Lecture Notes

Direct Methods for Solving Linear Systems

Davoud Mirzaei

Department of IT, Uppsala University

August 24, 2026

Contents

1 Gaussian Elimination **2**
1.1 Introduction . . . 2
1.2 Triangular systems . . . 3
1.3 Gaussian elimination . . . 5
1.4 In-place implementation . . . 10
1.5 Computational cost . . . 11
1.6 Gaussian elimination with partial pivoting . . . 11
1.7 Complete pivoting . . . 15

2 Stability and error analysis of Gaussian elimination **15**
2.1 A model of floating-point arithmetic . . . 16
2.2 Forward error, residual, and backward error . . . 17
2.3 Conditioning of a linear system . . . 18
2.4 From backward error to forward error . . . 19
2.5 Rounding errors in Gaussian elimination . . . 20
2.6 Element growth and the growth factor . . . 20
2.7 Backward error of the computed LU factors . . . 22
2.8 Backward error of the computed solution . . . 23
2.9 Residuals and the assessment of a computed solution . . . 23
2.10 Iterative refinement . . . 24
2.11 Partial versus complete pivoting . . . 24
2.12 Special cases in which pivoting is unnecessary . . . 25
2.13 Practical interpretation . . . 25

3 Symmetric positive definite matrices **26**
3.1 Cholesky factorization . . . 28

4 Additional exercises **31**

1 Gaussian Elimination

1.1 Introduction

Systems of linear equations appear whenever one seeks unknown quantities that are related by linear equations. For example, discretizations of ordinary and partial differential equations, least-squares approximation, optimization algorithms, and network problems all require the solution of one or more linear systems. In this lecture we consider the system

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

where $[x_1, x_2, \ldots, x_n]^T =: x$ is the unknown solution, and $(a_{ij}) =: A$ and $[b_1, b_2, \ldots, b_n]^T := b$ are given. We may write this system as

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

where

$$A \in \mathbb{R}^{n \times n}, \qquad x, b \in \mathbb{R}^n,$$

We always (in this lecture) assume that A is nonsingular so that the system possesses a unique solution. There are two broad classes of methods for solving linear systems.

- *Direct methods*, which attempt to compute the exact solution in a finite number of arithmetic operations (up to rounding errors).
- *Iterative methods*, which construct a sequence of approximations converging to the exact solution.

The present lecture is devoted to the most important direct method, *Gaussian elimination*. The algorithm is one of the cornerstones of numerical linear algebra. Nearly every scientific computing package contains an optimized implementation of Gaussian elimination, and many more advanced algorithms are built upon the same principles.

The basic idea of Gaussian elimination is simple. Instead of solving the original system directly, we repeatedly transform it into an equivalent system that is easier to solve. During each step we eliminate one unknown from the remaining equations.

Besides being an algorithm for solving linear systems, Gaussian elimination produces one of the most useful *matrix factorizations* in numerical linear algebra, namely the LU factorization,

$$A = LU$$

where L is a lower triangular and U is an upper triangular. The solution of system (1.1) then reduces to solving two triangular systems

$$Ly = b, \quad Ux = y$$

which can be solved easily by substitution. This factorization also allows us to solve several systems with the same coefficient matrix efficiently.

The purpose of this lecture is to introduce the Gaussian elimination. The development of the algorithm then raises some natural questions.

- Under what conditions does Gaussian elimination exist?
- How expensive is the algorithm?
- Is the computed solution numerically reliable?

We will also answer these questions. However, we first study how triangular systems are solved, since Gaussian elimination transforms a general system into a triangular one.

1.2 Triangular systems

A linear system is particularly easy to solve when its coefficient matrix is triangular. Since Gaussian elimination transforms a general matrix into an upper triangular matrix, triangular systems form the final stage of the algorithm.

There are two types of triangular matrices. A matrix is called *upper triangular* if all entries below the main diagonal are zero,

$$u_{ij} = 0, \qquad i > j,$$

whereas a matrix is called *lower triangular* if all entries above the diagonal are zero,

$$\ell_{ij} = 0, \qquad i < j.$$

Triangular systems can be solved by forward or backward substitution because each equation contains only a subset of the unknowns.

Backward substitution

Consider the upper triangular system

$$Ux = y,$$

where

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix},$$

and assume that every diagonal entry is nonzero, i.e. $u_{ii} \neq 0$, for $i = 1, \ldots, n$. Writing the equations explicitly, we obtain

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= y_1, \\ u_{22}x_2 + \cdots + u_{2n}x_n &= y_2, \\ &\vdots \\ u_{nn}x_n &= y_n. \end{aligned}$$

The last equation contains only one unknown and therefore immediately yields

$$x_n = \frac{y_n}{u_{nn}}.$$

Once x_n has been computed, the previous equation contains only the unknown x_{n-1},

$$u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = y_{n-1},$$

which gives

$$x_{n-1} = \frac{1}{u_{n-1,n-1}}\left(y_{n-1} - u_{n-1,n}x_n\right)$$

Proceeding upward through the equations leads to the recursive formula

$$x_i = \frac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right), \qquad i = n-1, n-2, \ldots, 1. \tag{1.2}$$

The process is called *backward substitution* because the unknowns are determined from the bottom equation upward.

Theorem 1.1. Suppose that U is an upper triangular matrix whose diagonal entries are all nonzero. Then the system $Ux = y$ possesses a unique solution, and backward substitution computes this solution exactly in exact arithmetic.

Proof. The last equation uniquely determines x_n. Assume that $x_{i+1}, \ldots, x_n$ have already been computed uniquely. Since $u_{ii} \neq 0$, equation i uniquely determines x_i through formula (1.2). By induction, every unknown is uniquely determined. □

Forward substitution

The solution of lower triangular systems follows exactly the same principle, except that the equations are processed in the opposite direction. Consider

$$Lx = b,$$

where

$$L = \begin{bmatrix} \ell_{11} & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \ell_{n1} & \cdots & \ell_{n,n-1} & \ell_{nn} \end{bmatrix},$$

and assume that all diagonal entries are nonzero. The first equation immediately gives

$$x_1 = \frac{b_1}{\ell_{11}}.$$

After computing the first $i-1$ unknowns, the ith equation becomes

$$\ell_{ii}x_i = b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j,$$

which yields

$$x_i = \frac{1}{\ell_{ii}} \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right), \qquad i = 2, \ldots, n. \tag{1.3}$$

This procedure is known as *forward substitution*.

Theorem 1.2. If L is lower triangular with nonzero diagonal entries, then the system $Lx = b$ has a unique solution, and forward substitution computes that solution exactly in exact arithmetic.

The proof is completely analogous to the proof for backward substitution.

Computational cost

In backward substitution (1.2), the computation of x_i requires approximately $2(n-i)$ multiplications and additions. The total cost therefore is

$$2 \sum_{i=1}^{n-1} (n-i) = n(n-1) = 2n^2 - n = \mathcal{O}(n^2)$$

floating point operations (flops). Forward substitution (1.3) has the same cost. This quadratic complexity should be compared with the cubic complexity of Gaussian elimination that we shall derive later.

1.3 Gaussian elimination

The purpose of Gaussian elimination is to transform a general linear system into an equivalent upper triangular system. The key observation is that replacing one equation by a linear combination of itself and another equation does not change the solution set of the system. Such operations are called *elementary row operations*. They consist of

1. interchanging two rows,
2. multiplying a row by a nonzero scalar,
3. adding a multiple of one row to another.

Gaussian elimination uses the third operation (and later row interchanges when pivoting is introduced).

In this section we assume, unless otherwise stated, that all *pivots* generated during the elimination process are nonzero. Later in the chapter we shall see that this assumption is not always satisfied and discuss how pivoting overcomes this difficulty.

Consider the linear system (1.1) where

$$A^{(1)} = A = \left(a_{ij}^{(1)} \right)_{i,j=1}^{n} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}, \qquad b^{(1)} = b = \left(b_i^{(1)} \right)_{i=1}^{n} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}.$$

and assume first that

$$a_{11}^{(1)} \neq 0.$$

Our first objective is to eliminate all entries below the first pivot $a_{11}^{(1)}$. For every row $i = 2, \ldots, n$, we replace row i by

$$(\text{row } i) \leftarrow (\text{row } i) - \ell_{i1}(\text{ row } 1),$$

where

$$\ell_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, \ldots, n.$$

The numbers ℓ_{i1} are called the *elimination multipliers*. The entries of the new matrix are denoted by $a_{ij}^{(2)}$, and are computed as

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \ell_{i1} a_{11}^{(1)}, \quad i, j = 2, \ldots, n.$$

They are chosen so that the first entry of every modified row becomes zero, since

$$a_{i1}^{(2)} = a_{i1}^{(1)} - \ell_{i1} a_{11}^{(1)} = a_{i1}^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} a_{11}^{(1)} = 0, \quad i = 2, \ldots, n.$$

The right-hand side vector $b^{(1)}$ is modified accordingly:

$$b_i^{(2)} = b_i^{(1)} - \ell_{i1} b_1^{(1)}, \quad i = 2, \ldots, n.$$

Consequently, after the first elimination step the matrix and right-hand side vector have the form

$$A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}, \qquad b^{(2)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}.$$

Notice that the first row remains unchanged. Only the rows below it are modified. The elimination process now continues with the trailing $(n-1) \times (n-1)$ submatrix. Assuming that the new diagonal entry $a_{22}^{(2)}$ is nonzero, we eliminate the entries below it by subtracting suitable multiples of the second row. We repeating this procedure successively for each column.

Suppose that the first $k-1$ columns have already been eliminated. The matrix and the vector therefore have the form

$$A^{(k)} = \begin{bmatrix} a_{11}^{(0)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{bmatrix}, \qquad b^{(k)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}.$$

Assume that the current pivot satisfies

$$a_{kk}^{(k)} \neq 0.$$

For every row below the pivot, define

$$\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, n. \tag{1.4}$$

The remaining submatrix and the right hand side vector are then updated to

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \ell_{ik} a_{kk}^{(k)}, \quad i, j = k+1, \ldots, n, \tag{1.5}$$

$$b_i^{(k+1)} = b_i^{(k)} - \ell_{ik}\, b_k^{(k)}, \quad i = k+1, \ldots, n. \tag{1.6}$$

Since

$$a_{ik}^{(k)} - \ell_{ik} a_{kk}^{(k)} = 0, \quad i = k+1, \ldots, n,$$

all entries below the pivot disappear. Because only rows below the pivot are modified, all zeros created during previous elimination steps remain unchanged. Consequently, the elimination process never destroys the triangular structure that has already been established.

After the $(n-1)$st elimination step, every entry below the diagonal has been eliminated and the matrix has become upper triangular,

$$U := A^{(n)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix}, \qquad y := b^{(n)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}.$$

The original linear system has therefore been transformed into the equivalent system

$$Ux = y,$$

where U is upper triangular and y is the transformed right-hand side. The solution is then obtained by backward substitution.

Matrix interpretation

The row operations performed during Gaussian elimination can be represented by matrix multiplication. To illustrate the idea, consider the first elimination step. Define the matrix

$$L_1 := \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -\ell_{21} & 1 & 0 & \cdots & 0 \\ -\ell_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\ell_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix} = I - \ell_1 \widehat{e}_1^T, \quad \text{where} \quad \ell_1 = \begin{bmatrix} 0 \\ \ell_{21} \\ \ell_{31} \\ \vdots \\ \ell_{n1} \end{bmatrix}$$

and $\widehat{e}_1^T = [1, 0, \ldots, 0]$ is the first coordinate vector. Multiplying A from the left by L_1 subtracts the appropriate multiple of the first row from every row below it. Hence,

$$A^{(2)} = L_1 A,$$

Similarly, the second elimination step is represented by another matrix $L_2 = I - \ell_2 \widehat{e}_2^T$ where $\ell_2 = [0, 0, \ell_{32}, \ldots, \ell_{n2}]^T$. In general, at step k we define

$$L_k := \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & \cdots & 0 \\ 0 & 0 & \cdots & -\ell_{k+1,k} & \cdots & 0 \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\ell_{n,k} & \cdots & 1 \end{bmatrix} = I - \ell_k \widehat{e}_k^T, \quad \text{where} \quad \ell_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \ell_{k+1,k} \\ \vdots \\ \ell_{n,k} \end{bmatrix}$$

and $\widehat{e}_k^T = [0, \ldots, 0, 1, 0, \ldots, 0]$ is the kth coordinate vector. According to update equation (1.5) We have

$$A^{(k+1)} = L_k A^{(k)}, \quad k = 1, 2, \ldots, n-1.$$

After the final elimination step, we have

$$U = A^{(n)} = L_{n-1} L_{n-2} \cdots L_2 L_1 A,$$

where U is the resulting upper triangular matrix. This matrix interpretation is much more than a convenient notation. It shows that Gaussian elimination is actually a sequence of elementary matrix transformations, and it immediately leads to the LU factorization,

$$A = LU$$

where

$$L = (L_{n-1} L_{n-2} \cdots L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}.$$

We note that all matrices L_k are non-singular (why?) and we can show that $L_k^{-1} = I + \ell_k \widehat{e}_k^T$ because

$$\left(I - \ell_k \widehat{e}_k^T\right)\left(I + \ell_k \widehat{e}_k^T\right) = I - \ell_k (\widehat{e}_k^T \ell_k) \widehat{e}_k^T = I$$

since the inner product $\widehat{e}_k^T \ell_k$ is zero. It is also straightforward to show that

$$L = L_1^{-1} \cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix}.$$

It is important to emphasize that the algorithm described above assumes that every pivot $a_{kk}^{(k)}$ encountered during the elimination process is nonzero. If one of the pivots vanishes, the algorithm breaks down because the corresponding multipliers cannot be computed. Even when all pivots are nonzero, very small pivots may produce very large multipliers, which can seriously degrade the numerical accuracy of the computed solution. This issue will motivate the introduction of pivoting later in this lecture.

Example 1.1. Consider the system $Ax = b$, where

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & -6 & 0 & 1 \\ -2 & 7 & 2 & 3 \\ 6 & 1 & 5 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The first pivot is $a_{11} = 2$. Since the pivot is nonzero, elimination can proceed. The multipliers for the first column are

$$\ell_{21} = \frac{4}{2} = 2, \qquad \ell_{31} = \frac{-2}{2} = -1, \qquad \ell_{41} = \frac{6}{2} = 3.$$

Subtracting suitable multiples of the first row from the remaining rows, i.e.,

$$a_{ij} \leftarrow a_{ij} - \ell_{i1} a_{11}, \quad b_i \leftarrow b_i - \ell_{i1} b_1, \quad i, j = 2, \ldots, 4,$$

give the updated matrix $A^{(2)}$ and updated vector $b^{(2)}$ as

$$A^{(2)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -8 & -2 & 1 \\ 0 & 8 & 3 & 3 \\ 0 & -2 & 2 & 4 \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} 1 \\ -1 \\ 2 \\ -2 \end{bmatrix}.$$

Observe that the first column below the diagonal now consists entirely of zeros. The first row remains unchanged. The second pivot equals $a_{22} = -8$. The corresponding multipliers are

$$\ell_{32} = \frac{8}{-8} = -1, \qquad \ell_{42} = \frac{-2}{-8} = \frac{1}{4}.$$

The second elimination step is therefore

$$a_{ij} \leftarrow a_{ij} - \ell_{i2} a_{22}, \quad b_i \leftarrow b_i - \ell_{i2} b_2, \quad i, j = 3, 4.$$

Updates become

$$A^{(3)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -8 & -2 & 1 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & \frac{5}{2} & \frac{15}{4} \end{bmatrix}, \quad b^{(3)} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -\frac{7}{4} \end{bmatrix}.$$

The third and final pivot is $a_{33} = 1$, and the remaining multiplier is

$$\ell_{43} = \frac{5}{2}.$$

The final update reads as

$$a_{43} \leftarrow a_{43} - \ell_{43} a_{33}, \quad b_4 \leftarrow b_4 - \ell_{43} b_3.$$

which gives the final upper triangular matrix and the final right-hand side vector y,

$$U = A^{(4)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -8 & -2 & 1 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & -\frac{25}{4} \end{bmatrix}, \quad y = b^{(4)} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -\frac{17}{4} \end{bmatrix}.$$

The original system has now been transformed into the equivalent upper triangular system $Ux = y$. The solution is obtained by backward substitution,

$$x_4 = \frac{17}{25} = 0.68, \quad x_3 = -1.72, \quad x_2 = 0.64, \quad x_1 = 1.04.$$

Collecting the elimination multipliers ℓ_{ij} in the lower part of matrix L,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{21} & 1 & 0 & 0 \\ \ell_{31} & \ell_{32} & 1 & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 3 & 0.25 & 2.5 & 1 \end{bmatrix},$$

results in LU factorization $A = LU$ for matrix A.

1.4 In-place implementation

Gaussian elimination can be carried out *in place*. That is, the original matrix is gradually overwritten during the elimination process. Also the elimination multipliers are not stored in a separate matrix L. Once column k has been eliminated, the entries

$$a_{k+1,k}, a_{k+2,k}, \ldots, a_{nk}$$

have all become zero. These locations are therefore no longer needed by the elimination algorithm. Instead of storing zeros, we simply overwrite these locations with the corresponding multipliers,

$$\ell_{k+1,k}, \ell_{k+2,k}, \ldots, \ell_{nk}.$$

At the end of the computation, the strictly upper triangular part contains the entries of the matrix U, while the strictly lower triangular part contains the elimination multipliers. The final array has the form

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ \ell_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\ \ell_{31} & \ell_{32} & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & u_{nn} \end{bmatrix}.$$

The diagonal belongs to the matrix U. Since the diagonal of the matrix L consists entirely of ones, there is no need to store these entries explicitly.

We note that if the only objective is to solve the linear system $Ax = b$, there is no need to store the matrix L. Its effect has already been incorporated into the updated right-hand side vector b during the elimination process. Thus, only the upper triangular matrix U and the transformed vector b are required to compute the solution by backward substitution. In some applications, however, the LU factorization itself is needed, for example when solving multiple linear systems with the same coefficient matrix or when the factors are used in subsequent computations.

Algorithm 1: Gaussian elimination without pivoting

Input: Nonsingular matrix $A \in \mathbb{R}^{n\times n}$; vector $b \in \mathbb{R}^n$;
Output: Array A with U placed in its upper triangular part and L (excluding its unit diagonal) in its lower triangular part; updated vector b;

```
for k = 1 to n − 1 do
    for i = k + 1 to n do
        ℓ = a_ik / a_kk
        for j = k + 1 to n do
            a_ij ← a_ij − ℓ a_kj
        end for
        b_i ← b_i − ℓ b_k
        a_ik = ℓ
    end for
end for
```

1.5 Computational cost

We now estimate the computational cost of Gaussian elimination. During elimination step k, the trailing submatrix has dimension $(n-k)\times(n-k)$. For each of the $n-k$ rows below the pivot, approximately $n-k$ multiplications and the same number of additions are required. Consequently, the dominant cost during elimination step k is $2(n-k)^2$ flops. Summing over all elimination steps gives

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

Using the identity

$$\sum_{r=1}^{n-1} r^2 = \frac{(n-1)n(2n-1)}{6},$$

we obtain

$$\frac{2}{6}(n-1)n(2n-1) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n.$$

Hence, ignoring the lower powers of n, the dominant computational cost of Gaussian elimination is

$$\frac{2}{3}n^3$$

flops. The cost of updating the vector b is of order n^2 as well as the costs for backward substitution, which are negligible compared with the cost of the elimination itself for large matrices. The cubic complexity of Gaussian elimination is optimal for classical dense direct methods.

1.6 Gaussian elimination with partial pivoting

In the previous section we assumed that every pivot encountered during the elimination process was nonzero. Under this assumption Gaussian elimination proceeds exactly as described and produces an upper triangular matrix. Unfortunately, this assumption is not always satisfied, even when the coefficient matrix is nonsingular. Moreover, even if all pivots are

nonzero, very small pivots may lead to severe rounding errors in floating-point arithmetic. Consequently, Gaussian elimination without pivoting is rarely used in practical computations.

The purpose of this section is to introduce *pivoting* (choosing a proper pivot element at each step), explain why it is necessary, and develop the Gaussian elimination algorithm with partial pivoting.

Why pivoting is necessary

To understand the need for pivoting, let us first consider the matrix

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

The matrix is clearly nonsingular since $\det(A) = -1 \neq 0$. Nevertheless, Gaussian elimination without pivoting cannot even begin because the first pivot is $a_{11} = 0$. The first elimination multiplier, $\ell_{21} = a_{21}/a_{11}$ is therefore undefined. The difficulty does not arise because the matrix is singular. If the two rows are interchanged, the matrix becomes

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

and Gaussian elimination proceeds without difficulty. This simple example shows that row interchanges may be necessary merely to allow the algorithm to continue. A second difficulty is more subtle and concerns numerical stability. Consider

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}.$$

The first pivot is nonzero, so Gaussian elimination without pivoting does not break down. However, the elimination multiplier becomes

$$\ell_{21} = \frac{1}{10^{-20}} = 10^{20}.$$

The second row is therefore replaced by

$$(\text{row } 2) - 10^{20}(\text{row } 1),$$

which generates entries of magnitude approximately 10^{20}, although all entries of the original matrix are of order one. Such unnecessarily large intermediate numbers amplify rounding errors and may lead to a completely inaccurate computed solution. The problem is not caused by the matrix itself but by the poor choice of pivot. The idea behind pivoting is therefore simple: before each elimination step, choose a more suitable pivot.

Partial pivoting

The most widely used pivoting strategy is called *partial pivoting*. Before eliminating column k, we search that column for the largest entry in absolute value below the current pivot. More precisely, we determine the index

$$p = \arg\max_{k \leq i \leq n} |a_{ik}|.$$

If $p \neq k$, rows k and p are interchanged. The elimination then proceeds exactly as before. The chosen pivot therefore satisfies

$$|a_{pk}| = \max_{k \leq i \leq n} |a_{ik}|.$$

Since the pivot is the largest entry in magnitude in the current column, all elimination multipliers satisfy

$$|\ell_{ik}| = \left| \frac{a_{ik}}{a_{kk}} \right| \leq 1, \qquad i = k+1, \ldots, n.$$

This simple inequality is one of the principal reasons for the success of partial pivoting. By preventing large multipliers, it greatly reduces the amplification of rounding errors during elimination.

Permutation matrices

Row interchanges can be represented by permutation matrices. A permutation matrix is obtained by permuting the rows of the identity matrix. For example,

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

interchanges the first two rows of any matrix A of size $3 \times n$, if A is left-multiplied by P. Every permutation matrix is nonsingular and satisfies

$$P^{-1} = P^T.$$

Thus permutation matrices are orthogonal matrices whose only nonzero entries are ones. Thus, solving the linear system $Ax = b$ for a nonsingular matrix A is equivalent to solving the row-permuted system

$$PAx = Pb.$$

This means that we must apply permutation on b, accordingly. During Gaussian elimination several row interchanges may be required. The product of the corresponding permutation matrices is again a permutation matrix. Consequently, all row exchanges performed during the algorithm may be represented by a single permutation matrix.

> **Exercise 1.1.** Show that if P is a permutation matrix then $P^{-1} = P^T$ (i.e. P is an orthogonal matrix). Also show that if P_1 and P_2 are two permutation matrices then $P = P_1 P_2$ is a permutation matrix.

The algorithm

Gaussian elimination with partial pivoting differs from ordinary Gaussian elimination only in the choice of the pivot. At elimination step k, perform the following stages:

1. Search column k from rows k to n and identify the largest entry in absolute value.
2. Interchange the current pivot row with the row containing this entry.

3. Compute the elimination multipliers $\ell_{ik} = a_{ik}/a_{kk}$, for $i = k+1, \ldots, n$.
4. Eliminate the entries below the pivot as in Gaussian elimination without pivoting.

Except for the additional row interchange, the algorithm is identical to Algorithm 1.

Algorithm 2: Gaussian elimination with partial pivoting

Input: Nonsingular matrix $A \in \mathbb{R}^{n\times n}$; vector $b \in \mathbb{R}^n$;
Output: Array A with U placed in its upper triangular part and L (excluding its unit diagonal) in its lower triangular part; updated vector b; permutation matrix P;

```
Set P = I
for k = 1 to n − 1 do
    p ← arg max_{k≤i≤n} |a_ik|
    if a_pk = 0 then stop and report A is singular
    if p ≠ k then
        interchange rows k and p of A (i.e. interchange a_kj and a_pj for j = 1, ..., n)
        interchange b_k and b_p
        interchange rows k and p of P
    end if
    for i = k + 1 to n do
        ℓ ← a_ik/a_kk
        for j = k + 1 to n do
            a_ij ← a_ij − ℓ a_kj
        end for
        b_i ← b_i − ℓ b_k
        a_ik ← ℓ
    end for
end for
```

Once the elimination has been completed the solution x can be computed by solving the triangular system $Ux = y$ using backward substitution. Here y is the last updated vector b.

Unlike Gaussian elimination without pivoting, the elimination no longer produces a factorization of the form $A = LU$. Instead, one obtains

$$PA = LU,$$

where $P = P_{n-1} \cdots P_2 P_1$ and each P_k is obtained by interchanging the rows k and p of the identity matrix in step k of the elimination process. This factorization is called the *LU factorization with partial pivoting*.

Exercise 1.2. Modify Algorithm 2 so that the permutation matrix P of size $n \times n$ is not formed or stored explicitly. Instead, use a permutation vector $v \in \mathbb{N}^n$, initialized by

$$v_i = i, \qquad i = 1, \ldots, n.$$

Whenever rows k and p are interchanged, interchange the corresponding entries v_k and v_p. Explain how the final vector v represents the permutation matrix P, and show how it can be used to compute the permuted right-hand side Pb. Compare the storage requirements of the two approaches.

1.7 Complete pivoting

Partial pivoting searches only within the current column. A more expensive strategy is *complete pivoting*, in which the largest entry in absolute value is sought in the entire trailing submatrix.

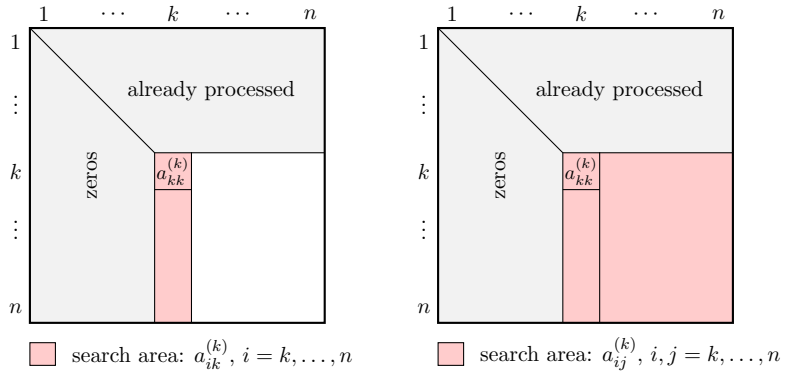

Figure 1: Pivot-search regions at step k. Partial pivoting (left), complete pivoting (right).

If the largest entry occurs in row p and column q, rows p and k, and columns q and k are interchanged before elimination continues. The resulting factorization has the form

$$PAQ = LU,$$

where both P and Q are permutation matrices, where P accounts for row permutation (as before) and Q for column permutation.

Exercise 1.3. Explain how the complete pivoting factorization can be used to solve the linear system $Ax = b$.

Complete pivoting generally provides a somewhat stronger theoretical guarantee of numerical stability. However, the additional search over the entire trailing submatrix increases the computational cost, while the improvement in accuracy is usually modest. For this reason complete pivoting is seldom used in practice. Partial pivoting provides a balance between efficiency and numerical stability and has become the standard pivoting strategy for dense linear systems. However, we should note that it does not guarantee perfect numerical behaviour for *every possible* matrix. There exist specially constructed matrices for which the intermediate entries generated during elimination become very large. Fortunately, such examples are extremely rare, and partial pivoting is highly successful for the overwhelming majority of practical problems.

2 Stability and error analysis of Gaussian elimination

In exact arithmetic, Gaussian elimination produces the exact solution of a nonsingular linear system, provided that the required pivots are nonzero. On a computer, however, arithmetic operations are carried out with finite precision, and the computed solution is generally dif-

ferent from the exact one. The purpose of this section is to understand the origin and size of this difference. Two distinct issues must be separated:

1. *conditioning of the mathematical problem*
2. *numerical stability of the algorithm.*

The first concerns how much does the exact solution change when the data A and b are perturbed? This property is measured by the condition number of A. The second concerns how large are the perturbations introduced by floating-point arithmetic? A numerically stable algorithm produces the exact solution of a nearby problem, but even such an algorithm may give a large forward error when the original problem is ill-conditioned.

We begin with a brief review of the floating-point model and then introduce forward error, residuals, and backward error. We give only a brief overview here. For a more detailed discussion, the reader is referred to the lecture notes *Fundamentals of Computing* by the author [2].

2.1 A model of floating-point arithmetic

Let u denote the unit roundoff of the floating-point system. Under the standard model of floating-point arithmetic, the result of a basic arithmetic operation satisfies

$$\mathrm{fl}(x \circ y) = (x \circ y)(1+\delta), \qquad |\delta| \le u,$$

where $\circ \in \{+, -, \times, /\}$, provided that no overflow or underflow occurs. The value of δ may be different for each operation. Thus, the model does not imply that all computations are affected by the same relative perturbation. It states only that each individual operation is performed with a relative error bounded by u.

When a sequence of floating-point operations is performed, products of factors of the form $1+\delta_i$ arise. The following notation is convenient:

$$\gamma_k = \frac{ku}{1-ku}, \qquad ku < 1.$$

A standard result states that if $|\delta_i| \le u$ for $i = 1, \dots, k$, then

$$\prod_{i=1}^{k} (1+\delta_i) = 1 + \theta_k, \qquad |\theta_k| \le \gamma_k.$$

For small ku, one has $\gamma_k \approx ku$. As an important consequence, the computed inner product of two vectors $x, y \in \mathbb{R}^n$ satisfies

$$\mathrm{fl}(x^T y) = x^T y + \Delta, \qquad |\Delta| \le \gamma_n |x|^T |y|,$$

where absolute values and inequalities involving vectors or matrices are understood componentwise.

2.2 Forward error, residual, and backward error

Let x denote the exact solution of

$$Ax = b,$$

and let $\widehat{x}$ denote a computed approximation. The quantity $\widehat{x} - x$ is called the *forward error*. Its normwise relative form is

$$\frac{\|\widehat{x} - x\|}{\|x\|},$$

which measures the disttance between the exact solution x and the computed solution $\widehat{x}$. The residual associated with $\widehat{x}$ is defined by

$$r = b - A\widehat{x}.$$

The *residual* measures how accurately the computed vector satisfies the original equations. Since

$$A(x - \widehat{x}) = r,$$

we have

$$x - \widehat{x} = A^{-1}r.$$

Consequently,

$$\|x - \widehat{x}\| \leq \|A^{-1}\| \, \|r\|.$$

A small residual does not by itself guarantee a small forward error. If $\|A^{-1}\|$ is large, a small residual may correspond to a large error in the solution. This is an expression of the sensitivity of the linear system.

Backward error takes a different point of view. Instead of asking how far $\widehat{x}$ is from the exact solution, it asks how much the original problem must be perturbed so that $\widehat{x}$ becomes an exact solution. For example, one may seek perturbations ΔA and Δb such that

$$(A + \Delta A)\widehat{x} = b + \Delta b.$$

An algorithm is called *backward stable* if the perturbations required to interpret the computed solution as an exact solution of a *nearby problem* are small relative to the original data. A simple normwise backward error is

$$\eta(\widehat{x}) = \min \left\{ \varepsilon : (A + \Delta A)\widehat{x} = b + \Delta b, \ \|\Delta A\| \leq \varepsilon \|A\|, \ \|\Delta b\| \leq \varepsilon \|b\| \right\}.$$

We can prove that if $\| \cdot \|$ is a vector norm, and let the matrix norm be the corresponding subordinate norm then $\eta(\widehat{x})$ can be written as

$$\eta(\widehat{x}) = \frac{\|b - A\widehat{x}\|}{\|A\| \, \|\widehat{x}\| + \|b\|}. \tag{2.1}$$

Indeed, using $r = b - A\widehat{x}$, we obtain $r = \Delta A\widehat{x} - \Delta b$. Therefore, by the triangle inequality and the subordinate property of the matrix norm,

$$\|r\| \leq \|\Delta A\widehat{x}\| + \|\Delta b\| \leq \|\Delta A\| \, \|\widehat{x}\| + \|\Delta b\|.$$

If $\|\Delta A\| \le \varepsilon\|A\|$ and $\|\Delta b\| \le \varepsilon\|b\|$, then

$$\|r\| \le \varepsilon\big(\|A\|\,\|\widehat{x}\| + \|b\|\big).$$

Hence every admissible value of ε satisfies

$$\varepsilon \ge \frac{\|r\|}{\|A\|\,\|\widehat{x}\| + \|b\|}.$$

We can also prove that this lower bound can be attained. We omit the proof here. From (2.1) we conclude that the residual becomes meaningful when it is scaled relative to the sizes of the matrix, the computed solution, and the right-hand side.

A componentwise backward error is often more informative when the entries of A and b vary greatly in magnitude. It is defined by

$$\omega(\widehat{x}) = \max_{1\le i\le n} \frac{|b_i - (A\widehat{x})_i|}{(|A||\widehat{x}| + |b|)_i},$$

with the convention that $0/0 = 0$. This quantity is the smallest ε for which there exist perturbations satisfying

$$|\Delta A| \le \varepsilon|A|, \qquad |\Delta b| \le \varepsilon|b|,$$

and

$$(A + \Delta A)\widehat{x} = b + \Delta b.$$

Here, $|A|$ denotes the matrix whose entries are $|a_{ij}|$, the absolute value of entries of A.

2.3 Conditioning of a linear system

The condition number of a nonsingular matrix A with respect to a subordinate matrix norm is

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

Since

$$\|I\| \le \|A\|\,\|A^{-1}\|,$$

one has

$$\kappa(A) \ge 1.$$

The condition number measures the sensitivity of the solution to perturbations in the data. To see this, consider the perturbed system

$$(A + \Delta A)(x + \Delta x) = b + \Delta b.$$

Subtracting $Ax = b$ gives

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

Therefore,

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

Taking norms yields

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

If $\|A^{-1}\|\,\|\Delta A\| < 1$, then

$$\frac{\|\Delta x\|}{\|x\|} \le \frac{\kappa(A)}{1-\kappa(A)\|\Delta A\|/\|A\|}\left(\frac{\|\Delta A\|}{\|A\|}+\frac{\|\Delta b\|}{\|b\|}\right),$$

where we have used

$$\|b\| = \|Ax\| \le \|A\|\,\|x\|.$$

For sufficiently small perturbations, the leading-order interpretation of this bound is

$$\frac{\|\Delta x\|}{\|x\|} \lesssim \kappa(A)\left(\frac{\|\Delta A\|}{\|A\|}+\frac{\|\Delta b\|}{\|b\|}\right).$$

Thus, if $\kappa(A)$ is moderate, small relative perturbations in the data produce small relative changes in the solution. If $\kappa(A)$ is large, the problem is ill-conditioned and the solution may be highly sensitive.

It is important to emphasize that *conditioning is a property of the mathematical problem, not of the algorithm used to solve it.* A numerically stable method cannot, in general, produce a highly accurate solution to a severely ill-conditioned problem.

2.4 From backward error to forward error

Suppose that a computed solution $\widehat{x}$ satisfies

$$(A+\Delta A)\widehat{x} = b + \Delta b.$$

Then $\widehat{x}$ is the exact solution of a nearby linear system. Applying the perturbation bound above gives

$$\frac{\|\widehat{x}-x\|}{\|x\|} \le \frac{\kappa(A)}{1-\kappa(A)\|\Delta A\|/\|A\|}\left(\frac{\|\Delta A\|}{\|A\|}+\frac{\|\Delta b\|}{\|b\|}\right),$$

provided that

$$\kappa(A)\frac{\|\Delta A\|}{\|A\|} < 1.$$

If the algorithm is backward stable, so that the relative perturbations $\frac{\|\Delta A\|}{\|A\|}$ and $\frac{\|\Delta b\|}{\|b\|}$ are of order u, then one expects

$$\frac{\|\widehat{x}-x\|}{\|x\|} = \mathcal{O}\big(\kappa(A)u\big).$$

This relation explains the different roles of stability and conditioning:

- the backward error measures the quality of the algorithm;
- the condition number measures the sensitivity of the problem;
- their product controls the forward error.

Consequently, a small forward error can generally be expected only when the algorithm is stable and the problem is reasonably well-conditioned.

2.5 Rounding errors in Gaussian elimination

We now consider Gaussian elimination in floating-point arithmetic. For simplicity, begin with elimination without pivoting. At step k, the exact multiplier is

$$\ell_{ik} = \frac{\widehat{a}_{ik}^{(k)}}{\widehat{a}_{kk}^{(k)}}, \qquad i = k+1, \dots, n,$$

where $\widehat{a}_{ij}^{(k)}$ denotes the entry present at the beginning of the kth elimination step. We use the hat notation for them because they have been affected by rounding errors in previous steps. Then the trailing matrix is updated according to (1.5). In floating-point arithmetic, both the multiplier and the update are rounded. Thus the computed quantities satisfy relations of the form

$$\widehat{m}_{ik} = \frac{\widehat{a}_{ik}^{(k)}}{\widehat{a}_{kk}^{(k)}}(1+\delta_{ik}), \qquad |\delta_{ik}| \le u,$$

and

$$\widehat{a}_{ij}^{(k+1)} = \mathrm{fl}\left(\widehat{a}_{ij}^{(k)} - \widehat{m}_{ik}\widehat{a}_{kj}^{(k)}\right).$$

The local error introduced by a single update is small relative to the quantities involved in that update. However, if the intermediate entries $\widehat{a}_{ij}^{(k+1)}$ become much larger than the entries of the original matrix, even small relative errors in the intermediate computations may correspond to large perturbations relative to A. This motivates the introduction of the growth factor.

2.6 Element growth and the growth factor

Let $A^{(k)} = (a_{ij}^{(k)})$ denote the matrix present at the beginning of the kth elimination step, with $A^{(1)} = A$. The *growth factor* is defined by

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

The numerator measures the largest entry generated at any stage of the elimination, while the denominator measures the largest entry in the original matrix. The growth factor satisfies

$$\rho_n \ge 1.$$

A large value of ρ_n indicates that the elimination process has created entries much larger than those in the original matrix. Since rounding errors are proportional to the magnitudes of the quantities being computed, large element growth may substantially increase the backward error.

Without pivoting, the multipliers may be arbitrarily large, and no useful general bound on ρ_n exists. For example, consider

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix}, \qquad 0 < \varepsilon \ll 1.$$

The first multiplier is $\ell_{21} = 1/\varepsilon$, and the second pivot becomes $1 - 1/\varepsilon$. Hence the growth factor is of order ε^{-1} and can be made arbitrarily large. Partial pivoting prevents this particular mechanism by ensuring that

$$|\ell_{ik}| \le 1.$$

Nevertheless, the updated entries are differences of the form

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \ell_{ik} a_{kj}^{(k)},$$

and therefore

$$|a_{ij}^{(k+1)}| \le |a_{ij}^{(k)}| + |a_{kj}^{(k)}| \le 2 \max_{r,s} |a_{rs}^{(k)}|.$$

It follows inductively that Gaussian elimination with partial pivoting satisfies the worst-case bound

$$\rho_n \le 2^{n-1}.$$

This exponential bound is sharp: there exist matrices for which equality is attained. See the following example.

Example 2.1 (Wilkinson matrix)**.** The worst-case growth-factor bound for Gaussian elimination with partial pivoting is attained by the Wilkinson matrix

$$W_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix}.$$

All entries of W_n have magnitude at most one, so

$$\max_{i,j} |w_{ij}| = 1.$$

When Gaussian elimination with partial pivoting is applied to W_n, no row interchange is required if ties are resolved by selecting the first available row. Indeed, at every stage all candidate pivots in the current column have magnitude one. The elimination multipliers are therefore

$$\ell_{ik} = -1, \qquad i = k+1, \dots, n.$$

At the first elimination step, adding the first row to each row below it changes the entries in the last column from 1 to 2. Thus, after the first step, the last column has the form

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 \end{bmatrix}^T.$$

At the second elimination step, the second row is added to all rows below it, and the entries in the last column below the second row become $2 + 2 = 4$. Continuing inductively, after elimination step k, the last-column entries below row k are equal to 2^k. Consequently, the

resulting upper triangular factor is

$$U = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 2^{n-2} \\ 0 & 0 & 0 & \cdots & 0 & 2^{n-1} \end{bmatrix}.$$

The largest entry generated during elimination is therefore

$$\max_{i,j,k} |w_{ij}^{(k)}| = 2^{n-1}.$$

Since the largest entry of the original matrix has magnitude one, the growth factor is $\rho_n = 2^{n-1}$. Hence the worst-case bound $\rho_n \leq 2^{n-1}$ for Gaussian elimination with partial pivoting is sharp.

However, for almost all matrices raised in practical applications the bound is very pessimistic. In most practical problems, the observed growth factor is modest, and Gaussian elimination with partial pivoting behaves very reliably. Nevertheless, the existence of matrices with exponential growth means that partial pivoting does not provide an unconditional backward-stability guarantee independent of the matrix.

2.7 Backward error of the computed LU factors

Suppose that Gaussian elimination, possibly with partial pivoting, is performed in floating-point arithmetic and produces computed factors $\widehat{L}$ and $\widehat{U}$. For partial pivoting, let P denote the computed permutation matrix. The computed factors satisfy a perturbed factorization

$$P(A + \Delta A) = \widehat{L}\widehat{U},$$

where, to first order, the perturbation obeys a componentwise bound of the form

$$|\Delta A| \leq \gamma_{cn} |\widehat{L}|\,|\widehat{U}|, \tag{2.2}$$

where c is a modest constant whose precise value depends on the implementation and the details of the rounding-error analysis. We refere the reader to book [1, 2002] for the details of the proof. This result is important because it states that the computed factors are the exact LU factors of a nearby matrix. However, the size of the perturbation is governed by the product

$$|\widehat{L}|\,|\widehat{U}|.$$

Under partial pivoting,

$$|\widehat{\ell}_{ij}| \leq 1, \qquad i > j,$$

so the entries of $\widehat{L}$ remain controlled. The entries of $\widehat{U}$, on the other hand, may grow during elimination, and their size is measured by the growth factor. A normwise consequence of (2.2) is

$$\frac{\|\Delta A\|}{\|A\|} \leq C_n u\, \rho_n + \mathcal{O}(u^2),$$

where C_n grows at most polynomially with n and depends on the chosen norm and implementation. Thus Gaussian elimination with partial pivoting has a small backward error whenever

$$C_n u \rho_n \ll 1.$$

Since ρ_n is usually modest in practice, the method is generally regarded as backward stable in a practical sense. More precisely, its backward stability is conditional on moderate element growth.

2.8 Backward error of the computed solution

After the factorization has been computed, the system is solved by forward and backward substitution. These triangular solves introduce additional rounding errors. The final computed solution $\widehat{x}$ can nevertheless be interpreted as the exact solution of a nearby system:

$$(A + \Delta A)\widehat{x} = b + \Delta b.$$

The perturbations satisfy a bound of the form

$$\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \leq C_n u \rho_n + O(u^2).$$

Combining this result with the perturbation theory for linear systems gives

$$\frac{\|\widehat{x} - x\|}{\|x\|} \lesssim \kappa(A) C_n u \rho_n,$$

provided that

$$\kappa(A) C_n u \rho_n \ll 1.$$

The estimate reveals three possible sources of an inaccurate computed solution:

1. the unit roundoff u may be too large for the required accuracy;
2. the growth factor ρ_n may be large, indicating instability in the elimination process;
3. the condition number $\kappa(A)$ may be large, indicating that the problem is intrinsically sensitive.

The first two are associated with the numerical computation, whereas the third is a property of the original linear system.

2.9 Residuals and the assessment of a computed solution

After computing an approximate solution $\widehat{x}$, one should not judge its quality solely by inspecting its entries. A natural first diagnostic is the residual

$$r = b - A\widehat{x}.$$

Since the entries of $A\widehat{x}$ may vary in scale, the unscaled residual norm $\|r\|$ is not always informative. A more meaningful quantity is the relative backward error

$$\eta(\widehat{x}) = \frac{\|b - A\widehat{x}\|}{\|A\|\,\|\widehat{x}\| + \|b\|}.$$

If $\eta(\widehat{x})$ is of the order of the unit roundoff, then $\widehat{x}$ is the exact solution of a nearby system. However, A small backward error does not necessarily imply a small forward error. The condition number must also be considered. Approximately,

$$\frac{\|\widehat{x} - x\|}{\|x\|} \lesssim \kappa(A)\eta(\widehat{x}).$$

Thus a computed solution may have an excellent residual and still possess few correct digits when A is severely ill-conditioned.

Conversely, a relatively large residual generally indicates that the computed solution is not numerically satisfactory, regardless of the condition number.

2.10 Iterative refinement

The accuracy of a computed solution can often be improved by iterative refinement. Suppose that an LU factorization of A has already been computed and that $\widehat{x}_0$ is the initial solution. At iteration m, compute the residual

$$r_m = b - A\widehat{x}_m.$$

Then solve the correction equation

$$A\delta_m = r_m$$

using the existing LU factors, and update

$$\widehat{x}_{m+1} = \widehat{x}_m + \delta_m$$

In exact arithmetic, the correction would satisfy

$$\delta_m = x - \widehat{x}_m,$$

and one iteration would recover the exact solution. In floating-point arithmetic, the correction is only approximate, but the procedure can still substantially reduce the forward and backward errors.

Iterative refinement is particularly effective when the residual is computed in higher precision than that used for the factorization and triangular solves. Modern mixed-precision algorithms exploit this observation by computing a relatively inexpensive low-precision factorization and recovering high accuracy through refinement.

2.11 Partial versus complete pivoting

Under complete pivoting, the largest entry in magnitude in the entire trailing submatrix is selected as the next pivot. This strategy controls both row and column growth more strongly than partial pivoting.

For complete pivoting, the worst-case growth factor is much smaller than the bound 2^{n-1} associated with partial pivoting. Nevertheless, the search for the largest entry in the entire trailing submatrix is more expensive and requires column interchanges in addition to row interchanges.

In most applications, the practical improvement in stability does not justify the additional cost and complexity. Partial pivoting is therefore the standard strategy for general dense linear systems. Complete pivoting may be useful when exceptionally strong reliability is required or when the matrix is known to have a structure that may cause substantial growth under partial pivoting.

2.12 Special cases in which pivoting is unnecessary

Although pivoting is essential for general matrices, there are important classes of matrices for which Gaussian elimination without pivoting is stable and well defined.

If A is symmetric positive definite, all leading principal submatrices are nonsingular, and elimination can be performed without row interchanges. In this case one normally uses the Cholesky factorization rather than a general LU factorization. More details will come in the next section.

Strictly row diagonally dominant matrices also admit elimination without pivoting. A matrix is strictly row diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \qquad i = 1, \ldots, n.$$

Such matrices are nonsingular, and their diagonal dominance prevents zero pivots during elimination.

Exercise 2.1. Suppose that A is strictly row diagonally dominant. Show that each trailing submatrix generated during Gaussian elimination is also strictly row diagonally dominant. Deduce that, at every elimination step k, the pivot element $a_{kk}^{(k)}$ is the largest entry in magnitude in the first column of the trailing submatrix. Hence, Gaussian elimination with partial pivoting performs no row interchanges.

Related stability results hold for some other structured classes, including certain banded matrices and M-matrices. Whenever such structure is known in advance, it may be possible to avoid pivoting and preserve the structure of the matrix more efficiently.

2.13 Practical interpretation

The main conclusions of the error analysis may be summarized as follows.

1. Gaussian elimination without pivoting may be unstable because small pivots can produce large multipliers and substantial element growth.
2. Partial pivoting ensures that the multipliers satisfy

   $$|\ell_{ik}| \leq 1,$$

   but it does not completely prevent growth in the entries of U.
3. The backward error of Gaussian elimination with partial pivoting is proportional to the unit roundoff, a polynomial factor in n, and the growth factor ρ_n.

4. Partial pivoting is therefore highly reliable when the growth factor is moderate, as it is for most matrices encountered in practice.
5. The forward error depends additionally on the condition number:

$$\text{(forward error)} \approx \text{(condition number)} \times \text{(backward error)}.$$

6. A large forward error does not necessarily indicate an unstable algorithm. It may instead reflect the intrinsic ill-conditioning of the linear system.
7. The scaled residual and backward error should be used to assess whether a computed solution satisfies a nearby linear system.

Gaussian elimination with partial pivoting combines moderate computational cost, efficient in-place storage, and practical stability (stable in almost all practical systems). For these reasons, it remains the standard direct method for solving general dense systems of linear equations.

3 Symmetric positive definite matrices

Symmetric positive definite matrices arise naturally in the numerical solution of differential equations, least-squares problems, optimization, statistics, covariance estimation, and many other areas of science and engineering. Such matrices enjoy many remarkable properties. In particular, they admit a factorization that is considerably simpler, more efficient, and more stable than the general LU factorization. This factorization is known as the *Cholesky factorization*. We first give the definition.

Definition 3.1. A real symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called *positive definite* if

$$x^T Ax > 0, \quad \text{for all } x \in \mathbb{R}^n \text{ with } x \neq 0.$$

A symmetric matrix A is called *positive semidefinite* if

$$x^T Ax \geq 0, \quad \text{for all } x \in \mathbb{R}^n \text{ with } x \neq 0.$$

The symmetry assumption is essential. Without symmetry, the quantity $x^T Ax$ alone does not adequately characterize the matrix.

Example 3.1. The identity matrix I is positive definite since

$$x^T Ix = x^T x = \|x\|_2^2 > 0,$$

for every nonzero vector x. More generally, every diagonal matrix $D = \text{diag}(d_1, \ldots, d_n)$ with positive diagonals is positive definite because

$$x^T Dx = \sum_{i=1}^{n} d_i x_i^2 > 0.$$

As another example, consider the symmetric matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

For every nonzero vector $x = [x_1, x_2]^T \neq 0$ we obtain

$$x^T Ax = 2x_1^2 + 2x_2^2 + 2x_1x_2 = (x_1 + x_2)^2 + x_1^2 + x_2^2 > 0.$$

Hence A is positive definite. The symmetric matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

is positive semidefinite. Because for $x = [x_1, x_2]^T \neq 0$ we have

$$x^T Ax = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_1x_2 + x_2^2 = (x_1 + x_2)^2 \geq 0.$$

Hence A is positive semidefinite. However, A is not positive definite since for $x = [1, -1] \neq 0$ we get $x^T Ax = 0$.

Positive definite matrices possess many useful properties. We list some of them in the following proposition.

Proposition 3.1. Let A be symmetric positive definite. Then

1. A^T is positive definite,
2. all diagonal entries satisfy $a_{kk} > 0$,
3. A is nonsingular and A^{-1} is also symmetric positive definite,
4. all eigenvalues are (strictly) positive,
5. the matrix B obtained by removing some rows and corresponding columns of A is positive definite,
6. every leading principal submatrix is symmetric positive definite,
7. the determinant of every leading principal submatrix is positive.

Proof.

1. For a nonzero vector $x \in \mathbb{R}^n$ we have $x^T A^T x = (x^T Ax)^T > 0$.
2. Choose $x = \widehat{e}_k \neq 0$ (the kth coordinate vector) which gives $0 < x^T Ax = \widehat{e}_k^T A\widehat{e}_k = a_{kk}$.
3. Assume that $Ax = 0$ for some nonzero vector x, then $x^T Ax = 0$ which contradicts positive definiteness. Hence A is nonsingular. Besides, since A is symmetric, A^{-1} is also symmetric. Assume that $x \neq 0$ is given. Since A is nonsingular there exists a vector $z \neq 0$ such that $Az = x$. Now we can write

$$x^T A^{-1} x = (Az)^T A^{-1}(Az) = z^T A^T A^{-1} Az = z^T A^T z > 0,$$

which shows that A^{-1} is positive definite.

4. Assume that λ is an eigenvalue of A corresponding to eigenvector $v \neq 0$. Since A is symmetric, both λ and v are real. We have $Av = \lambda v$. Multiplying by v^T we get $v^T Av = \lambda v^T v = \lambda \|v\|_2^2$. Since A is positive definite, $v^T Av > 0$, and since $v \neq 0$, $\|v\|_2 > 0$. Thus $\lambda > 0$.
5. Assume that rows and columns $i_1, i_2, \ldots, i_k$ of A are removed and we are left with submatrix B of size $(n-k) \times (n-k)$. For every nonzero vector $y \in \mathbb{R}^{n-k}$, we assume that $x \in \mathbb{R}^n$ is an expansion of y by inserting zeros in places $i_1, i_2, \ldots, i_k$. Then we have
$$y^T By = x^T Ax > 0,$$
because A is positive definite. Since y is arbitrary, B is positive definite.
6. This is a consequence of the previous item.
7. This is a consequence of the previous item.

□

The converses of most of the above statements are also true, which yield equivalent characterizations of positive definiteness.

Theorem 3.2. The symmetric A is positive definite if and only if all eigenvalues of A are positive.

Proof. We already showed that if A is symmetric positive definite then all eigenvalues are positive. We prove that if all eigenvalues of A are positive then A is positive definite. First, since A is symmetric all its eigenvalues and eigenvectors are real, and the set of its eigenvectors $\{v_1, v_2, \ldots, v_n\}$ forms an orthogonal basis for space $\mathbb{R}^n$, i.e. $v_i^T v_j = \delta_{ij}$. Given a non-zero vector $x \in \mathbb{R}^n$, one can write

$$x = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

where c_k are some coefficients. We then have

$$\begin{aligned} x^T Ax &= (c_1 v_1 + c_2 v_2 + \cdots + c_n v_n)^T A(c_1 v_1 + c_2 v_2 + \cdots + c_n v_n) \\ &= c_1 Av_1 + c_2 Av_2 + \cdots + c_n Av_n \\ &= c_1^2 \lambda_1 + c_2^2 \lambda_2 + \cdots + c_n^2 \lambda_n > 0, \end{aligned}$$

where we have used the orthogonality of vectors v_i. □

Theorem 3.3. The symmetric A is positive definite if and only if every leading principal submatrix of A has a positive determinant.

We leave the proof of this Theorem as an exercise. See also [3, 2010].

3.1 Cholesky factorization

One of the most important consequence of positive definiteness is the existence of a triangular square-root factorization.

Theorem 3.4 (Cholesky)**.** A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if and only if there exists a unique factorization

$$A = LL^T,$$

where L is lower triangular with strictly positive diagonal entries. The matrix L is called the *Cholesky factor* of A.

Proof. Suppose first that such a factorization exists. Then $A = LL^T$ is symmetric. Moreover, since the diagonal entries of L are nonzero, L is nonsingular. Therefore, for every $x \neq 0$,

$$x^T Ax = x^T LL^T x = (L^T x)^T (L^T x) = \|L^T x\|_2^2 > 0,$$

because $L^T x \neq 0$. Hence A is positive definite.

We now prove the converse by induction on the dimension. For $n = 1$, a positive definite matrix has the form $A = [a_{11}]$ with $a_{11} > 0$, and hence

$$A = [\,\sqrt{a_{11}}\,]\,[\sqrt{a_{11}}\,]^T.$$

Thus the result holds for 1×1 matrices. Assume that every symmetric positive definite matrix of order $k-1$ admits a Cholesky factorization. Let A be a symmetric positive definite matrix of order n. Partition A as

$$A = \begin{bmatrix} A_1 & w \\ w^T & a_{nn} \end{bmatrix},$$

Since every principal submatrix of a positive definite matrix is positive definite, A_1 is symmetric positive definite. By the induction hypothesis, there exists a lower triangular matrix L_1 with positive diagonal entries such that $A_1 = L_1 L_1^T$. We seek a lower triangular factor of the form

$$L = \begin{bmatrix} L_1 & 0 \\ u^T & \ell_{nn} \end{bmatrix}, \qquad \ell_{nn} > 0.$$

Multiplying the two factors gives

$$LL^T = \begin{bmatrix} L_1 L_1^T & L_1 u \\ u^T L_1^T & u^T u + \ell_{nn}^2 \end{bmatrix}.$$

To obtain $LL^T = A$, we must have

$$L_1 u = w$$

and

$$u^T u + \ell_{nn}^2 = a_{nn}.$$

Since L_1 is nonsingular, the first equation has the unique solution $u = L_1^{-1} w$. It remains to show that $a_{nn} - u^T u > 0$. Using $A_1 = L_1 L_1^T$ and $w = L_1 u$, we obtain

$$w^T A_1^{-1} w = u^T L_1^T (L_1 L_1^T)^{-1} L_1 u = u^T u.$$

Now consider the nonzero vector

$$z = \begin{bmatrix} -A_1^{-1} w \\ 1 \end{bmatrix}.$$

Since A is positive definite, $z^T Az > 0$. A direct calculation gives

$$z^T Az = a_{nn} - w^T A_1^{-1} w = a_{nn} - u^T u.$$

Therefore, $a_{nn} - u^T u > 0$. We may consequently define

$$\ell_{nn} = \sqrt{a_{nn} - u^T u} > 0.$$

With these choices of u and ℓ_{nn}, we have

$$LL^T = \begin{bmatrix} A_1 & w \\ w^T & a_{nn} \end{bmatrix} = A.$$

Thus every symmetric positive definite matrix admits a Cholesky factorization. It remains to prove uniqueness. This is what you are asked to prove in the next exercise. □

Exercise 3.1. Prove that the Cholesky factorization $A = LL^T$ for symmetric positive definite matrix A is unique.

The proof of the above theorem propose also an algorithm to compute the Cholesky factor L. We can also compute the the entries of L directly. Let $A = (a_{ij})$, and $L = (\ell_{ij})$. Since $A = LL^T$, the entries satisfy

$$a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik}\ell_{jk}.$$

Equating corresponding entries leads to the recursive formulas for $i = 1, 2, \ldots, n$,

$$\ell_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2}, \qquad \ell_{ji} = \frac{1}{\ell_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} \ell_{jk}\ell_{ik} \right), \qquad j = i+1, \ldots, n.$$

Since every leading principal submatrix is positive definite, the quantity under each square root is strictly positive, and therefore the algorithm never breaks down.

Algorithm 3: Cholesky factorization

Input: Symmetric positive definite matrix $A \in \mathbb{R}^{n \times n}$;
Output: Lower triangular matrix L such that $A = LL^T$;

for $i = 1$ **to** n **do**
  $\ell_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2}$
  for $j = i + 1$ **to** n **do**
    $\ell_{ji} = \frac{1}{\ell_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} \ell_{jk}\ell_{ik} \right)$
  end for
end for

Exercise 3.2. Show that the leading term in computation cost of the Cholesky algorithm for an $n \times n$ matrix A is

$$\frac{1}{3}n^3$$

and compare it with the LU decomposition using Gaussian elimination.

One of the most attractive features of the Cholesky factorization is its excellent numerical stability. Unlike general Gaussian elimination, no pivoting is required. Indeed, positive definiteness guarantees that every pivot is positive, so the factorization always exists and no breakdown can occur. Moreover, Cholesky factorization is backward stable. The computed factor $\widehat{L}$ satisfies

$$A + \Delta A = \widehat{L}\widehat{L}^T,$$

where

$$\frac{\|\Delta A\|}{\|A\|} = \mathcal{O}(u),$$

provided that the computation is performed in floating-point arithmetic. For a detailed proof see [1, 2002]. Since no pivoting is required, there is no analogue of the growth factor that appears in Gaussian elimination with partial pivoting. Consequently, Cholesky is generally both faster and more reliable than LU factorization whenever the coefficient matrix is symmetric positive definite.

4 Additional exercises

Exercise 4.1. Assume that the LU factorization $A = LU$ of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ has already been computed. Suppose that we want to solve

$$AX = B,$$

where $B \in \mathbb{R}^{n\times p}$ contains p right-hand sides.

1. Explain how the factorization $A = LU$ can be reused to compute all columns of X.
2. Determine the leading-order computational cost of solving for all p right-hand sides after the factorization is available.
3. Compare this with the cost of performing a new Gaussian elimination separately for every right-hand side. For what reason is a matrix factorization particularly useful when many systems with the same coefficient matrix must be solved?

Exercise 4.2. Let $A \in \mathbb{R}^{n\times n}$ be nonsingular. Show that Gaussian elimination without pivoting can be completed without encountering a zero pivot if and only if all leading principal submatrices

$$A_k = A(1:k, 1:k), \qquad k = 1, \ldots, n,$$

are nonsingular.

Deduce that, whenever Gaussian elimination without pivoting exists,

$$\det(A_k) = u_{11}u_{22}\cdots u_{kk},$$

where u_{jj} are the pivots, i.e., the diagonal entries of the upper triangular factor U.

Exercise 4.3. Suppose Gaussian elimination with partial pivoting is applied to a matrix A. Using the update formula show that

$$\max_{i,j} |a_{ij}^{(k+1)}| \leq 2 \max_{i,j} |a_{ij}^{(k)}|.$$

Use this relation inductively to derive the bound

$$\rho_n \leq 2^{n-1}$$

for the growth factor. Why does this estimate not imply that Gaussian elimination with partial pivoting is unstable in practice?

Exercise 4.4. Consider the family of linear systems

$$A_\varepsilon x = b_\varepsilon, \qquad A_\varepsilon = \begin{pmatrix} 1 & 1 \\ 1 & 1+\varepsilon \end{pmatrix}, \qquad b_\varepsilon = \begin{pmatrix} 2 \\ 2+\varepsilon \end{pmatrix},$$

where $0 < \varepsilon \ll 1$. The exact solution is $x = [1, 1]^T$ Consider instead the approximation $\widehat{x} = [0, 2]^T$ Using the infinity norm,

1. compute the relative forward error.
2. compute the residual $r = b_\varepsilon - A_\varepsilon \widehat{x}$ and its norm,
3. compute the normwise backward error $\eta(\widehat{x})$ in infinity norm,
4. Compute the condition number of A_ε in infinity norm.
5. explain how this example illustrates the difference between backward error and forward error.

Exercise 4.5. The residual of an approximate solution is sometimes used as an indicator of the accuracy of the solution. Let x be the exact solution of $Ax = b$, let $\widehat{x}$ be an approximation, and let $r = b - A\widehat{x}$.

1. Show that
$$\frac{\|x - \widehat{x}\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|A\| \, \|x\|}.$$
2. Suppose that $\|x\|$ is unknown. Use
$$\|b\| \leq \|A\| \, \|x\|$$
to derive a bound involving $\|r\|/\|b\|$.

3. Explain why a relative residual of size 10^{-12} may still correspond to a poor solution if $\kappa(A) \approx 10^{12}$.
4. On the other hand, what conclusion can be drawn if both the backward error and the condition number are small?

Exercise 4.6. Consider iterative refinement starting from an approximate solution $\widehat{x}_0$. At step m,

$$r_m = b - A\widehat{x}_m,$$

and a correction $\widehat{\delta}_m$ is computed from

$$A\widehat{\delta}_m \approx r_m,$$

followed by

$$\widehat{x}_{m+1} = \widehat{x}_m + \widehat{\delta}_m.$$

1. Show that if the correction equation were solved exactly, then one refinement step would give the exact solution in exact arithmetic.
2. Suppose instead that
$$\widehat{\delta}_m = (x - \widehat{x}_m) + d_m.$$
Show that the new error satisfies
$$x - \widehat{x}_{m+1} = -d_m.$$
3. Explain why iterative refinement can improve a solution even though the same LU factors are reused at every step.
4. Implement iterative refinement and test it on matrices with increasing condition numbers. At every step, record the forward error, residual norm, and backward error. Discuss the results.

Exercise 4.7. Let

$$A = \begin{pmatrix} A_1 & w \\ w^T & a \end{pmatrix}$$

be symmetric, where $A_1 \in \mathbb{R}^{(n-1)\times(n-1)}$ is symmetric positive definite.
Show that A is positive definite if and only if

$$a - w^T A_1^{-1} w > 0.$$

Hint: For $x = \begin{bmatrix} y \\ t \end{bmatrix}$ complete the square in the expression $x^T A x$.
The scalar

$$a - w^T A_1^{-1} w$$

is called the Schur complement of A_1 in A. Explain how this condition appears naturally in the inductive proof of the Cholesky factorization.

Exercise 4.8. Let A be symmetric positive definite with Cholesky factorization

$$A = CC^T,$$

where

$$C = \begin{pmatrix} c_{11} & & \\ * & \ddots & \\ * & * & c_{nn} \end{pmatrix}, \qquad c_{ii} > 0.$$

Define $D = \operatorname{diag}(c_{11}^2, \ldots, c_{nn}^2)$ and $L = C \operatorname{diag}\left(\frac{1}{c_{11}}, \ldots, \frac{1}{c_{nn}}\right)$.

1. Show that L is unit lower triangular.
2. Prove that
$$A = LDL^T.$$
3. Deduce an LU factorization of A from this decomposition.
4. Explain why the pivots produced by Gaussian elimination without pivoting are positive for a symmetric positive definite matrix.

Exercise 4.9. Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite and let $A = LL^T$ be its Cholesky factorization.

1. Show that $\det(A) = \prod_{i=1}^{n} \ell_{ii}^2$.
2. Deduce that $\log(\det(A)) = 2 \sum_{i=1}^{n} \log \ell_{ii}$.
3. Let $u \in \mathbb{R}^n$. Prove that $A + uu^T$ is also symmetric positive definite.
4. Is $A - uu^T$ necessarily positive definite? Give a counterexample.
5. More generally, show that $A - uu^T$ is positive definite if and only if $u^T A^{-1} u < 1$.
Hint: Write $u = Lz$ and reduce the question to the positive definiteness of $I - zz^T$.

References

[1] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 2nd edition, 2002.

[2] D. Mirzaei. *Fundamentals of Computing*, 2024. Lecture notes, Uppsala University. Available at `https://arxiv.org/abs/2412.19958`.

[3] D. S. Watkins. *Fundamentals of Matrix Computations*. John Wiley & Sons, Hoboken, NJ, 2nd edition 2002, 3rd edition, 2010.