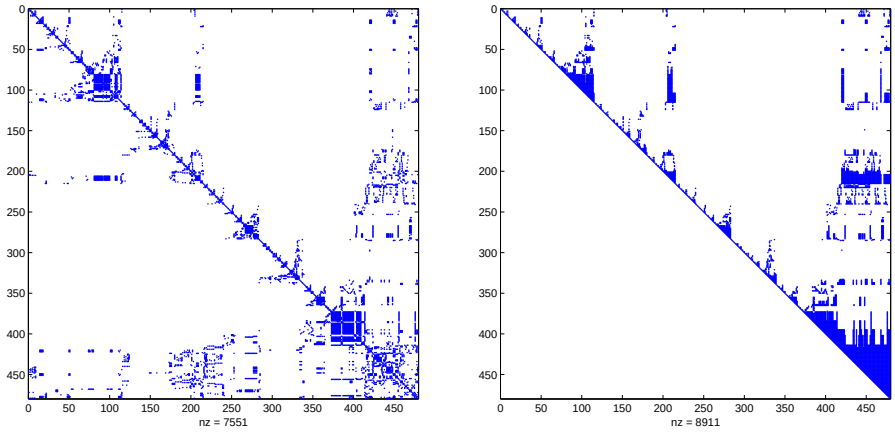

Figure 7.8.9. *Matrix and its Cholesky factor after minimum-degree reordering.*

2. Give an example of a sparse matrix A, which suffers extreme fill-in in Gaussian elimination..

3. Describe the Markowitz algorithm for ordering rows and columns of a non-symmetric matrix before factorization.

4. Describe threshold pivoting. Why is this used instead of partial pivoting in some schemes for LU factorization?

5. What does the reverse Cuthill–McKee ordering minimize?

Problems

1. Let $A, B \in \mathbf{R}^{n\times n}$ be sparse matrices. Show that the number of multiplications to compute the product $C = AB$ is $\sum_{i=1}^{n} \eta_i\theta_i$, where η_i denotes the number of nonzero elements in the ith *column* of A and θ_i the number of nonzeros in the ith *row* of B.

Hint: Use the outer product formulation $C = \sum_{i=1}^{n} a_{\cdot i}b_i^T$.

2. (a) It is often required to add a multiple a of a sparse vector x to another sparse vector y. Show that if the vector x is held in coordinate form as nx pairs of values and indices, and y is held in a full length array this operation may be expressed thus:

$$\begin{aligned}&\text{for } k = 1 : nx\\ &\quad y(\text{index}(k)) = a * x(k) + y(\text{index}(k));\end{aligned}$$

(b) Give an efficient algorithm for computing the inner product of two compressed vectors.

3. Consider a matrix with the symmetric structure

$$A = \begin{pmatrix} \times & & \times & & & \\ & \times & & \times & \times & \\ \times & & \times & \times & & \\ & \times & \times & & & \times \\ & \times & & & \times & \times \\ & & & \times & \times & \times \end{pmatrix}.$$

(a) What is the envelope of A? Where will fill-in occur during Gaussian elimination?

(b) Draw the undirected graph G, which represents the sparsity structure of A.

7.9 Structured Systems

The coefficient matrices in systems of linear equations arising from signal processing, control theory and linear prediction often have some special structure that can be taken advantage of. Several classes of such structured systems can be solved by fast methods in $O(n^2)$ operations, or by super-fast methods even in $O(n \log n)$ operations rather than $O(n^3)$ otherwise required by Gaussian elimination. This has important implications for many problems in signal restoration, acoustics, seismic exploration and many other application areas. Since the numerical stability properties of super-fast methods are generally either bad or unknown we consider only fast methods in the following.

7.9.1 Toeplitz and Hankel Matrices

Note: The following subsection are not yet complete and will be amended.

A **Toeplitz matrix** T is a matrix whose entries are constant along every diagonal; $T = (t_{i-j})_{1 \le i,j \le n}$,

$$T = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \\ t_{-1} & t_0 & \dots & t_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{-n+1} & t_{-n+2} & \dots & t_0 \end{pmatrix} \in \mathbf{R}^{n \times n},$$

and is defined by the $2n-1$ values of $t_{-n+1}, \dots, t_0, \dots, t_{n-1}$. Toeplitz matrices arising in applications are often large, and dimensions of $10,000$ not uncommon. Consequently there is a need for special fast methods for solving Toeplitz systems. In large problems also storage requirements are important. The original matrix T only requires $2n-1$ storage. However, if standard factorization methods are used, at least $n(n+1)/2$ storage is needed.

A **Hankel matrix** is a matrix whose elements are constant along every an-

tidiagonal, i.e.,$H = (h_{i+j-2})_{1\le i,j\le n}$

$$H = \begin{pmatrix} h_0 & h_1 & \dots & h_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-2} & h_{n-1} & \dots & h_{2n-3} \\ h_{n-1} & h_n & \dots & h_{2n-2} \end{pmatrix} \in \mathbf{R}^{n\times n}.$$

Reversing the rows (or columns) of a Hankel matrix we get a Toeplitz matrix. Hence methods developed for solving Toeplitz systems apply also to Hankel systems.

7.9.2 Cauchy-Like Matrices

A **Cauchy matrix** is a matrix of the following form:

$$C = \left(\frac{1}{y_i - z_j}\right)_{1\le i,j\le n}, \qquad a_i, b_j \in \mathbf{R}^p. \tag{7.9.1}$$

where we assume that $y_i \neq z_j$ for $1 \le i, j \le n$.

Example 7.9.1. Consider the problem of finding the coefficients of a rational function

$$r(x) = \sum_{j=1}^{n} a_j \frac{1}{x - y_j},$$

which satisfies the interpolation conditions $r(x_i) = f_i$, $i = 1, \dots, n$. With $a = (a_1, \dots, a_n)$, $f = (f_1, \dots, f_n)$ this leads to the linear system $Ca = f$, where C is the Cauchy matrix in (7.9.1).

Cauchy gave in 1841 the following explicit expression for the determinant

$$\det(C) = \frac{\prod_{1\le i<j\le n} (y_j - y_i)(z_j - z_i)}{\prod_{1\le i\le j\le n} (y_j + z_i)}.$$

We note that any row or column permutation of a Cauchy matrix is again a Cauchy matrix. This property allows fast and stable version of Gaussian to be developed for Cauchy systems.

Many of these methods also apply in the more general case of **Loewner matrices** of the form

$$C = \left(\frac{a_i^T b_j}{y_i - z_j}\right)_{1\le i,j\le n}, \qquad a_i, b_j \in \mathbf{R}^p. \tag{7.9.2}$$

Example 7.9.2. The most famous example of a Cauchy matrix is the **Hilbert matrix**, which is obtained by taking $y_i = z_i = i - 1/2$:

$$H_n \in \mathbf{R}^{n\times n}, \quad h_{ij} = \frac{1}{i+j-1}.$$

For example,

$$H_4 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}.$$

The Hilbert matrix is symmetric and positive definite Hankel matrix. It is also **totally positive** . The inverse of H_n is known explicitly and has integer elements. Hilbert matrices of high order are known to be very ill-conditioned; for large n it holds that $\kappa_2(H_n) \sim e^{3.5n}$.

7.9.3 Vandermonde systems

In Chapter 4 the problem of interpolating given function values $f(\alpha_i)$, $i = 1, \ldots, n$ at distinct points α_i with a polynomial of degree $\leq n-1$ was shown to lead to a linear system of equations with matrix $M = [p_j(\alpha_i)]_{i,j=1}^m$. In the case of the power basis $p_j(z) = z^{j-1}$, the matrix M equals V^T, where V is the **Vandermonde matrix**

$$V = [\alpha_j^{i-1}]_{i,j=1}^n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \cdots & \alpha_n^{n-1} \end{pmatrix}. \tag{7.9.3}$$

Hence the unique polynomial $P(z)$ satisfying the interpolating conditions $P(\alpha_i) = f_i$, $i = 1, \ldots, n$ is given by

$$P(z) = (1, z, \ldots, z^{n-1})a,$$

where a is the solution of the dual Vandermonde system.

$$V^T a = f \tag{7.9.4}$$

One of the most efficient ways to determine $P(x)$ is by Newton's interpolation formula, which uses the basis polynomials

$$Q_1(z) = 1, \qquad Q_k(z) = (z-\alpha_1)\cdots(z-\alpha_{k-1}), \quad k = 2:n-1.$$

We write the polynomial in the form

$$P(z) = (Q_1(z), Q_2(z), \ldots, Q_n(z))c,$$

where c are the divided differences of $f_1 : f_n$. These divided differences can be recursively computed, see Section 4.?. This leads to the algorithm below for computing the coefficient vector a in the power basis. Note that the algorithm operates directly on the α_j's and the matrix V^T is never formed,

Algorithm 7.9.1 Dual Vandermonde System

Given distinct scalars $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $f = (f_1, f_2, \ldots, f_n)^T$ the following algorithm solves the dual Vandermonde system $V^T a = f$:

$a = \text{dvand}(\alpha, f)$
  $a := f;$
  for $k = 1 : n-1$
    for $j = n : (-1) : k+1$
      $a_j := (a_j - a_{j-1})/(\alpha_j - \alpha_{j-k})$
    end
  end
  for $k = n-1 : (-1) : 1$
    for $j = k : n-1$
      $a_j := a_j - \alpha_k * a_{j+1}$
    end
  end

The accuracy of this algorithm depends on the ordering of the interpolation points α_i. Often the best ordering is the monotone ordering for which

$$\alpha_1 < \alpha_2 \cdots < \alpha_n.$$

If moreover $0 \leq \alpha_1$ this algorithm often gives remarkably accurate solutions.

To interpret the Newton interpolation algorithm in matrix terms we define lower bidiagonal matrices

$$L_k(\alpha) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & B_{n-k+1}(\alpha) \end{pmatrix}, \quad k = 1, \ldots, n-1,$$

where

$$B_p(\alpha) = \begin{pmatrix} 1 & & & \\ -\alpha & 1 & & \\ & \ddots & \ddots & \\ & & -\alpha & 1 \end{pmatrix} \in \mathbf{R}^{p \times p}.$$

We further let

$$D_k = \text{diag}\,(1, \ldots, 1, (\alpha_{k+1} - \alpha_1), \ldots, (\alpha_n - \alpha_{n-k})).$$

Then we find that the dual Vandermonde algorithm can be written as

$$\begin{aligned} c &= U^T f, \qquad U^T = D_{n-1}^{-1} L_{n-1}(1) \cdots D_1^{-1} L_1(1), \\ a &= L^T c, \qquad L^T = L_1^T(\alpha_1) L_2^T(\alpha_2) \cdots L_{n-1}^T(\alpha_{n-1}). \end{aligned}$$

Systems of equations with Vandermonde matrix

$$Vx = b \tag{7.9.5}$$

are called primal Vandermonde systems and occur, e.g., in approximation of linear functionals (see Chapter 4). The matrix representation of the algorithm for the dual Vandermonde system allows us to derive an algorithm also for solving primal Vandermonde systems.

Since $a = V^{-T}f = L^T U^T f$, we have $V^{-T} = L^T U^T$. Transposing this relation and find

$$V^{-1} = UL,$$

Hence the solution to the primal system $Vx = b$ is given by $x = V^{-1}b = U(Lb)$ or

$$\begin{aligned} d &= Lb, \qquad L = L_{n-1}(\alpha_{n-1}) \cdots L_2(\alpha_2) L_1(\alpha_1) \\ x &= Uf, \qquad U = M_1^T D_1^{-1} \cdots M_{n-1}^T D_{n-1}^{-1} \end{aligned}$$

This gives rise to the following algorithm:

Algorithm 7.9.2 Primal Vandermonde System

Given distinct scalars $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $b = (b_1, b_2, \ldots, b_n)^T$ the following algorithm solves the primal Vandermonde system $Vx = b$:

```
x = pvand(α, b)
  x := b;
  for k = 1 : n − 1
    for j = n : (−1) : k + 1
      x_j := x_j − α_k ∗ x_{j−1}
    end
  end
  for k = n − 1 : (−1) : 1
    for j = k + 1 : n
      x_j := x_j/(α_j − α_{j−k})
    end
    for j = k : n − 1
      x_j := x_j − x_{j+1}
    end
  end
```

This is the **Björck–Pereyra algorithm**. It solves primal Vandermonde systems with only $\frac{1}{2}n(n+1)(3A+2M)$ operations, where A and M denotes one floating point addition and multiplication, respectively. Note also that no extra storage is needed since a can overwrite f.

Notes

Although the history of Gaussian elimination goes back at least to Chinese mathematicians about 250 B.C., there was no practical experience of solving large linear

systems until the advent of computers in the 1940s. Gaussian elimination was the first numerical algorithm to be subjected to a rounding error analysis. In 1946 there was a mood of pessimism about the stability of Gaussian elimination. Hotelling [44] had produced bounds showing that the error in the solution would be proportional to 4^n, which suggested that it would be impossible to solve even systems of modest order. A few years later J. von Neumann and H. H. Goldstein published more relevant error bounds. In 1948 A. M. Turing wrote a remarkable paper [63], where he formulated the LU factorization and introduced matrix condition numbers. The more or less final form of error analysis of Gaussian elimination was given by J. H. Wilkinson [65]. For a more detailed historical perspective of Gaussian elimination we refer to N. J. Higham [41, Sec. 9.13].

Rook pivoting for nonsymmetric matrices was introduced by Neal and Poole in [51]. Related pivoting strategies for symmetric indefinite matrices were introduced earlier by Fletcher [26].

The idea of doing only half the elimination for symmetric systems, while preserving symmetry is probably due to Gauss, who first sketched his elimination algorithm in 1809. The Cholesky method is named after Andre-Louis Cholesky, who was a French military officer. He devised his method to solve symmetric, positive definite system arising in a geodetic survey in Crete and North Africa just before World War I.

The literature on linear algebra is very extensive. For a theoretical treatise a classical source is Gantmacher [28, 1959]. Several nonstandard topics are covered in depth in two excellent volumes by Horn and Johnson [42, 1985] and [43, 1991].

An interesting survey of classical numerical methods in linear algebra can be found in Faddeev and Faddeeva [25, 1963], but many of the methods treated are now dated. A compact, lucid and modern presentation is given in Householder [45, 1964]. Bellman [7, 1960] is an original and readable complementary text.

An up to date and indispensable book for of anyone interested in computational linear algebra is Golub and Van Loan [35, 1996]. The book by Higham [41, 2002] is a wonderful and useful source book for information about the accuracy and stability of algorithms in numerical linear algebra. Other excellent textbooks on matrix computation include Stewart [60, 1998]. For results on on perturbation theory and related topics a very complete reference book is Stewart and Sun [61, 1990]. In particular, an elegant treatise on norms and metrics is found in [61, Chapter II].

Direct methods for sparse symmetric positive definite systems are covered in George and Liu [31, 1981], while a more general treatise is given by Duff et al. [22, 1986].

Bauer [6] was the first to study componentwise perturbation theory. This did not catch on in English publications until Skeel took it up in two papers [56, 1979], and [57, 1980].

A gallery of test matrices is documented in N. J. Higham [40]. available from the Web,; see also [41, Appendix D].

Bibliography

[1] Jan Ole Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.

[2] Edward Anderson, Zhaojun Bai, Christian Bischof, S. Blackford, J. Demmel, J. Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, A. McKenney, and Danny Sorensen, editors. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[3] Mario Arioli, James W. Demmel, and Iain S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.*, 10:165–190, 1989.

[4] Cleve Ashcraft, Roger G. Grimes, and John G. Lewis. Accurate symmetric indefinite linear system solvers. *SIAM J. Matrix Anal. Appl.*, 20:2:513–561, 1998.

[5] Edgar Asplund. Inverses of matrices $\{a_{ij}\}$ which satisfy $a_{ij} = 0$ for $j > i + p$. *Math. Scand.*, 7:57–60, 1959.

[6] F. L. Bauer. Genauigkeitsfragen bei der Lösung linearer Gleichungssysteme. *Z. Angew. Math. Mech.*, 46:7:409–421, 1966.

[7] Richard Bellman. *Introduction to Matrix Analysis*. SIAM, Philadelphia, PA, 1995.

[8] Åke Björck and Tommy Elfving. Algorithms for confluent Vandermonde systems. *Numer. Math.*, 21:130–137, 1973.

[9] Åke Björck and Victor Pereyra. Solution of Vandermonde system of equations. *Math. Comp.*, 24:893–903, 1970.

[10] Z. Bothe. Bounds for rounding errors in the Gaussian elimination for band systems. *J. Inst. Maths. Applics.*, 16:133–142, 1975.

[11] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31:163–179, 1977.

[12] Eleanor Chu and Alan George. A note on estimating the error in gaussian elimination without pivoting. *ACM SIGNUM Newsletter*, 20:2:2–7, 1985.

[13] Philippe G. Ciarlet. *Introduction to Numerical Linear Algebra and Optimization*. Cambridge University Press, Cambridge, UK, 1989.

[14] Alan K. Cline, Cleve B. Moler, George W. Stewart, and James H. Wilkinson. An estimate for the condition number of a matrix. *SIAM J. Numer. Anal.*, 16:368–375, 1979.

[15] Carl de Boor and Allan Pinkus. Backward error analysis for totally positive linear systems. *Numer. Math.*, 27:485–490, 1977.

[16] James W. Demmel, Nicholas J. Higham, and Robert S. Schreiber. Stability of block LU factorizations. *Numer. Linear Algebra Appl.*, 2:173–190, 1995.

[17] Inderjit S. Dhillon. Reliable computation of the condition number of a tridiagonal matrix in $O(n)$ time. *SIAM J. Matrix Anal. Appl.*, 19:3:776–796, 1998.

[18] J. J. Dongarra, James R. Bunch, Cleve B. Moler, and George W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, PA, 1979.

[19] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1988.

[20] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. A extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 14:1–17, 1988.

[21] Jeremy Du Croz and Nicholas J. Higham. Stability of methods for matrix inversion. *IMA J. Numer. Anal.*, 12:1–19, 1992.

[22] Iain S. Duff, A. M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[23] Iain S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1:1–14, 1989.

[24] Erik Elmroth, F. G. Gustavson, Isak Jonsson, and Bo Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46:1, 2004.

[25] D. K. Faddeev and V. N. Faddeeva. *Computational Methods of Linear Algebra*. W. H. Freeman, San Francisco, CA, 1963.

[26] Roger Fletcher. Factorizing symmetric indefinite matrices. *Linear Algebra Appl.*, 14:257–272, 1976.

[27] George E. Forsythe and Cleve B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[28] F. R. Gantmacher. *The Theory of Matrices. Vols. I and II*. Chelsea Publishing Co, New York, 1959.

[29] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and G. W. Stewart. *Matrix Eigensystems Routines: EISPACK Guide Extension*. Springer-Verlag, New York, 1977.

[30] Alan George, Kkakim D. Ikramov, and Andrey B. Kucherov. On the growth factor in Gaussian elimination for generalized Higham matrices. *Numer. Linear Algebra Appl.*, 9:107–114, 2002.

[31] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[32] Alan George and Joseph W. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[33] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 9:862–874, 1992.

[34] John R. Gilbert and Tim Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sc. Statist. Comput.*, 13:1:333–356, 1988.

[35] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[36] William W. Hager. Condition estimates. *SIAM J. Sci. Statist. Comput.*, 5:311–316, 1984.

[37] Eldon Hansen. *Topics in Interval Analysis*. Oxford University Press, Oxford, 1969.

[38] G. I. Hargreaves. Interval analysis in MATLAB. Numer. anal. report 418, Department of Mathematics, University of Manchester, 2002.

[39] Nicholas J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with application to condition estimation. *ACM Trans. Math. Software*, 14:4:381–396, 1988.

[40] Nicholas J. Higham. The Matrix Computation Toolbox, 1995. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[41] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, second edition, 2002.

[42] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.

[43] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991.

[44] Harold Hotelling. Some new methods in matrix calculus. *Ann. Math. Statist.*, 14:1–34, 1943.

[45] Alston S. Householder. *The Theory of Matrices in Numerical Analysis*. Dover, New York, 1975.

[46] Yasuhiko Ikebe. On inverses of Hessenberg matrices. *Linear Algebra Appl.*, 24:93–97, 1979.

[47] W. M. Kahan. Numerical linear algebra. *Canad. Math. Bull.*, 9:757–801, 1966.

[48] Charles L. Lawson, Richard J. Hanson, D. R. Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software*, 5:308–323, 1979.

[49] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, A. Kapur, M. C. Martin, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. LAPACK working note 149 Tech. Report CS-00-451, Department of Computer Science, University of Tennessee, Knoxville, TN, 2000.

[50] Jean Meinguet. Refined error analysis of Cholesky factorization. *SIAM J. Numer. Anal.*, 20:1243–1250, 1983.

[51] Larry Neal and George Poole. A geometric analysis of Gaussian elimination. II. *Linear Algebra Appl.*, 173:239–264, 1992.

[52] John von Neumann. In A. H. Taub, editor, *Collected Works*. Pergamon Press, New York, 1962.

[53] G. Peters and James H. Wilkinson. On the stability of Gauss–Jordan elimination with pivoting. *Comm. ACM*, 18:20–24, 1975.

[54] S. M. Rump. Fast and parallel interval arithmetic. *BIT*, 39:3:534–554, 1999.

[55] Siegfrid M. Rump. INTLAB—INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[56] Robert D. Skeel. Scaling for stability in Gaussian elimination. *J. Assoc. Comput. Mach.*, 26:494–526, 1979.

[57] Robert D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.*, 35:817–832, 1980.

[58] B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystems Routines—EISPACK Guide*. Springer-Verlag, New York, second edition, 1976.

[59] Torsten Söderström and G. W. Stewart. On the numerical properties of an iterative method for computing the Moore–Penrose generalized inverse. *SIAM J. Numer. Anal.*, 11:61–74, 1974.

[60] George W. Stewart. *Matrix Algorithms Volume I: Basic Decompositions*. SIAM, Philadelphia, PA, 1998.

[61] George W. Stewart and Ji guang. Sun. *Matrix Perturbation Theory*. Academic Press, Boston, MA, 1990.

[62] Lloyd N. Trefethen and Robert S. Schreiber. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11:335–360, 1990.

[63] A. M. Turing. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.*, 1:287–308, 1948.

[64] James M. Varah. On the solution of block-tridiagonal systems arising from certain finite-difference equations. *Math. Comp.*, 26(120):859–869, 1972.

[65] James H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. ACM*, 8:281–330, 1961.

[66] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.

[67] James H. Wilkinson. A priori error analysis of algebraic processes. In *Proceedings International Congress Math.*, pages 629–639. Izdat. Mir, Moscow, 1968.

[68] James H. Wilkinson and C. Reinsch, editors. *Handbook for Automatic Computation. Vol. II, Linear Algebra*. Springer-Verlag, New York, 1971.

[69] Max A. Woodbury. Inverting modified matrices. Memorandum Report 42, Statistical Research Group, Princeton, 1950.

Index