CHAPTER 1

Linear equation solving

In this chapter we discuss the computational aspects of the equation

$$A\,x = b$$

and in particular, the basic algorithms for solving it together with their space and time complexities, and their numerical behavior. First we focus on the *direct methods* for this task, that is, methods that are exact in the absence of rounding. Later we will study some *iterative methods*, that produce sequences of approximations to the searched solution.

1.1. Preliminaries

We will deal with matrices having entries in a field $\mathbb{F}$ that can be either the field of real numbers $\mathbb{R}$ or to that of complex numbers $\mathbb{C}$. The space of $m \times n$ matrices with entries in $\mathbb{F}$ is denoted by

$$\mathbb{F}^{m\times n}.$$

The (i,j)-entry of an $m \times n$ matrix A is denoted by $a_{i,j}$, so that

$$A = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix}.$$

Similarly, the i-th entry of an n-vector x is denoted by x_i, so that $x = (x_1, \dots, x_n)$. By default, these vectors will be treated as *columns*, that is, $n \times 1$ matrices:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{1.1}$$

For an $m \times n$ matrix A, this identification allows to consider the linear map

$$L_A \colon \mathbb{F}^n \longrightarrow \mathbb{F}^m$$

defined by $L_A(x) = A\,x$, that is, the multiplication of the matrix A and a given n-vector x. This multiplication can be written as a linear combination of the columns of A:

$$A\,x = x_1 \operatorname{col}_1(A) + \cdots + x_n \operatorname{col}_n(A),$$

for instance

$$\begin{bmatrix} 2 & 3 \\ 2 & 4 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 \\ 4 \\ 7 \end{bmatrix}.$$

Hence the *image* of this linear map coincides with the *column space* of the matrix, that is, the linear span of its columns:

$$\mathrm{Im}(L_A) = \mathrm{span}(\mathrm{col}_1(A), \dots, \mathrm{col}_n(A)).$$

The *kernel* of this linear map is the set of vectors where it vanishes:

$$\mathrm{Ker}(L_A) = \{x \in \mathbb{F}^n \mid A\,x = 0\}.$$

The image and the kernel of L_A are linear subspaces of $\mathbb{F}^m$ and of $\mathbb{F}^n$, respectively. A fundamental theorem of linear algebra states that

$$\dim(\mathrm{Im}(L_A)) + \dim(\mathrm{Ker}(L_A)) = n$$

or equivalently,

$$\#\text{ independent columns } + \dim(\mathrm{Ker}(L_A)) = \#\text{ columns}.$$

An $r \times s$-*block matrix* is an $m \times n$ matrix written as

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,r} \\ \vdots & & \vdots \\ A_{s,1} & \cdots & A_{r,s} \end{bmatrix}$$

with *blocks* $A_{i,j}$ that are $m_i \times n_j$ matrices, for given sequences of nonnegative integers m_i, n_j, $i = 1, \dots, r$, $j = 1, \dots, s$, such that

$$\sum_{i=1}^{r} m_i = m \quad \text{and} \quad \sum_{j=1}^{s} n_j = n.$$

When we want to precise the sizes of these blocks, we say that A is an $(m_1, \dots, m_r) \times (n_1, \dots, n_s)$ *block matrix.*

Block matrices with corresponding blocks of equal sizes can be added through the block structure: if

$$B = \begin{bmatrix} B_{1,1} & \cdots & B_{1,r} \\ \vdots & & \vdots \\ B_{s,1} & \cdots & B_{r,s} \end{bmatrix}$$

and each $B_{i,j}$ is an $m_i \times n_j$ matrix, then

$$A + B = [A_{i,j} + B_{i,j}]_{i,j}.$$

Similarly, block matrices with blocks of compatible sizes can be multiplied in the natural way: for an $n \times p$ matrix written as

$$C = \begin{bmatrix} C_{1,1} & \cdots & C_{1,t} \\ \vdots & & \vdots \\ C_{s,1} & \cdots & C_{s,t} \end{bmatrix},$$

where each $C_{j,k}$ is an $n_j \times p_k$ for another sequence of nonnegative integers p_k, $k = 1, \dots, t$, such that $\sum_{k=1}^{t} p_k = p$, we have that

$$A\,C = \left[\sum_{j=1}^{s} A_{i,j}\, C_{j,k} \right]_{i,k}.$$

Unfortunately, the determinant of a block matrix cannot be computed with the usual formula applied to the blocks. Nevertheless, this formula does hold when the matrix is block triangular. For instance, if

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ \mathbb{0} & A_{2,2} \end{bmatrix}$$

then $\det(A) = A_{1,1}\, A_{2,2}$.

1.2. Gaussian elimination

1.2.1. From Gaussian elimination to the PLU factorization. The first and most fundamental problem of numerical linear algebra is to solve the equation

$$A\, x = b \tag{1.2}$$

for a given $m \times n$ matrix A and m-vector b in terms of an unknown n-vector x, all of them with entries in the field $\mathbb{F}$. We will restrict to the main case of interest, that arises when $m = n$ and A is *nonsingular*. By definition, a matrix is nonsingular when it admits an *inverse*, that is, a matrix A^{-1} such that $A\, A^{-1} = A^{-1} A = \mathbb{1}_n$ with

$$\mathbb{1}_n = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix},$$

the *identity* $n \times n$ matrix.

In general, the equation (1.2) being solvable is equivalent to the fact that b belongs to the image of the linear map L_A, that is $b \in \operatorname{Im}(L_A)$. When it exists, such a solution is unique if and only if the kernel of L_A is trivial, that is $\operatorname{Ker}(L_A) = \{0\}$. Hence

$$\boxed{\text{Equation (1.2) has a unique solution for all } b \iff m = n \text{ and } A \text{ is nonsingular}}$$

which is exactly the case we are going to study.

Gaussian elimination is the basic algorithm for this task. We next recall how it works in a simple example.

EXAMPLE 1.2.1. Consider the system of linear equations

$$\begin{cases} x_1 + 2\, x_2 = b_1 \\ 3\, x_1 + 4\, x_2 = b_2 \end{cases}$$

To simplify it, we subtract three times the first equation from the second, to eliminate the variable x_1 from it:

$$\begin{cases} x_1 + 2\, x_2 = b_1 \\ \quad\; - 2\, x_2 = -3\, b_1 + b_2 \end{cases}$$

We solve the obtained system of linear equations by *backward substitution*: we first compute the value of x_2 using the last equation and then plug it into the first one to get the value of x_1:

$$x_2 = \frac{1}{-2}(-3\, b_1 + b_2) = \frac{3}{2}\, b_1 - \frac{1}{2}\, b_2 \quad \text{and} \quad x_1 = b_1 - 2\, x_2 = -2\, b_1 + b_2.$$

In matrix notation, the system of linear equations in Example 1.2.1 writes down as

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}b_1\\ b_2\end{bmatrix},$$

and the elimination procedure therein is equivalent to a left multiplication of both sides of this vector equation by a lower triangular matrix:

$$\begin{bmatrix}1 & 2\\ 0 & -2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}1 & 0\\ -3 & 1\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}1 & 0\\ -3 & 1\end{bmatrix}\begin{bmatrix}b_1\\ b_2\end{bmatrix}=\begin{bmatrix}b_1\\ -3\,b_1+b_2\end{bmatrix}. \tag{1.3}$$

Setting

$$A=\begin{bmatrix}1 & 2\\ 0 & -2\end{bmatrix},\quad L=\begin{bmatrix}1 & 0\\ 3 & 1\end{bmatrix},\quad U=\begin{bmatrix}1 & 2\\ 0 & -2\end{bmatrix},$$

then $L^{-1}=\begin{bmatrix}1 & 0\\ -3 & 1\end{bmatrix}$ and so (1.3) implies that $L^{-1}A=U$, or equivalently that

$$A=L\,U.$$

Notice that the matrix L is *unit* lower triangular (all its diagonal entries are equal to 1) and U is upper triangular.

When some pivot vanishes, Gaussian elimination permutes the equations so that the algorithm can continue eliminating variables, as in the next example.

EXAMPLE 1.2.2. Consider the system of linear equations

$$\begin{cases} 2\,x_1 - x_2 \phantom{{}+x_3} = b_1\\ 2\,x_1 - x_2 + x_3 = b_2\\ -2\,x_1 + 3\,x_2 - x_3 = b_3\end{cases}$$

Gaussian elimination proceeds by subtracting the first equation from the second and adding the first equation to the third, thus eliminating the variable x_1 from both of them. In the resulting system, the coefficient for the variable x_2 in the second equation is zero. Hence the algorithm permutes this second equation with the third, to obtain a system of linear equations in row echelon form:

$$\begin{cases} 2\,x_1 - x_2 = b_1\\ 2\,x_2 - x_3 = b_1+b_3\\ x_3 = -b_1+b_2\end{cases}$$

As before, this system can be easily solved by backward substitution:

$$x_3=-b_1+b_2,\quad x_2=\frac{1}{2}(b_1+b_3+x_3)=\frac{1}{2}\,b_2+\frac{1}{2}\,b_3,$$
$$x_1=\frac{1}{2}(b_1+x_2)=\frac{1}{2}\,b_1+\frac{1}{4}\,b_2+\frac{1}{4}\,b_3. \tag{1.4}$$

In matrix notation, the system in Example 1.2.2 writes down as

$$\begin{bmatrix}2 & -1 & 0\\ 2 & -1 & 1\\ -2 & 3 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}b_1\\ b_2\\ b_3\end{bmatrix}$$

and the elimination procedure therein leads to the matrix identity

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ -1 & 1 & 0\\ 1 & 0 & 1\end{bmatrix}\begin{bmatrix}2 & -1 & 0\\ 2 & -1 & 1\\ -2 & 3 & -1\end{bmatrix}=\begin{bmatrix}2 & -1 & 0\\ 0 & 2 & -1\\ 0 & 0 & 1\end{bmatrix}. \tag{1.5}$$

Set P, L, A and U for the matrices successively appearing in this identity. We can verify by a direct computation that $(P\,L)^{-1} = P\,L$, and so (1.5) implies that

$$A = (P\,L)^{-1}U = P\,L\,U. \tag{1.6}$$

This factorization extends to the general case! We can deduce from the Gaussian elimination algorithm that our nonsingular $n \times n$ matrix A always factors as the product of a permutation, a unit triangular, and an upper triangular matrix. With this, the equation $A\,x = b$ reduces to similar equations for each of these factors that, thanks to their structure, are easy to solve.

Precisely, given the PLU factorization of the matrix A we can solve the linear equation $A\,x = b$ following the steps:

(1) solve $P\,z = b$ permuting the entries of b,
(2) solve $L\,y = z$ by forward substitution,
(3) solve $U\,x = y$ by backward substitution.

Example 1.2.3. Consider the PLU factorization in (1.6)

$$A = \begin{bmatrix} 2 & -1 & 0 \\ 2 & -1 & 1 \\ -2 & 3 & -1 \end{bmatrix} = P\,L\,U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

To solve $A\,x = b$ we first solve $P\,z = b$, which gives

$$z_1 = b_1, \quad z_2 = b_3, \quad z_3 = b_2.$$

Next we solve $L\,y = z$, which gives

$$y_1 = z_1 = b_1, \quad y_2 = z_2 + y_1 = b_1 + b_3, \quad y_3 = z_3 - y_1 = -b_1 + b_2.$$

Finally, the solution vector is computed by solving $U\,x = y$:

$$x_3 = y_3 = -b_1 + b_2, \quad x_2 = \frac{1}{2}(y_2 + x_3) = \frac{1}{2}\,b_2 + \frac{1}{2}\,b_3,$$
$$x_1 = \frac{1}{2}(y_1 + x_2) = \frac{1}{2}\,b_1 + \frac{1}{4}\,b_2 + \frac{1}{4}\,b_3,$$

in agreement with (1.4).

1.2.2. The GEPP algorithm. For a nonsingular $n \times n$ matrix A with entries in $\mathbb{F}$, its *PLU factorization* is its expression as a product of $n \times n$ matrices

$$A = P\,L\,U$$

with P a permutation, L unit lower triangular, and U upper triangular. As discussed in §1.2.1, this factorization solves the linear equation

$$A\,x = b$$

by reducing it to three analogous and simpler equations that can be solved directly. It gives us a more advanced point of view of the Gaussian elimination algorithm, and it is our first example of a matrix factorization that solves a linear algebra problem. In that section we also indicate how the standard version of Gaussian elimination implies the existence of this factorization and gives a method for computing it. However, it will be more convenient to discuss here from scratch that computation and its properties.

An *n-permutation* is a bijection $\sigma\colon \{1,\dots,n\}\to\{1,\dots,n\}$. It is usually written verbose as an explicit sequence of preimage/image pairs:

$$\sigma=\begin{pmatrix}1 & \cdots & n\\ \sigma(1) & \cdots & \sigma(n)\end{pmatrix}.$$

The set of all n-permutations is denoted by $\mathcal{S}_n$, and it forms a group with respect to the composition of functions.

To each $\sigma\in\mathcal{S}_n$ we associate the permutation $n\times n$-matrix obtained by permuting the columns of the identity $n\times n$ matrix according to σ, that is

$$P_\sigma=\begin{bmatrix}e_{\sigma(1)} & \cdots & e_{\sigma(n)}\end{bmatrix}$$

where the e_i's are the vectors in the standard basis of $\mathbb{F}^n$, that is

$$e_i=(0,\dots,0,\overset{i}{1},0,\dots,0)$$

plugged into P_σ as columns following our convention in (1.1).

Every permutation n-matrix is of this form, and so it can be coded by the corresponding element of $\mathcal{S}_n$. For instance

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

corresponds to the permutation $\begin{pmatrix}1&2&3\\1&3&2\end{pmatrix}$.

This correspondence is compatible with the group structure of $\mathcal{S}_n$: we have that

$$P_\sigma^{-1}=P_{\sigma^{-1}}=P_\sigma^T$$

and in particular, the inverse of a permutation n-matrix coincide with its transpose. Also, for $\sigma_1,\sigma_2\in\mathcal{S}_n$ we have that

$$P_{\sigma_1}P_{\sigma_2}=P_{\sigma_1\circ\sigma_2}.$$

Pre-multiplication of A by a permutation $n\times n$ matrix produces a permutation of its rows, whereas its post-multiplication produces a permutation of its columns. Precisely

$$P^TA=\begin{bmatrix}\mathrm{row}_{\sigma(1)}(A)\\ \vdots\\ \mathrm{row}_{\sigma(n)}(A)\end{bmatrix}\quad\text{and}\quad A\,P=\begin{bmatrix}\mathrm{col}_{\sigma(1)}(A) & \cdots & \mathrm{col}_{\sigma(n)}(A)\end{bmatrix}. \tag{1.7}$$

To compute the PLU factorization of A we apply an iterative procedure of pivoting and elimination by columns, as we next explain. Choose first $i_1\in\{1,\dots,n\}$ such that $a_{i_1,1}\neq 0$, let σ_1 be the n-permutation that swaps 1 and i_1 and consider the associated permutation matrix

$$P_{\sigma_1}.$$

Relabel the entries of A so that $P_{\sigma_1}^TA=\begin{bmatrix}a_{i,j}\end{bmatrix}_{i,j}$ and write this latter matrix as

$$P_{\sigma_1}^TA=\begin{array}{c}\begin{array}{cc}\scriptstyle 1 & \scriptstyle n-1\end{array}\\ \begin{bmatrix}a_{1,1} & A_{1,2}\\ A_{2,1} & A_{2,2}\end{bmatrix}\end{array}\begin{array}{l}\\ \scriptstyle 1\\ \scriptstyle n-1\end{array}. \tag{1.8}$$

Here $a_{1,1} \neq 0$ is the $(1,1)$-entry of $P_{\sigma_1}^T A$, and $A_{1,2}$ and $A_{2,1}$ are the rest of its first row and column respectively, whereas $A_{2,2}$ is the remaining $(n-1) \times (n-1)$ block. We then obtain the block LU factorization

$$\begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} u_{1,1} & U_{1,2} \\ \mathbb{0} & A_1 \end{bmatrix} \tag{1.9}$$

by setting

$$u_{1,1} = a_{1,1}, \quad L_{2,1} = a_{1,1}^{-1} A_{2,1}, \quad U_{1,2} = A_{1,2}, \quad A_1 = A_{2,2} - L_{2,1} U_{1,2}. \tag{1.10}$$

This latter $(n-1)\times(n-1)$ matrix A_1 is nonsingular, and is called the *Schur complement* of $P_{\sigma_1}^T A$. Proceeding similarly with it, we choose $i_2 \in \{2, \dots, n\}$ such that $a_{i_2,2} \neq 0$ and we denote by σ_2 the permutation of the set $\{2, \dots, n\}$ swapping 2 and i_2. Then we relabel the entries of the permuted matrix $P_{\sigma_2}^T A_1$ similarly as in (1.8) and applying the formulae in (1.10), we compute its $(1, n-2) \times (1, n-2)$ block LU factorization:

$$P_{\sigma_2}^T A_1 = L_2 U_2 \quad \text{with} \quad L_2 = \begin{bmatrix} 1 & \mathbb{0} \\ L_{3,2} & \mathbb{1}_{n-2} \end{bmatrix} \quad \text{and} \quad U_2 = \begin{bmatrix} u_{2,2} & U_{2,3} \\ \mathbb{0} & A_2 \end{bmatrix}. \tag{1.11}$$

It follows from (1.8), (1.9) and (1.11) that

$$\begin{aligned} A &= P_{\sigma_1} \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} u_{1,1} & U_{1,2} \\ \mathbb{0} & P_{\sigma_2} L_2 U_2 \end{bmatrix} \\ &= P_{\sigma_1} \begin{bmatrix} 1 & \mathbb{0} \\ \mathbb{0} & P_{\sigma_2} \end{bmatrix} \begin{bmatrix} 1 & \mathbb{0} \\ P_{\sigma_2}^T L_{2,1} & L_2 \end{bmatrix} \begin{bmatrix} u_{1,1} & U_{1,2} \\ \mathbb{0} & U_2 \end{bmatrix}. \end{aligned} \tag{1.12}$$

Iterating this procedure, we choose for $j = 3, \dots, n-1$ a permutation σ_j of the set $\{j, \dots, n\}$ swapping j with an index i_j such that $a_{i_j,j} \neq 0$, relabel accordingly the entries of the permuted Schur complement $P_{\sigma_j}^T A_{j-1}$ and compute the corresponding $(1, n-j) \times (1, n-j)$ block LU factorization

$$P_{\sigma_j}^T A_{j-1} = L_j U_j,$$

which produces the next Schur complement A_j as an $(n-j) \times (n-j)$ block of U_j.

Finally, for each j consider the n-permutation matrix $P_j = \begin{bmatrix} \mathbb{1}_{j-1} & \mathbb{0} \\ \mathbb{0} & P_{\sigma_j} \end{bmatrix}$ and set

$$P = \prod_{j=1}^{n-1} P_j.$$

The upper triangular matrix U is given by the first rows of the U_j's, and the unit lower triangular matrix L is similarly obtained from the first columns of the L_j's permuted by σ_k for $k > j$ as in (1.12).

In *Gaussian elimination with partial pivoting (GEPP)* at the j-th step we choose an index i_j such that $|a_{i_j,j}|$ is maximal among $|a_{p,j}|$, $j \leq p \leq n$. By (1.10), this choice will imply that all the entries of L have absolute value bounded by 1.

We next we describe in *pseudocode notation* the obtained algorithm. For convenience, we write the entries of the successive Schur complements as

$$A_j = \left[a_{i,k}^{(j)}\right]_{j+1 \leq i,k \leq n} \quad \text{for } j = 0, \dots, n-1.$$

Note that at the j-th step of Algorithm 1.2.1, each entry $a_{k,j}^{(j-1)}$ is assigned to the corresponding entry of U and is never used again. Moreover, each entry $a_{i,j}^{(j-1)}$ is

Algorithm 1.2.1 (GEPP)

```
for $i = j, \dots, n-1$
    choose $i_j \in \{j, \dots, n\}$ such that $\left|a_{i_j,i}^{(j-1)}\right| = \max_{j \le p \le n} \left|a_{p,j}^{(j-1)}\right|$
    swap row $i_j$ and row $j$ of $A_j$ and $L$
    for $k = j+1, \dots, n$
        $u_{j,k} \leftarrow a_{j,k}^{(j-1)}$        (compute the $j$-th row of $U$)
    for $i = j+1, \dots, n$
        $l_{i,j} \leftarrow a_{i,j}^{(j-1)}/a_{j,j}^{(j-1)}$        (compute the $j$-th column of $L$)
    for $i, k = i+1, \dots, n$
        $a_{i,k}^{(j)} \leftarrow a_{i,k}^{(j-1)} - l_{i,j}\, u_{j,k}$        (compute the $j$-th Schur complement)
$P \leftarrow P_\sigma$ for the permutation $\sigma = \sigma_1 \circ \cdots \circ \sigma_{n-1}$ with $\sigma_j$ the swap between $j$ and $i_j$
```

used to compute the corresponding entry of L and is never used again, and the same happens with the entries that compute the j-th Schur complement. Hence at that iteration we can safely rewrite A with the nontrivial entries in the j-th column of L and the j-th row of U and with those of the j-th Schur complement, and so we need no extra space to store them. At the end of the algorithm, the matrix A would contain all the nontrivial entries of L and of U:

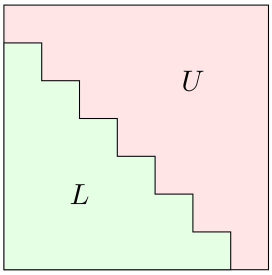

Figure 1.2.1. Storage distribution for GEPP

Moreover, the permutation matrix P can be coded by the corresponding element of $\mathcal{S}_n$, leading to a more efficient implementation.

Algorithm 1.2.2 (GEPP with storage management)

```
for $j = 1, \dots, n-1$
    choose $i_j \in \{j, \dots, n\}$ such that $|a_{i_j,i}| = \max_{j \le p \le n} |a_{p,j}|$
    swap row $i_j$ and row $j$ of $A$
    for $i = j+1, \dots, n$
        $a_{i.j} \leftarrow a_{i.j}/a_{j,j}$
    for $i, k = j+1, \dots, n$
        $a_{i,k} \leftarrow a_{i,k} - a_{i,j}\, a_{j,k}$
$\sigma \leftarrow \sigma_1 \circ \cdots \circ \sigma_{n-1}$ with $\sigma_j$ the swap between $j$ and $i_j$
```

As an illustration, we apply this algorithm to recompute the PLU factorization in Example 1.2.3.

Example 1.2.4. Set

$$A = \begin{bmatrix} 2 & -1 & 0 \\ 2 & -1 & 1 \\ -2 & 3 & -1 \end{bmatrix}.$$

For $j = 1$ we choose $i_1 = 1$, and so the rows of the matrix need not be permuted at this step. We compute the first column of L, the first row of U and the entries of the first Schur complement, and overwrite the matrix A with the obtained values: the used operations and updated matrix are

$$A = \begin{bmatrix} 2 & -1 & 0 \\ 2/2 & -1-1\cdot(-1) & 1-1\cdot 0 \\ -2/2 & 3-(-1)\cdot(-1) & -1-(-1)\cdot 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ 1 & 0 & 1 \\ -1 & 2 & -1 \end{bmatrix}.$$

For $j = 2$ we choose $i_2 = 3$ and swap the second and the third rows of A. We compute the second column of L, the second row of U and the entries of the second Schur complement, and overwrite the matrix A with the obtained values to get

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 1 & 0/2 & 1-0\cdot(-1) \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 1 & 0 & 1 \end{bmatrix}.$$

This matrix together with the chosen indexes $i_1 = 1$ and $i_2 = 3$ code the full PLU factorization of A:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Another variant of this algorithm is *Gaussian elimination with complete pivoting (GECP)*, that at the j-th step reorders all the rows and columns of the matrix so to choose a pivot whose absolute value is maximal among all the entries of the $(j-1)$-th Schur complement. It yields a factorization

$$A = P_1\, L\, U P_2$$

where both P_1 and P_2 are permutation matrices. This variant is slower than GEPP, but can be also more numerically stable.

1.3. Complexity and error analysis

1.3.1. Floating-point arithmetic. Given integers

$$\beta \geq 2, \quad p \geq 1, \quad L \leq U$$

we can consider the *floating-point system* $F(\beta, p, L, U)$ whose elements are the zero and the rational numbers of the form

$$f = \pm .d_1 d_2 \dots d_p \times \beta^e \tag{1.13}$$

with $0 < d_1 < \beta$, $0 \leq d_i < \beta$, $i = 2, \dots, p$ and $L \leq e \leq U$.

The parameter β is the *base* the floating-point system and it is typically 2 in machines and 10 in humans. The parameter p is the *precision*, L is the *underflow* and U the *overflow*. For the *floating-point number* f in (1.13) we say that the d_i's are the *digits*, the sequence $d_1 d_2 \dots d_p$ is the *mantissa*, and the integer e is the *exponent*.

Let's take a simple particular case:

$$\beta = 2, \quad p = 3, \quad L = -1, \quad U = 1. \tag{1.14}$$

The possible mantissas are

$$0.100, \quad 0.101, \quad 0.110, \quad 0.111,$$

that is, the rational numbers 1/2, 5/8, 3/4 and 7/8, that we are allowed to multiply by ±2, ±1 and ±1/2. Figure 1.3.1 gives the graphical representation of these floating-point numbers.

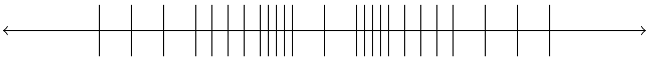

Figure 1.3.1. A simple floating-point system

Currently, the IEEE standards for floating-point arithmetic are the most used ones. They include two systems: single precision (32 bits) and double precision (64 bits). In the *IEEE single precision standard*, each floating-point number is coded as a string

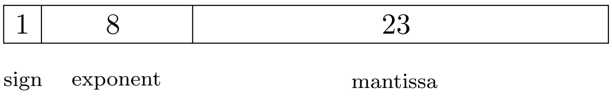

Figure 1.3.2. IEEE single precision encoding

The sign s is coded by 1 bit, the exponent e by 8 bits, and the mantissa q by the remaining 23 bits. The *coded number* is

$$f = (-1)^s (1+q)\, 2^{e-127}.$$

Since the base is 2, the first digit in the mantissa will always be 1, and so we do not need to code it. With respect to our notation, it corresponds to the situation when

$$\beta = 2, \quad p = 24, L = -126, \quad U = 127.$$

In the *IEEE double precision standard*, the sign s is coded by 1 bit, the exponent e by 11 bits, and the mantissa q by the remaining 52 bits:

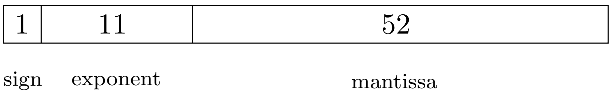

Figure 1.3.3. IEEE double precision encoding

The coded number is

$$f = (-1)^s (1+q)\, 2^{e-1023}.$$

With respect to our notation, it corresponds to the situation when

$$\beta = 2, \quad p = 53, \quad L = -1022, \quad U = 1026.$$

Each floating-point number $f \in F(\beta, p, L, U)$ verifies that

$$m \leq |f| \leq M \tag{1.15}$$

with $m = \beta^{L-1}$ and $M = (1 - \beta^{-p})\, \beta^U$, the *underflow threshold* and the *overflow threshold*, respectively.

Moreover, the floating-point line is plenty of holes, and so we need a rounding function to represent real numbers and operate with them. The models of floating-point arithmetic implemented in machines, and in particular those for the the IEEE standards, are quite complicated in their details. Here we are going to present a simple version that is sufficient for our purposes.

Our *rounding function* is the function

$$\mathrm{fl}\colon \mathbb{R} \longmapsto F(\beta, p, L, U) \cup \{\mathrm{Err}\} \tag{1.16}$$

such that for $\lambda \in \mathbb{R}$ the value $\mathrm{fl}(\lambda)$ is defined as 0 if $|\lambda| < m$, as Err if $|\lambda| > M$, and otherwise as the floating-point number that is closest to λ, choosing the largest one in case there are two of them.

The floating-point version of the arithmetic operations $+$,$-$, $\times$, $/$ are defined rounding their actual result. For instance

$$0.100 \times 2^1 + 0.110 \times 2^{-1} = 0.10000 \times 2^1 + 0.00110 \times 2^1 = 0.10110 \times 2^1$$

and so in our toy floating-point system (1.14) we have that

$$0.100 \times 2^1 \oplus 0.110 \times 2^{-1} = \mathrm{fl}(0.100 \times 2^1 + 0.110 \times 2^{-1}) = 0.101 \times 2^1.$$

The *machine epsilon* or *macheps*, is the quantity

$$\varepsilon = \frac{\beta^{1-p}}{2}. \tag{1.17}$$

It is the smallest positive real number such that $\mathrm{fl}(1+\varepsilon) > 1$. In other terms, it is the the maximum error that can occur when rounding to the unit value.

For $\lambda \in \mathbb{R}$ within the machine capacity, that is, whose absolute value lies within the underflow and overflow threshold, the *relative error* of its representation in the floating-point system $F(\beta, p, L, U)$ is bounded above by the machine epsilon:

$$\frac{|\lambda - \mathrm{fl}(\lambda)|}{|\lambda|} < \varepsilon, \tag{1.18}$$

the worst relative error occurring when λ approaches from below the number $1 + \varepsilon$. Hence the cost of each individual floating-point operation or *flop* is satisfactory: if $*$ is one of the arithmetic operations, then

$$\frac{|\lambda * \mu - \mathrm{fl}(\lambda * \mu)|}{|\lambda * \mu|} < \varepsilon \quad \text{if } m \leq |\lambda * \mu| \leq M.$$

The machine epsilon, and the underflow and overflow thresholds of the IEEE single precision standard are

$$\varepsilon = 2^{-24} \approx 6 \times 10^{-8}, \quad m = 2^{-127} \approx 6 \times 10^{-38}, \quad M = 2^{129}(1 - 2^{-24}) \approx 7 \times 10^{39},$$

whereas those for the double precision standard are

$$\varepsilon = 2^{-53} \approx 6 \times 10^{-16}, \quad m = 2^{-1022} \approx 2 \times 10^{308}, \quad M = 2^{1029} \times (1 - 2^{-24}) \approx 7 \times 10^{309}.$$

1.3.2. The complexity of GEPP.

All floating-point numbers occupy the same space in the machine memory, and so the cost of a computation depends on the following factors:

(1) the number of flops $\oplus$, $\ominus$, $\otimes$, $\oslash$ and of tests $x < y$,
(2) reading and writing in memory,
(3) use of memory space.

In numerical analysis, we mostly consider (1) and (3).

The use of memory space of an algorithm measured in terms of its *space complexity*, defined as the number of floating-point numbers simultaneously needed to perform computations. For instance, GEPP with storage management (Algorithm 1.2.2) is optimal in this respect, since it runs with n^2 floating-point numbers, that is

$$\epsilon_{\mathrm{GEPP}}(n) = n^2,$$

which is also the size of the input matrix A.

The *time complexity* of an algorithm is defined as the number of flops used through computations. This parameter can be computed from the pseudocode of the algorithm. To bound the expressions that appear, we recall from basic calculus the asymptotic formulae for power sums: for $k \geq 0$ we have that

$$\sum_{i=1}^{n} i^k = \int_0^n x^k\,dx + O(n^k) = \frac{n^{k+1}}{k+1} + O(n^k),$$

where $O(n^k)$ denotes the "big O" notation, indicating that there is a constant $c \geq 0$ such that

$$\left| \sum_{i=1}^{n} i^k - \frac{n^{k+1}}{k+1} \right| \leq c\, n^k.$$

The time complexity of the GEPP algorithm acting on $n \times n$ matrices can be computed and bounded as

$$\tau_{\mathrm{GEPP}}(n) = \sum_{i=1}^{n-1} \Bigg(\sum_{j=i+1}^{n} 1 + \sum_{j=i+1}^{n} \sum_{k=i+1}^{n} 2 \Bigg) = \sum_{i=1}^{n-1} ((n-i) + 2\,(n-i)^2) = \frac{2}{3}\,n^3 + O(n^2).$$

On the other hand, the algorithms for forward and backward substitution indicated in §1.2.1 have each a time complexity of $n^2 + O(n)$ flops. Hence GEPP solves the equation $A\,x = b$ with an overall time complexity of

$$\frac{2}{3}\,n^3 + O(n^2) \text{ flops.}$$

1.3.3. Vector and matrix norms. Norms are used to measure errors in matrix computations. A *vector norm* is a function

$$\|\cdot\| \colon \mathbb{F}^n \longrightarrow \mathbb{R}$$

such that for all $x, y \in \mathbb{F}^n$ and $\alpha \in \mathbb{F}$ we have that

(1) $\|x\| \geq 0$ *(positivity)*,
(2) $\|x\| = 0$ if and only if $x = 0$ *(definiteness)*,
(3) $\|\alpha\,x\| = |\alpha|\,\|x\|$ *(homogeneity)*,
(4) $\|x + y\| \leq \|x\| + \|y\|$ *(triangle inequality)*.

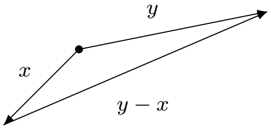

FIGURE 1.3.4. The triangle inequality

EXAMPLE 1.3.1. For $1 \leq p \leq +\infty$, the *p-norm* is the vector norm on $\mathbb{F}^n$ defined as

$$\|x\|_p = \begin{cases} \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} & \text{if } 1 \leq p < +\infty, \\ \max_i |x_i| & \text{if } p = +\infty. \end{cases}$$

Any two vector norms $\|\cdot\|_1$ and $\|\cdot\|_2$ can be compared: there are constants $c_1, c_2 > 0$ such that

$$\|x\|_1 \leq c_1 \|x\|_2 \quad \text{and} \quad \|x\|_2 \leq c_2 \|x\|_1 \quad \text{for all } x \in \mathbb{F}^n.$$

For instance, for all $x \in \mathbb{F}^n$ we have that

$$\|x\|_2 \leq \|x\|_1 \leq n^{1/2}\|x\|_2 \quad \text{and} \quad \|x\|_\infty \leq \|x\|_1 \leq n\,\|x\|_\infty.$$

A *matrix norm* is simply a vector norm on the space of $m \times n$ matrices $\mathbb{F}^{m\times n}$. Matrix norms on the spaces of $m \times n$, $n \times p$ and $m \times p$ matrices are said to be *compatible* is they are submultiplicative, that is, if for all $A \in \mathbb{F}^{m\times n}$ and $B \in \mathbb{F}^{n\times p}$ we have that

$$\|A\,B\| \leq \|A\|\,\|B\|. \tag{1.19}$$

EXAMPLE 1.3.2. The *Frobenius norm* of an $m \times n$ matrix A is defined as its Euclidean norm when considered as a vector of $\mathbb{F}^{n\times n}$, that is,

$$\|A\|_{\mathrm{F}} = \left(\sum_{i,j} |a_{i,j}|^2\right)^{1/2}.$$

Frobenius norms on $m \times n$, $n \times p$ and $m \times p$ matrices are compatible in the sense of (1.19).

Given norms on $\mathbb{F}^m$ on $\mathbb{F}^n$, the associated *operator norm* is the matrix norm defined as

$$\|A\| = \sup_{x\neq 0} \frac{\|A\,x\|}{\|x\|}.$$

If we consider a further norm on $\mathbb{F}^p$, the associated operator norms on $m \times n$, $n \times p$ and $m \times p$ matrices are also compatible in the sense of (1.19).

We list a number of basic properties of matrix norms. Let A be an $m \times n$ matrix with entries in $\mathbb{F}$. Then:

(1) $\|A\|_\infty = \max_i \sum_j |a_{i,j}|$ *(maximum row sum)*,

(2) $\|A\|_1 = \max_j \sum_i |a_{i,j}|$ *(maximum column sum)*,

(3) $\|A\|_2 = \lambda_{\max}(A^*A)^{1/2}$, where $A^* = \overline{A}^T$ denotes the conjugate transpose and $\lambda_{\max}$ the maximal eigenvalue.

An $n \times n$ matrix Q with real entries is *orthogonal* if

$$Q^T Q = Q\,Q^T = \mathbb{1}_n.$$

More generally, if Q has complex entries then it is *unitary* if

$$Q^* Q = Q\,Q^* = \mathbb{1}_n.$$

If Q' is another $m \times m$ matrix that is orthogonal ($\mathbb{F} = \mathbb{R}$) or unitary ($\mathbb{F} = \mathbb{C}$) then

$$\|Q'\,A\,Q\|_{\mathrm{F}} = \|A\|_{\mathrm{F}} \quad \text{and} \quad \|Q'\,A\,Q\|_2 = \|A\|_2.$$

In particular $\|Q\|_{\mathrm{F}} = n^{1/2}$, $\|Q'\|_{\mathrm{F}} = m^{1/2}$ and $\|Q\|_2 = \|Q'\|_2 = 1$ The proofs of these properties can be found in [**Dem97**, Lemma 1.7].

1.3.4. Perturbation theory in linear equation solving. In this section we study the *forward stability* of the linear equation $A\,x = b$, that is, how errors in the input data (A, b) propagate to the solution x. Such errors are typically produced by approximate measurements and prior computations, and by rounding. They are unavoidable, and so it is important to understand how they affect the quality of our computations.

Some matrices behave really bad with respect to perturbations.

Example 1.3.3. Set

$$A = \begin{bmatrix} 3.55 & 1.13 \\ 2.2 & 0.7 \end{bmatrix} \quad \text{and} \quad b = (3.55, 2.2).$$

The solution to the equation $A\,x = b$ is the vector $x = (1, 0)$. Taking the perturbation $\widehat{b} = b + \delta b$ with

$$\delta b = 0.00017,$$

the solution to the equation $A\,x = \widehat{b}$ is $\widehat{x} = (1.9775, -0.3111)$, far away from the solution to the original problem.

Let $(\widehat{A}, \widehat{b})$ be the perturbed data and $\widehat{x}$ the corresponding solution, so that

$$\widehat{A}\,\widehat{x} = \widehat{b}.$$

To compare the *errors*

$$\delta A = A - \widehat{A}, \quad \delta b = b - \widehat{b}, \quad \delta x = x - \widehat{x}$$

we consider a norm $\|\cdot\|$ on $\mathbb{F}^n$ and its associated operator norm on $\mathbb{F}^{n\times n}$.

All these matrices and vectors are coded by floating-point numbers, and a standard rule of thumb says that their norm gives the most significant exponent of these floating-point numbers, whereas the relative error gives the precision of the approximation. We can express this in a slightly more concrete way as the fact that $\log_\beta \|x\|$ and $\log_\beta \|\widehat{x}\|$ approximate the largest exponent of x and of $\widehat{x}$ respectively, and that

$$-\log_\beta \Big(\frac{\|\delta x\|}{\|x\|}\Big) \approx \text{ number of correct digits of } \widehat{x}, \tag{1.20}$$

and similarly for A and b. These are certainly not mathematical statements, but nevertheless can be followed as guiding principles in practice.

Since we are interested in the precision of our computations, we translate this interest to the study of the propagation of relative errors, that is, how we can control the ratio

$$\frac{\|\delta x\|}{\|x\|} \tag{1.21}$$

in terms of $\|\delta A\|/\|A\|$ and $\|\delta b\|/\|b\|$.

The behavior of the solution the equation $A\,x = b$ with respect to perturbations of the input data is quantified by the notion of condition number. The *condition number* of A with respect to the vector norm $\|\cdot\|$ is defined as

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

and also noted as $\kappa(A)$ when the norm is clear from the context. It is a real number greater or equal to 1, because

$$\|A^{-1}\|\,\|A\| \geq \|A^{-1}A\| = \|\mathbb{1}_n\| = 1$$

by the submultiplicativity of the operator norm.

To bound the relative error in the solution to (1.21), we consider the difference

$$\begin{array}{rrl} & (A+\delta A)\,(x+\delta x) & = b-\delta b \\ - & A\,x & = \quad b \\ \hline & \delta\,A\,x+(A+\delta\,A)\,\delta x = & \quad \delta b \end{array}$$

which readily implies that

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

Taking norms we obtain $\|\delta x\| \le \|A^{-1}\|\,(\|\delta A\|\,(\|x\|+\|\delta\,x\|)+\|\delta b\|)$, that can be rearranged to

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

because $\|b\| \le \|A\|\,\|x\|$. This readily implies the following upper bound for the relative error of x in terms of those of A and b:

$$\frac{\|\delta x\|}{\|x\|} \le \frac{\kappa(A)}{1-\kappa(A)\,\frac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right). \tag{1.22}$$

The multiplier in the right-hand side of (1.22) is close to the condition number when the relative error of the matrix A is small, which is the case of interest. Disregarding the denominator in this multiplier, we can write this inequality as

$$-\log_\beta\left(\frac{\|\delta x\|}{\|x\|}\right) \ge -\log_\beta\left(\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right)-\log_\beta \kappa(A)$$

Hence following the rule of thumb (1.20), when solving the linear equation $A\,x=b$ we expect a loss of precision of $\approx \log_\beta \kappa(A)$ digits.

In linear equation solving, the matrix A is said to be *well/ill conditioned* if its condition number is small/large when compared to its norm. When A is ill conditioned, small relative changes in the data (A,b) might produce large changes in the solution x, as it was the case in Example 1.3.3.

Example 1.3.4. With notation as in Example 1.3.3, the relative errors with respect to the ∞-norm are, up to 5 decimal digits, are

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty}=0,\qquad \frac{\|\delta b\|_\infty}{\|b\|_\infty}=\frac{0.00017}{3.55}=0.00005,\qquad \frac{\|\delta x\|_\infty}{\|x\|_\infty}=\frac{0.9775}{1}=0.9775. \tag{1.23}$$

The relative error has been propagated from the input data to the solution almost by a factor 20,000! In terms of floating-point numbers, we have that

$$b=(0.355\times 10^1, 0.22\times 10^1),\quad \widehat{b}=(0.35498\times 10^1, 0.220017\times 10^1),$$
$$x=(0.1\times 10^1, 0),\quad \widehat{x}=(0.19775\times 10^1, -0.3111\times 10^0).$$

Hence $\widehat{b}$ has 4 correct digits, whereas $\widehat{x}$ has none: all the precision has been lost.

Indeed, the inverse is $A^{-1}=\begin{bmatrix}-3550 & 2200\\ 1130 & -700\end{bmatrix}$ and so the condition number of A is

$$\kappa(A)=\|A^{-1}\|\,\|A\|=5750\times 4.68=26910.$$

The upper bound (1.22) writes down in this case as

$$0.9775=\frac{\|\delta x\|_\infty}{\|x\|_\infty}\le \kappa(A)\,\frac{\|\delta b\|_\infty}{\|b\|_\infty}=26910\times 0.00005=1.3455,$$

reflecting the behavior in (1.23). Moreover,

$$\log_{10} \kappa(A) = 3.75966,$$

and so rule in (1.20) is valid in this case.

Ill-conditioned matrices matrices destroy the quality of your approximations. Recall that rounding with IEEE single precision and double precision give approximations with 24 and 53 correct bits, respectively. Hence if you have exact data truncated with IEEE single or double precision, the computed solution (with *any* algorithm!) of $A\,x = b$ will be meaningless as as soon

$$\kappa(A) > 2^{24} \approx 6 \cdot 10^8 \text{ (single precision)} \quad \text{and} \quad \kappa(A) > 2^{53} \approx 10^{16} \text{ (double precision)}.$$

For the 2-norm, the condition number has a beautiful geometric interpretation as the inverse of the distance of the matrix to the set of singular matrices:

$$\min\left\{ \frac{\|\delta A\|_2}{\|A\|_2} \;\middle|\; A + \delta A \text{ is singular} \right\} = \frac{1}{\kappa_2(A)},$$

see for instance [**Dem97**, Theorem 2.1]. This reciprocal relation between the condition number of a well-posed problem and its distance to the set of ill-posed ones, is an instance of a general principle in numerical analysis.

1.3.5. Error analysis in GEPP. The GEPP algorithm takes as input the pair (A, b) and proceeds by computing a factorization

$$A = P\,L\,U$$

with P a permutation, L unit lower triangular, and U upper triangular. Once this is achieved, the equation

$$A\,x = b$$

is solved permuting the entries of b and applying forward and backward substitution. In an actual implementation, computations are performed numerically, that is, using floating-point arithmetic, thus producing an approximate solution

$$x_{\mathrm{GEPP}}.$$

To control the error in this computation, we apply the two steps:

(1) analyze roundoff errors to show the existence of a matrix A_{GEPP} such that $A_{\mathrm{GEPP}}\, x_{\mathrm{GEPP}} = b$ having a small relative error with respect to A *(backward analysis)*,

(2) apply perturbation theory to bound the relative error of x_{GEPP} with respect to x in terms of the condition number of A and the precision of our floating-point system.

We next study this strategy for the ∞-norm, although we might equally well consider any other standard norm like the 1-norm or the 2-norm. Rounding A gives a matrix

$$\widehat{A} = A + \delta A$$

whose entries approximate in relative error those of A, up to the machine epsilon ε, as explained in (1.18). Similarly rounding b gives a vector $\widehat{b}$ whose coefficients approximate those of b with a relative error bounded by ε too. Then

$$\|\delta A\|_\infty = \max_i \sum_{j=1}^n |\delta a_{i,j}| \leq \max_i \sum_{j=1}^n \varepsilon\,|a_{i,j}| \leq \varepsilon\, \max_i \sum_{j=1}^n |a_{i,j}| = \varepsilon\,\|A\|_\infty,$$
$$\|\delta b\|_\infty = \max_i |\delta b_i| \leq \max_i \varepsilon\,|b_i| \leq \varepsilon\, \max_i |b_i| = \varepsilon\,\|b\|_\infty,$$

and so

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty}, \frac{\|\delta b\|_\infty}{\|b\|_\infty} < \varepsilon.$$

By the perturbation theorem (1.22), this error will be amplified to

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \leq \frac{\kappa(A)}{1-\varepsilon\,\kappa(A)}\, 2\,\varepsilon.$$

Hence to keep the quality of this bound, we need to show that $A_{\text{GEPP}} = A + \delta A$ with

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \leq c\,\varepsilon$$

for a small constant $c > 0$. To this end, we must be careful about *pivoting.*

Example 1.3.5. Consider the matrix

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

with η a power of the base β that is smaller than ε. Its LU factorization (without pivoting) is $A = L\,U$ with

$$L = \begin{bmatrix} 1 & 0 \\ \eta^{-1} & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} \eta & 1 \\ 0 & 1-\eta^{-1} \end{bmatrix}.$$

We have that

$$A^{-1} = \begin{bmatrix} \frac{1}{\eta-1} & \frac{-1}{\eta-1} \\ \frac{-1}{\eta-1} & \frac{\eta}{\eta-1} \end{bmatrix}, \quad L^{-1} = \begin{bmatrix} 1 & 0 \\ -\eta^{-1} & 1 \end{bmatrix}, \quad U^{-1} = \begin{bmatrix} \eta^{-1} & \frac{1}{1-\eta} \\ 0 & \frac{\eta}{\eta-1} \end{bmatrix}$$

and so

$$\kappa(A) \approx 4, \quad \kappa(L) \approx \eta^{-2}, \quad \kappa(U) \approx \eta^{-2}.$$

The disparity between the condition number of A and those of L and U indicates that this procedure is be numerically unstable. We next verify that this is indeed the case.

Set $b = (1, 2)$, so that the solution to the equation $A\,x = b$ is

$$(x_1, x_2) = \Big(\frac{1}{1-\eta}, \frac{1-2\,\eta}{1-\eta}\Big) \approx (1, 1).$$

On the other hand, by the definition of the machine epsilon we have that

$$1 \ominus \eta^{-1} = \eta^{-1}(1 \ominus \eta).$$

Hence in floating-point arithmetic, the computed factors of A are

$$L_{\text{GEWP}} = \begin{bmatrix} 1 & 0 \\ \eta^{-1} & 1 \end{bmatrix} \quad \text{and} \quad U_{\text{GEWP}} = \begin{bmatrix} \eta & 1 \\ 0 & 1 \ominus \eta^{-1} \end{bmatrix} = \begin{bmatrix} \eta & 1 \\ 0 & -\eta^{-1} \end{bmatrix}.$$

Solving $L_{\text{GEWP}}\, y = b$ gives

$$y_1 = 1 \quad \text{and} \quad y_2 = 2 \ominus \eta^{-1} = -\eta^{-1}.$$

Then $U_{\mathrm{GEWP}}\, x = y$ gives

$$x_2 = \frac{-\eta^{-1}}{-\eta^{-1}} = 1 \quad \text{and} \quad x_1 = \frac{1 \ominus 1}{\eta} = 0.$$

We obtain $x_{\mathrm{GEWP}} = (1, 0)$, which is *not* close to the actual solution: the relative error

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \approx 1$$

cannot be bounded by $c\,\varepsilon$ for a small constant $c > 0$, and so GEWP is not backward stable.

In this example, pivoting eliminates the unstability just illustrated. Indeed, GEPP gives the factorization is $A = P\,L\,U$ with

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 \\ 0 & 1 - \eta \end{bmatrix}. \tag{1.24}$$

We have that $\kappa(P) = 1$, $\kappa(L) \approx 1$ and $\kappa(U) \approx 4$, and so all these factors are well-conditioned. Moreover,

$$P_{\mathrm{GEPP}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad L_{\mathrm{GEPP}} = \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix}, \quad U_{\mathrm{GEPP}} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \ominus \eta \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

and successively solving $P_{\mathrm{GEPP}}\, z = b$, $L_{GEPP} y = z$ and $U_{\mathrm{GEPP}} x = y$ gives the accurate solution $x_{\mathrm{GEPP}} = (1, 1)$.

1.3.6. Backward error analysis of GEPP. To see that GEPP is a numerically stable algorithm we need to show that the computed solution x_{GEPP} satisfies $A_{\mathrm{GEPP}}\, x_{\mathrm{GEPP}} = b$ with $A_{\mathrm{GEPP}} = A + \delta A$ such that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \le c\,\varepsilon$$

for a small constant $c > 0$ *(backward stability)*.

As shown in Example 1.3.5, crucial information might be lost when intermediate quantities of disparate size are added together. This is an actual risk for us, because GEPP does perform a lot of additions and subtractions and, as a matter of fact, GEPP is *not* backward stable. Still we can give an interesting upper bound for the relative error it produces. In the sequel we will explain this bound and give some remarks of practical interest.

We denote by $|A|$ the $n \times n$ matrix whose entries are the absolute values of those of A, that is

$$|A| = \big[|a_{i,j}|\big]_{i,j}\,.$$

Analyzing the roundoff errors of the involved computations, one can find an $n \times n$ matrix A_{GEPP} with $A_{\mathrm{GEPP}}\, x = b$ such that setting $\delta A = A - A_{\mathrm{GEPP}}$ we have that

$$|\delta A| \le (3\,n\,\varepsilon + n^2\,\varepsilon^2)\,|L_{\mathrm{GEPP}}|\,|U_{\mathrm{GEPP}}|.$$

see [**Dem97**, pages 47-49] for the details. Taking norms, this readily implies that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty} \le (3\,n\,\varepsilon + n^2\,\varepsilon^2)\,\frac{\|L_{\mathrm{GEPP}}\|_\infty\,\|U_{\mathrm{GEPP}}\|_\infty}{\|A\|_\infty}. \tag{1.25}$$

The practice of numerical linear solving is that GEPP *almost* always keeps

$$\|L_{\mathrm{GEPP}}\|_\infty\,\|U_{\mathrm{GEPP}}\|_\infty \approx \|A\|_\infty,$$

as in (1.24). If this were the case, then we would have

$$\frac{\|\delta_{\mathrm{GEPP}}A\|}{\|A\|}\lesssim n\,\varepsilon.$$

Thus we say that GEPP is backward stable *in practice.*

For a theoretical upper bound we can consider the *pivot growth factor*

$$g_{\mathrm{GEPP}}=\frac{\max_{i,j}|u_{i,j}|}{\max_{i,j}|a_{i,j}|}.$$

GEPP guarantees that the entries of L_{GEPP} are bounded by 1 in absolute value and so

$$\|L\|_\infty\leq n\quad\text{and}\quad\|U\|_\infty\leq n\,g_{\mathrm{GEPP}}\,\|A\|_\infty.$$

Together with (1.25) this implies that

$$\frac{\|\delta A\|_\infty}{\|A\|_\infty}\leq 3\,n^3\varepsilon\,g_{\mathrm{GEPP}}. \tag{1.26}$$

Backward stability amounts to the fact that g_{GEPP} is small or grows slowly as a function of n. In general, we have that

$$g_{\mathrm{GEPP}}\leq 2^{n-1},$$

and unfortunately, there are rare cases where this exponential bound is attained: for instance, when $n=4$ we have that

$$\begin{bmatrix}1&0&0&1\\-1&1&0&1\\-1&-1&1&1\\-1&-1&-1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\-1&1&0&0\\-1&-1&1&0\\-1&-1&-1&1\end{bmatrix}\begin{bmatrix}1&0&0&1\\0&1&0&2\\0&0&1&4\\0&0&0&8\end{bmatrix},$$

and this example can be easily generalized to an $n\times n$ matrix with pivot growth 2^{n-1}

1.4. Special linear systems

As a general principle, we want to take advantage of any special structure that may be present in the problem under consideration, to increase the efficiency of our computations. This increase of efficiency might be translated into faster computations using less memory space, and/or more reliable results. Next we will study matrices exhibiting properties like *symmetry*, *definiteness* and *bandedness.*

1.4.1. Symmetric matrices.

An $n\times n$ matrix A is *symmetric* if

$$A^T=A.$$

Since $a_{j,i}=a_{i,j}$ for all i,j, to represent A we only need to specify the $\frac{n\,(n+1)}{2}$ entries $a_{i,j}$ for $i\geq j$, instead of the usual n^2 ones, and so to store a symmetric matrix we need approximately half the space that for general one. Consequently, in this case we would like to solve the linear equation

$$A\,x=b$$

with half the complexity of the general case, that is, using (approximately) $\frac{n^2}{2}$ memory slots instead of n^2, and $\frac{n^3}{3}$ flops instead of $\frac{2\,n^3}{3}$.

Permuting rows destroys symmetry, as can be seen already when $n=2$:

$$\begin{bmatrix}0&1\\1&0\end{bmatrix}\begin{bmatrix}a&b\\b&c\end{bmatrix}=\begin{bmatrix}b&c\\a&b\end{bmatrix}.$$

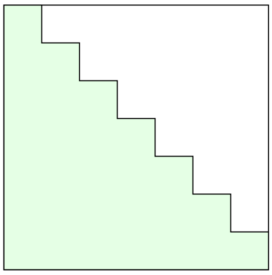

Figure 1.4.1. Storage of a symmetric matrix

Hence to preserve symmetry, we will avoid using any pivoting strategy, and consequently we will only consider its LU factorization, whenever it exists.

From now on, suppose that A is a symmetric nonsingular $n \times n$ matrix that has a factorization

$$A = L\,U \tag{1.27}$$

with L unit lower triangular and U upper triangular. When this occurs, these factors are connected: for instance, when $n = 2$ we have that

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & b \\ 0 & c - \frac{b^2}{a} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & c - \frac{b^2}{a} \end{bmatrix} \begin{bmatrix} 1 & \frac{b}{a} \\ 0 & 1 \end{bmatrix},$$

and so U is a row scaling of L. This is a general fact: in the factorization (1.27) we have that

$$U = D\,L^T$$

with $D = \operatorname{diag}(u_{1,1}, \dots, u_{n,n})$, and so this factorization is equivalent to

$$A = L\,D\,L^T$$

with L unit lower triangular and D diagonal.

This LDLT factorization also allows to solve the linear equation $A\,x = b$ following the steps:

(1) solve $L\,z = b$ by forward substitution,
(2) solve $D\,y = z$ scaling each entry,
(3) solve $L^T\,x = y$ by backward substitution.

To compute it, we proceed similarly as we did before for the PLU factorization: consider the 2×2 block decomposition

$$A = \begin{array}{c} \begin{array}{cc} {\scriptstyle 1} & {\scriptstyle n-1} \end{array} \\ \begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \end{array} \begin{array}{l} {\scriptstyle 1} \\ {\scriptstyle n-1} \end{array}$$

where $a_{1,1}$ is the $(1,1)$-entry of A, $A_{1,2}$ and $A_{2,1}$ are the rest of its first row and columns respectively, and $A_{2,2}$ is the remaining $(n-1) \times (n-1)$ block. Since we assume that A admits an LU factorization we have that $a_{1,1} \neq 0$, and since it is symmetric we also have that $A_{1,2} = A_{2,1}^T$. Then we can compute the the block LDLT factorization

$$\begin{bmatrix} a_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = \begin{bmatrix} 1 & \mathbb{0} \\ L_{2,1} & \mathbb{1}_{n-1} \end{bmatrix} \begin{bmatrix} d_1 & \mathbb{0} \\ \mathbb{0} & A_1 \end{bmatrix} \begin{bmatrix} 1 & L_{2,1}^T \\ \mathbb{0} & \mathbb{1}_{n-1} \end{bmatrix}$$

by setting

$$d_1 = a_{1,1}, \quad L_{2,1} = a_{1,1}^{-1} A_{2,1}, \quad D_1 = A_{2,2} - A_{2,1} L_{2,1}^T. \tag{1.28}$$

The Schur complement A_1 is a symmetric nonsingular $(n-1)\times(n-1)$ matrix and so we can repeat the procedure on it, leading to a symmetric version of Algorithm 1.2.1.

For convenience, we write the Schur complements defined by the successive application of the formulae in (1.28) as

$$A_j = \big[a_{i,k}^{(j)}\big]_{j+1\le i,k\le n} \qquad \text{for } j = 0, \dots, n-1.$$

Algorithm 1.4.1 (LDLT algorithm)

```
for $j = 1, \dots, n$
    $d_j \leftarrow a_{j,j}^{(j-1)}$        (compute the $j$-th diagonal entry of $D$)
    for $i = j+1, \dots, n$
        $l_{i,j} \leftarrow a_{i,j}^{(j-1)} / a_{j,j}^{(j-1)}$        (compute the $j$-th column of $L$)
    for $i, k = j+1, \dots, n$
    $a_{i,k}^{(j)} \leftarrow a_{i,j}^{(j-1)} - a_{i,j}^{(j-1)}\, l_{k,j}$        (compute the $j$-th Schur complement)
$d_n \leftarrow a_{n,n}^{(n-1)}$        (compute the $n$-th diagonal entry of $D$)
```

As it stands, this algorithm is inefficient both from the point of view of memory usage and speed of execution. To fully profit from the symmetry of the input matrix A, first recall that it can be represented keeping only its (i,j)-th entries for $i \ge j$. At the j-th step of Algorithm 1.4.1, the (j,j)-entry of the $(j-1)$-th Schur complement A_{j-1} is assigned to the j-th diagonal entry of D and never used again. On the other hand, its (i,j)-th entry is used both to compute the (i,j)-th entry of L and the i-th row of the j-th Schur complement. Moreover, the (i,k)-entries of A_{j-1} are only used to compute this next Schur complement. Moreover, since it is symmetric, we only need to compute its lower triangular part.

Hence we can safely rewrite the lower triangular part of A with the j-th diagonal entry of D, the nontrivial entries in the j-th column of L, and the lower triangular part of the j-th Schur complement. Hence we need no extra space to store them apart from an auxiliary $(n-1)$-th vector c to keep the values of the j-th column of A. At the end of the procedure, the lower triangular part of A would contain all the diagonal entries of D and the nontrivial entries of L.

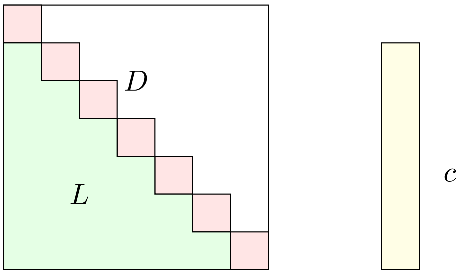

FIGURE 1.4.2. Storage distribution for LDLT

Algorithm 1.4.2 (LDLT algorithm with storage management)

```
for $j = 1, \dots, n$
    for $i = j+1, \dots, n$
        $c_i \leftarrow a_{i,j}$
        $a_{i,j} \leftarrow a_{i,j}/a_{i,j}$
    for $k = j+1, \dots, n$ and $i = k, \dots, n$
    $a_{i,k} \leftarrow a_{i,j} - c_i\, a_{k,j}$
```

As shown in Figure 1.4.2, this algorithm uses an (essentially) optimal amount of space, namely

$$\frac{n\,(n+1)}{2} + n - 1 = \frac{n^2}{2} \text{ memory slots.}$$

On the other hand, its time complexity is

$$\sum_{j=1}^{n-1}\Bigg(\sum_{i=j+1}^{n} 1 + \sum_{j+1 \le k \le i \le n} 2\Bigg) = \sum_{j=1}^{n-1}((n-j) + (n-j)\,(n-j+1))$$
$$= \sum_{j=1}^{n-1}((n-j)^2 + O(n-j)) = \frac{n^3}{3} + O(n^2),$$

which is asymptotically half the time complexity of the PLU factorization.

Example 1.4.1. Consider the 3 × 3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}.$$

Applying Algorithm 1.4.2, for $j = 1$ we have that

$$c = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1/1 & 5-(-1)\cdot(-1) & \\ 2/1 & 2-2\cdot(-1) & 17-2\cdot 2 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4 & 13 \end{bmatrix}.$$

For $j = 2$ we have that

$$c = \begin{bmatrix} * \\ 4 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 4/4 & 13-4\cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 4 & \\ 2 & 1 & 9 \end{bmatrix},$$

from where we extract the matrices in the LDLT factorization:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{bmatrix}.$$

In general, the LDLT of a symmetric matrix can be numerically unstable, as shown already by the matrix

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

for $0 < \eta$ smaller than the machine epsilon (Example 1.3.5). Still this factorization can be useful in practice, whenever the successive pivots are not too small.

1.4.2. Symmetric positive definite systems. Let A be a symmetric $n \times n$ matrix with real coefficients. We say that A is a *positive-definite* if for all $x \in \mathbb{R}^n \setminus \{0\}$ we have that

$$x^T A\, x > 0.$$

Positive definite symmetric (PDS) matrices play an important role in convex optimization. For instance, given a function of several real variables that is twice differentiable, if its Hessian matrix (the matrix of its second partial derivatives) is positive-definite then the function is convex in a neighborhood of that point.

PDS matrices can be characterized in several ways. First of all, there is an important result from linear algebra saying that a real $n \times n$ matrix A is symmetric if and only if it is orthogonally similar to a diagonal matrix, that is

$$A = Q^T \Lambda\, Q \tag{1.29}$$

with Q orthogonal and Λ diagonal, and then A is positive-definite if and only if the diagonal entries of Λ are positive.

In the factorization (1.29), we have that $Q^{-1} = Q^T$ and that the diagonal entries of Λ coincide with the eigenvalues of the given symmetric matrix. Hence PDS matrices can also be characterized as the symmetric real matrices whose eigenvalues are positive.

A third characterization of PDS matrices is given by the existence of a *Cholesky factorization*, a variant of the LU factorization with a positivity property. Namely, a real $n \times n$ matrix A is PDS if and only if there is a lower triangular $n \times n$ matrix G with positive diagonal entries such that

$$A = G\, G^T. \tag{1.30}$$

When it exists, this lower triangular matrix is unique [**Dem97**, Proposition 2.2].

This factorization is closely related to the LDLT factorization of a symmetric matrix studied in §1.4.1. Indeed, consider the diagonal matrix $\Lambda = \operatorname{diag}(g_{1,1}, \dots, g_{n,n})$ and set

$$L = G\, \Lambda^{-1} \quad \text{and} \quad D = \Lambda^2.$$

Then L is unit lower triangular and D is diagonal, and the Cholesky factorization (1.30) translates into the LDLT factorization

$$A = L\, D\, L^T.$$

Indeed, one way of computing the Cholesky factorization a PDS matrix proceeds by computing its LDLT factorization with Algorithm 1.2.2. The diagonal entries of D are positive and we might then obtain the lower triangular matrix G in (1.30) by setting

$$G = L \operatorname{diag}(d_{1,1}^{1/2}, \dots, d_{n,n}^{1/2}).$$

As a matter of fact, it will be easier to proceed in a different way. Given the PDS $n \times n$ matrix A, the matrix G is the solution of a system of $\frac{n\,(n+1)}{2}$ equations (one per each entry of A in its lower triangular part) in $\frac{n\,(n+1)}{2}$ variables (the nontrivial entries of G).

These equations are nonlinear but nevertheless, they can be easily solved when appropriately ordered: for each $i \geq j$ we have that $a_{i,j} = \sum_{k=0}^{j} g_{j,k}\, g_{i,k}$, or equivalently

$$g_{j,j}\, g_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} g_{j,k}\, g_{i,k}.$$

If the first $j-1$ columns of G are already known, we can use this equation to compute the diagonal entry $g_{j,j}$ and then the rest of the entries in this column. Similarly as with GEPP (and to a great extent with the LDLT factorization) each (i,j)-th entry of A is used only to compute the corresponding entry of G, and so we can safely overwrite it. At the end of the procedure, the matrix A would contain all the nontrivial entries of G in its lower triangular part.

Algorithm 1.4.3 (Cholesky algorithm))

```
for $j = 1, \dots, n$
    $a_{j,j} \leftarrow (a_{j,j} - \sum_{k=1}^{j-1} a_{j,k}^2)^{1/2}$
    for $i = j+1, \dots, n$
        $a_{i,j} \leftarrow (a_{i,j} - \sum_{k=1}^{j-1} a_{i,k}\, a_{j,k})/a_{j,j}$
```

Thanks to the characterization of PDS matrices in (1.30), this algorithm might be applied to an arbitrary real symmetric A: if this matrix is PDS then we would obtain its Cholesky factorization as explained and if it is not, then the computation algorithm would break down because of a forbidden operation, like taking the square root of a negative number or dividing by zero. Indeed, this is the cheapest way to test if a given real symmetric matrix is definite-positive or not.

Example 1.4.2. Consider again the 3×3 matrix

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -1 & 5 & 2 \\ 2 & 2 & 17 \end{bmatrix}$$

as in Example 1.4.1. Applying the Cholesky algorithm, for $j = 1$ we obtain

$$A = \begin{bmatrix} 1^{1/2} & & \\ -1/1 & 5 & \\ 2/1 & 2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 5 & \\ 2 & 2 & 17 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 1 & & \\ -1 & (5-(-1)\cdot(-1))^{1/2} & \\ 2 & (2-(-1)\cdot 2)/2 & 17 \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 17 \end{bmatrix}$$

Finally, for $j = 3$ we obtain

$$A = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & (17-(2\cdot 2+2\cdot 2))^{1/2} \end{bmatrix} = \begin{bmatrix} 1 & & \\ -1 & 2 & \\ 2 & 2 & 3 \end{bmatrix}.$$

Hence A is positive-definite, and the matrix in its Cholesky factorization is

$$G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 2 & 3 \end{bmatrix}.$$

Algorithm 1.4.3 uses an optimal amount of space, namely

$$\frac{n\,(n+1)}{2} \text{ memory slots,}$$

exactly the space needed to represent A.

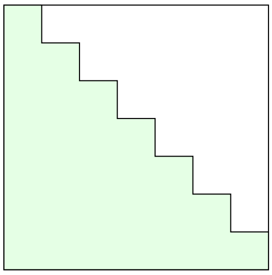

FIGURE 1.4.3. Storage distribution for the Cholesky algorithm

Its time complexity is

$$\begin{aligned}\sum_{j=1}^{n}\Big(2\,j-1+\sum_{i=j+1}^{n}(2\,j-1)\Big) &= \sum_{j=1}^{n}(n-j+1)\,(2\,j-1)\\ &= \sum_{j=1}^{n}2\,n\,j-\sum_{j=1}^{n}2\,j^2+O(n^2)\\ &=(n^3+O(n^2))-\Big(\frac{2}{3}\,n^3+O(n^2)\Big)+O(n^2)\\ &=\frac{1}{3}\,n^3+O(n^2),\end{aligned}$$

which is also asymptotically half the time complexity of the PLU factorization.

Furthermore, the Cholesky algorithm is also backward stable. Indeed, let A be a PDS matrix and b a real n-vector, and let x_{Ch} be the computed solution of the equation $A\,x=b$ using the Cholesky factorization, combined with forward and backward substitution for solving the equations $G\,y=b$ and $G^T x=y$ respectively. The same analysis of GEPP in § 1.3.6 shows that this solution satisfies

$$(A+\delta A)\,x_{\mathrm{Ch}}=b$$

with

$$|\delta A|\le 3\,n\,\varepsilon\,|G|\,|G^T| \tag{1.31}$$

with ε the machine epsilon and where the bars indicate the matrices whose entries are the absolute values of those in the indicated ones, and where the inequality occurs at every entry.

By the Cauchy-Schwartz inequality, for each i,j we have that

$$(|G|\,|G^T|)_{i,j}\le\sum_{k=1}^{n}|g_{i,k}|\,|g_{j,k}|\le\Big(\sum_{k=1}^{n}g_{i,k}^2\Big)^{1/2}\Big(\sum_{k=1}^{n}g_{j,k}^2\Big)^{1/2}=a_{i,i}^{1/2}a_{j,j}^{1/2}\le\max_{i,j}|a_{i,j}|.$$

Hence $\||G|\,|G^T|\|_\infty\le n\,\|A\|_\infty$, and so we deduce from (1.31) the upper bound for the relative backward error

$$\frac{\|\delta\,A\|_\infty}{\|A\|_\infty}\le 3\,n^2\,\varepsilon.$$

Properly done, all the elements in this section can be extended to the complex case. An $n\times n$ matrix A with complex entries is *Hermitian* if it coincides with its conjugate transpose that is, if

$$A^*=A.$$

A Hermitian matrix is *positive-definite* if for all $x \in \mathbb{C}^n \setminus \{0\}$ we have that

$$x^* A\, x > 0.$$

Hermitian matrices enjoy similar properties as those of symmetric real matrices: for instance, a complex $n \times n$ matrix A is Hermitian if and only if

$$A = Q^* \Lambda\, Q$$

with Q a unitary matrix and Λ a diagonal matrix with real entries. When this is the case, the Hermitian matrix A is positive-definite if and only if the diagonal entries of Λ are positive.

The Cholesky factorization also extends to this setting: a complex $n \times n$ matrix A is positive-definite Hermitian if and only if it can be factored as

$$A = G\, G^*$$

with G lower triangular con positive diagonal entries, and Algorithm 1.4.3 can be easily modified to compute this matrix.

1.4.3. Band matrices. Roughly speaking, a *band matrix* is a matrix whose entries are concentrated around the diagonal. Such matrices arise from systems of (scalar) linear equations that can be ordered in such a way that each i-th variable only appears in a neighborhood of the i-th equation, like those appearing when discretizing ordinary differential equations (ODE's).

More precisely, the *lower bandwidth* and the *upper bandwidth* of an $n \times n$ matrix A are respectively defined as the smallest integers b_L and b_U such that

$$a_{i,j} = 0$$

whenever $i > j - b_L$ or $i < j + b_U$. For instance, a nonsingular lower (respectively upper) triangular matrix has lower (respectively upper) bandwidth equal to zero, a tridiagonal matrix with nonzero subdiagonal and supdiagonal entries has lower and upper bandwidths equal to one, and so on.

A nonsingular matrix A with lower bandwidth b_L and upper band with b_U has the shape

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,b_U+1} & & \\ \vdots & & & \ddots & \\ a_{b_L+1,1} & & & & a_{n-b_U,n} \\ & \ddots & & & \vdots \\ & & a_{n,n-b_L} & \cdots & a_{n,n} \end{bmatrix}$$

and so we need at most $(b_L + b_U)\, n$ entries to represent it.

Its LU factorization (whenever it exists!) preserves its band structure: we have that

$$A = L\, U$$

with L unit lower triangular with lower bandwidth b_L and U upper triangular with upper bandwidth b_U.

Indeed, this factorization can be computed iteratively using Algorithm 1.2.2 skipping the pivoting strategy. At the j-th step, the j-th Schur complement is computed with the formula

$$a_{i,k} \leftarrow a_{i,k} - a_{i,j}\, a_{j,k} \qquad \text{for } i, k = j+1, \dots, n.$$

Hence the lower and upper bandwidths of this Schur complement are bounded by those of the previous one, and so by those of the input matrix A.

Thus this factorization can be computed by overwriting the nontrivial entries of A and so with at most $(b_L + b_U)\, n$ memory slots, and this can be done using at most

$$2\, n\, b_L\, b_U + O(n\,(b_L + b_U)) \text{ flops.}$$

Example 1.4.3. Consider the 4×4 matrix

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}$$

which has lower and upper bandwidths equal to 1. Applying Algorithm 1.2.2 without pivoting, for $j = 1$ we obtain

$$A = \begin{bmatrix} 2 & -1 & & \\ 4/2 & -1 - 2\cdot(-1) & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & -2 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we have that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1/1 & -2 - (-1)\cdot 3 & 1 \\ & & 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we obtain that

$$A = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3/1 & 4 - 3\cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & & \\ 2 & 1 & 3 & \\ & -1 & 1 & 1 \\ & & 3 & 1 \end{bmatrix}.$$

We conclude that

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 3 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

GEPP can exploit band structure, but the band properties of the factors L and U are less simple. Indeed, GEPP produces a factorization

$$A = P\, L\, U$$

where U is banded with upper bandwidth $b_L + b_U$, and L has at most $b_L + 1$ nonzero entries per column.

Indeed, at the j-th step of Algorithm 1.2.2, pivoting can only be done within the first b_L rows, because the others have only zeros on the j-th column. Hence the corresponding permutation can increase the upper bandwidth by b_L, and it can also reorder the entries of the previously computed columns of L.

Example 1.4.4. Consider again the matrix in Example 1.4.3

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}$$

Applying GEPP (Algorithm 1.2.2), for $j = 1$ we first swap rows 1 and 2 and then we compute the first column of L, the first row of U, and the first Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 2/4 & -1 - \frac{1}{2}\cdot(-1) & 0 - \frac{1}{2}\cdot 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ \frac{1}{2} & \frac{-1}{2} & \frac{-3}{2} & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

For $j = 2$ we swap the rows 2 and 3 of A and then compute the second column of L, row of U, and Schur complement:

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{-1}{2}/(-1) & \frac{-3}{2} - \frac{1}{2}\cdot(-2) & 0 - \frac{1}{2}\cdot 1 \\ 0 & 0 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} & \frac{-1}{2} \\ 0 & 0 & 3 & 4 \end{bmatrix}.$$

Finally, for $j = 3$ we swap rows 3 and 4 and we compute the third column of L, row of U and Schur complement to obtain

$$A = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2}/3 & \frac{-1}{2} - \frac{-1}{6}\cdot 4 \end{bmatrix} = \begin{bmatrix} 4 & -1 & 3 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 3 & 4 \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & \frac{1}{6} \end{bmatrix}.$$

We conclude that

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{6} & 1 \end{bmatrix}, \quad U \begin{bmatrix} 4 & -1 & 3 & 0 \\ & -1 & -2 & 1 \\ & & 3 & 4 \\ & & & \frac{1}{6} \end{bmatrix}.$$

In this case, U has upper bandwidth 2 and L has at most 2 nonzero elements per column.

1.5. Iterative methods

Iterative methods for solving the linear equation $Ax = b$ are used when the direct methods, like GEPP, require either too much time or too much space. They are usually based on matrix-vector multiplications, and so they are particularly convenient when the cost of this operation is low, like it happens for sparse matrices.

These methods do not produce an exact answer after a finite number of steps, but rather decrease the error by some amount after each step. The final error depends on how many iterations are done, and on the properties of the considered method and the matrix to which it is applied.

1.5.1. Splittings and iterative methods. Let A be a nonsingular $n \times n$ matrix and b an n-vector. Given an initial n-vector x_0, an iterative method generate a sequence of n-vectors

$$(x_l)_{l\geq 0}$$

hopefully converging to the solution $x = A^{-1}\, b$, and where each vector is easy to compute from the previous one.

A *splitting* of A is a decomposition

$$A = M - K$$

with M nonsingular. Such a decomposition produces an iterative method as above in the following way: the equation $A\,x = b$ is equivalent to $M\,x = K\,x + b$ or still to

$$x = M^{-1}K\,x + M^{-1}b.$$

Decoupling both sides of this equality we obtain the iteration

$$x_l = R\,x_{l-1} + c \qquad \text{for } l \geq 1 \tag{1.32}$$

with $R = M^{-1}K$ and $c = M^{-1}b$. When this iteration converges, its limit x_∞ satisfies the equation $x_\infty = R\,x_\infty + c$ or equivalently

$$A\,x_\infty = b,$$

and so this limit *is* the solution of the linear equation.

The convergence of this method depends on the *spectral radius* of the iteration matrix, denoted by $\rho(R)$ and defined as the maximum absolute value of its eigenvalues. Indeed, the iteration (1.32) converges for every choice of the initial vector x_0 if and only if

$$\rho(R) < 1. \tag{1.33}$$

To see this, let x be the solution of the linear equation $A\,x = b$. If $\rho(R) \geq 1$ then choose x_0 such that $x - x_0$ is an eigenvector for an eigenvalue λ of R of absolute value ≥ 1. Hence

$$x - x_l = R^l\,(x - x_0) = \lambda^l\,(x - x_0),$$

and so x_l does not converge to x in this case.

For the converse, suppose for simplicity that R is diagonalizable, so that we can factor this matrix as

$$R = S\,\Lambda\,S^{-1}$$

with S nonsingular and $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ diagonal. Then for each $l \geq 0$ we have that

$$\begin{array}{rr@{\,=\,}l} & x & R\,x + c \\ - & x_l & R\,x_{l-1} + c \\ \hline & x - x_l & R\,(x - x_{l-1}) \end{array}$$

which readily implies that

$$x - x_l = R^l(x - x_0) = S\,\Lambda^l\,S^{-1}(x - x_0).$$

Taking ∞-norms we obtain that

$$\begin{aligned} \|x - x_l\|_\infty &= \|S\,\Lambda^l\,S^{-1}(x - x_0)\|_\infty \\ &\leq \|S\|_\infty \|\Lambda\|_\infty^l \|S^{-1}\|_\infty \|x - x_0\|_\infty = \kappa(S)\,\rho(R)^l \|x - x_0\|_\infty \end{aligned} \tag{1.34}$$

because $\|\Lambda\|_\infty = \max_j |\lambda_j| = \rho(R)$. Hence if the condition (1.33) holds, then the upper bound in (1.34) tends to 0 when $l \to +\infty$ and to the iteration converges.

Moreover, if $x \neq 0$ or equivalently $b \neq 0$, then we deduce from (1.34) the upper bound for the relative error

$$\frac{\|x - x_l\|_\infty}{\|x\|_\infty} \leq \kappa(S)\, \rho(R)^l\, \frac{\|x - x_0\|_\infty}{\|x\|_\infty}.$$

Setting $\gamma(R) = -\log_\beta \rho(R) > 0$ with β the base of our floating point system, this upper bound can be rewritten as

$$-\log_\beta \Big(\frac{\|x - x_l\|_\infty}{\|x\|_\infty}\Big) \geq \gamma(R)\, l - \log_\beta \kappa(S) - \log_\beta \Big(\frac{\|x - x_0\|_\infty}{\|x\|_\infty}\Big)$$

which, following the rule of thumb (1.20), shows that when the condition (1.33) holds, the precision of the approximations increases *linearly* with rate $\gamma(R)$: the smaller the spectral radius is, the higher is the rate of convergence.

To design an efficient iterative scheme for a given problem, we need to find a splitting $A = M - K$ verifying the conditions:

(1) $R = M^{-1}K$ and $c = M^{-1}b$ are easy to compute,
(2) $\rho(R)$ is small.

1.5.2. The Richardson iteration. This is a simple iterative method that will mainly serve us as an example where the relevant aspects can be fully understood.

Given a nonsingular $n \times n$ matrix A and an n-vector b, the *Richardson iteration* aims at improving an approximate solution $\widehat{x}$ of the equation $A\,x = b$ by adding to it multiple of the residual $b - A\,\widehat{x}$. Precisely, for a parameter $\omega > 0$ this iteration starts from an initial n-vector x_0 and constructs the sequence $(x_l)_{l\geq 0}$ by setting

$$x_l = x_{l-1} + \omega\,(b - A\,x_{l-1}) \quad \text{for } l \geq 1. \tag{1.35}$$

In the context of the previous section, it corresponds to the splitting $A = M_\omega - K_\omega$ with $M_\omega = \omega^{-1}\,\mathbb{1}_n$ and $K_\omega = \omega^{-1}\,\mathbb{1}_n - A$, and so we can rewrite it as

$$x_l = R_\omega\, x_{l-1} + c_\omega$$

with $R_\omega = M_\omega^{-1} K_\omega = \mathbb{1}_n - \omega\,A$ and $c_\omega = M^{-1}b = \omega\,b$.

The eigenvalues of this iteration matrix are of the form $1 - \omega\,\lambda$ with $\lambda \in \mathbb{C}$ an eigenvalue of A. Hence if we denote by $\Lambda(A)$ the *spectrum* of A, that is, the set of its eigenvalues, then

$$\rho(R_\omega) = \max_{\lambda \in \Lambda(A)} |1 - \omega\,\lambda|.$$

Hence by the criterion in (1.33), the Richardson iteration (1.35) converges for every choice of initial vector x_0 if and only if

$$|1 - \omega\,\lambda| < 1 \quad \text{for all } \lambda \in \Lambda(A) \tag{1.36}$$

or equivalently, if the spectrum of A lies in the open disk centered at the point ω^{-1} and of radius ω^{-1}:

To make the full convergence analysis, we assume for simplicity that the eigenvalues of A are real. We respectively denote by $\lambda_{\min}$ and $\lambda_{\max}$ the minimal and the maximal ones, so that $\lambda_{\min} \leq \lambda \leq \lambda_{\max}$ for all $\lambda \in \Lambda(A)$.

The condition for convergence in (1.36) then translates into $-1 < 1 - \omega\,\lambda_{\max}$ and $1 - \omega\,\lambda_{\min} < 1$, which are equivalent to

$$0 < \lambda_{\min} \quad \text{and} \quad \lambda_{\max} < \frac{2}{\omega}.$$

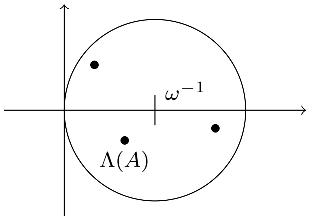

Figure 1.5.1. Convergence of the Richardson iteration

Hence in this situation, the iteration converges if and only if all the eigenvalues are positive and the parameter satisfies

$$\omega < \frac{2}{\lambda_{\max}}.$$

When this is the case, the spectral radius is given by the piecewise affine function

$$\rho(R_\omega) = \max(|1 - \omega\,\lambda_{\min}|, |1 - \omega\,\lambda_{\max}|),$$

whose graph is shown in Figure 1.5.2.

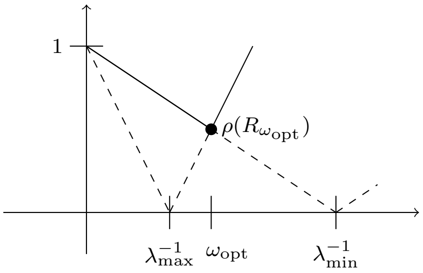

Figure 1.5.2. The spectral radius of R_ω

As shown in this figure, the best value ω_{opt}, that is the value that minimizes the spectral radius, is reached at the point where the graph of $\omega \mapsto |1 - \omega\,\lambda_{\max}|$ with positive slope crosses the graph of $\omega \mapsto |1 - \omega\,\lambda_{\min}|$ with negative slope, that is when $-1 + \lambda_1\,\omega_{\text{opt}} = 1 - \lambda_n\omega_{\text{opt}}$. This gives the value

$$\omega_{\text{opt}} = \frac{2}{\lambda_{\max} + \lambda_{\min}},$$

whose corresponding spectral radius is

$$\rho(\omega_{\text{opt}}) = \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}.$$

If A has very large and very small eigenvalues, then the Richardson iteration will be extremely slow, even when using the optimal parameter ω_{opt}. Moreover, the determination of this optimal parameter requires knowledge of the maximal and minimal eigenvalue, which is not easily accessible in realistic problems.

1.5.3. Basic iterative methods.

For the methods to be discussed in the section, we assume that all the diagonal entries of A are nonzero and we decompose it as

$$A = D - \widetilde{L} - \widetilde{U} = D\,(\mathbb{1}_n - L - U)$$

with D diagonal, $\widetilde{L}$ and L strictly lower triangular, and $\widetilde{U}$ and U strictly upper triangular.

Jabobi's method can be interpreted as going successively through each scalar linear equation, so that at the l-th iteration the updated value of j-th component of the approximate solution satisfies the j-th scalar linear equation when evaluated at the previous values of the other components. Precisely, for each $l \geq 1$ we want to have

$$a_{j,1}\,x_{l-1,1} + \cdots + a_{j,j-1}\,x_{l-1,j-1} + a_{j,j}\,x_{l,j} + a_{l,j+1}\,x_{l-1,j+1} + \cdots + a_{l,n}\,x_{l-1,n} = b_j$$

for $j = 1, \ldots, n$. This leads to an iterative method starting with a given initial vector x_0 and where the l-th iteration is defined by the rule:

Algorithm 1.5.1 (Jacobi's method)

```
for $j = 1, \ldots, n$
    $x_{l,j} \leftarrow \frac{1}{a_{j,j}} \big(b_j - \sum_{k \neq j} a_{j,k}\,x_{l-1,k}\big)$
```

It can be expressed in matrix notation as $x_l = R_{\mathrm{J}}\,x_{l-1} + c_{\mathrm{J}}$ with

$$R_{\mathrm{J}} = D^{-1}(\widetilde{L} + \widetilde{U}) = L + U \quad \text{and} \quad c_{\mathrm{J}} = D^{-1}b. \tag{1.37}$$

The *Gauss-Seidel method* is motivated by the observation that at the j-th step of Jacobi's method we have already computed improved values for the first $j-1$ components of the approximate solution, and so it is reasonable to take advantage of them when updating the j-th component: for each $l \geq 0$ we aim at

$$a_{j,1}\,x_{l,1} + \cdots + a_{j,j-1}\,x_{l,j-1} + a_{j,j}\,x_{l,j} + a_{l+1,j+1}\,x_{l-1,j+1} + \cdots + a_{l,n}\,x_{l-1,n} = b_j$$

for $j = 1, \ldots, n$. The corresponding iterative method is defined by the rule:

Algorithm 1.5.2 (Gauss-Seidel method)

```
for $j = 1, \ldots, n$
    $x_{l,j} \leftarrow \frac{1}{a_{j,j}} \Big(b_j - \underbrace{\sum_{k<j} a_{j,k}\,x_{l,k}}_{\text{updated } x\text{'s}} - \underbrace{\sum_{k>j} a_{j,k}\,x_{l-1,k}}_{\text{older } x\text{'s}}\Big)$
```

In matrix notation, we have that $x_{l+1} = R_{\mathrm{GS}}\,x_l + c_{\mathrm{GS}}$ with

$$\begin{aligned} R_{\mathrm{GS}} &= (D - \widetilde{L})^{-1}\widetilde{U} = (\mathbb{1}_n - L)^{-1}U, \\ c_{\mathrm{GS}} &= (D - \widetilde{L})^{-1}b = (\mathbb{1}_n - L)^{-1}D^{-1}b. \end{aligned} \tag{1.38}$$

Successive overrelaxation for a parameter $\omega \in \mathbb{R}$ is a weighted average of the vectors x_{l+1} and x_l from the Gauss-Seidel method: it is defined by

$$x_l^{\mathrm{SOR}(\omega)} = (1-\omega)\,x_{l-1}^{\mathrm{GS}} + \omega x_l^{\mathrm{GS}} \qquad \text{for } l \geq 1.$$

This yields the following iterative scheme:

Algorithm 1.5.3 (SOR(ω))

```
for $j = 1, \dots, n$
    $x_{l,j} \leftarrow (1-\omega) x_{l-1,j} + \frac{\omega}{a_{j,j}} \big(b_j - \sum_{k<j} a_{j,k}\, x_{l,k} - \sum_{k>j} a_{j,k}\, x_{l-1,k}\big)$
```

In matrix notation, we have that $x_l = R_{\mathrm{SOR}(\omega)}\, x_{l-1} + c_{\mathrm{SOR}(\omega)}\, b$ with

$$\begin{aligned} (1.39)\quad R_{\mathrm{SOR}(\omega)} &= (D - \omega\, \widetilde{L})^{-1}((1-\omega)\, D + \omega\, \widetilde{U}) = (\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\, \mathbb{1}_n + \omega\, U), \\ c_{\mathrm{SOR}(\omega)} &= \omega\, (D - \omega\, \widetilde{L})\, b = \omega\, (\mathbb{1}_n - \omega\, L)\, D^{-1} b. \end{aligned}$$

We distinguish three cases, depending on the value of the relaxation parameter ω: $\omega = 1$ is equivalent to the Gauss-Seidel method, $\omega < 1$ it is called *underrelaxation*, and $\omega > 1$ is called *overrelaxation*. A somewhat superficial motivation for overrelaxation is that if going from x_{l-1} to x_l is a good direction towards the solution, then moving $\omega > 1$ times as far should be even better.

Example 1.5.1. Let

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad \text{and} \quad b = (1, 0).$$

The solution of the linear equation $A\, x = b$ is the 2-vector $x = (\frac{3}{2}, \frac{-1}{3})$.

The decompositions $A = D - \widetilde{L} - \widetilde{U} = D\, (\mathbb{1}_2 - L - U)$ are respectively given by

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ -1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ \frac{-1}{2} & 0 \end{bmatrix} - \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & 0 \end{bmatrix} \right)$$

Hence

$$R_{\mathrm{J}} = L + U = \begin{bmatrix} 0 & \frac{-1}{2} \\ \frac{-1}{2} & 0 \end{bmatrix} \quad \text{and} \quad c_{\mathrm{J}} = D^{-1} b = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}$$

and so the corresponding Jacobi iteration writes down as

$$\begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ \frac{-1}{2} & 0 \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{-x_{l-1,2}}{2} + \frac{1}{2} \\ \frac{x_{l-1,1}}{2} \end{bmatrix}.$$

The corresponding characteristic polynomial is

$$\chi_{R_{\mathrm{J}}} = \det \begin{bmatrix} -t & \frac{-1}{2} \\ \frac{-1}{2} & -t \end{bmatrix} = t^2 - \frac{1}{4}.$$

Its spectrum is $\Lambda(R_{\mathrm{J}}) = \{t \mid \chi_{R_{\mathrm{J}}}(t) = 0\} = \big\{ \pm \frac{1}{2} \big\}$ and so its spectral radius is

$$\rho(R_{\mathrm{J}}) = \frac{1}{2},$$

showing that the method converges to the solution x for every choice of initial vector.

We also have that

$$\begin{aligned} R_{\mathrm{GS}} &= (\mathbb{1}_2 - L)^{-1} U = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} \end{bmatrix}, \\ c_{\mathrm{GS}} &= (\mathbb{1}_2 - L)^{-1} D^{-1} b = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{4} \end{bmatrix} \end{aligned}$$

and so the corresponding Gauss-Seidel iteration writes down as

$$\begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{2} \\ 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{4} \end{bmatrix} = \begin{bmatrix} \frac{-x_{l,2}}{2} + \frac{1}{2} \\ \frac{x_{l,2}}{4} - \frac{1}{4} \end{bmatrix}.$$

We have that

$$\chi_{R_{\text{GS}}} = \det \begin{bmatrix} -t & \frac{1}{2} \\ 0 & -t+\frac{1}{4} \end{bmatrix} = t^2 - \frac{1}{4}\,t.$$

Hence $\Lambda(R_{\text{GS}}) = \left\{0, \frac{1}{4}\right\}$ and so $\rho(R_{\text{GS}}) = \frac{1}{4}$. We have that

$$\rho(R_{\text{GS}}) = \rho(R_{\text{J}})^2$$

and so in this example, the Gauss-Seidel method converges with the *double of the speed* of the Jacobi method.

Now for $\omega \in \mathbb{R}$ we have that

$$\begin{aligned} R_{\text{SOR}(\omega)} &= (\mathbb{1}_2 - \omega\, L)^{-1}((1-\omega)\,\mathbb{1}_2 + \omega\, U) \\ &= \begin{bmatrix} 1 & 0 \\ \omega/2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1-\omega & -\omega/2 \\ 0 & 1-\omega \end{bmatrix} = \begin{bmatrix} 1-\omega & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 \end{bmatrix} \end{aligned}$$

and

$$c_{\text{SOR}(\omega)} = \omega\,(\mathbb{1}_2 - \omega\, L)^{-1} D^{-1} b = \omega \begin{bmatrix} 1 & 0 \\ \frac{\omega}{2} & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\omega}{2} \\ \frac{-\omega^2}{4} \end{bmatrix},$$

and so the corresponding SOR(ω) iteration writes down as

$$\begin{aligned} \begin{bmatrix} x_{l,1} \\ x_{l,2} \end{bmatrix} &= \begin{bmatrix} 1-\omega & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 \end{bmatrix} \begin{bmatrix} x_{l-1,1} \\ x_{l-1,2} \end{bmatrix} + \begin{bmatrix} \frac{\omega}{2} \\ \frac{-\omega^2}{4} \end{bmatrix} \\ &= \begin{bmatrix} (1-\omega)\, x_{l-1,1} - \frac{\omega}{2}\, x_{l-1,2} + \frac{\omega}{2} \\ (\frac{\omega^2}{2} - \frac{\omega}{2})\, x_{l-1,1} + (\frac{\omega^2}{4} - \omega + 1)\, x_{l-1,2} - \frac{\omega^2}{4} \end{bmatrix}. \end{aligned}$$

We have that

$$\chi_{R_{\text{SOR}(\omega)}} = \det \begin{bmatrix} 1-\omega-t & -\frac{\omega}{2} \\ \frac{\omega^2}{2} - \frac{\omega}{2} & \frac{\omega^2}{4} - \omega + 1 - t \end{bmatrix} = t^2 + \Big(\frac{-\omega^2}{4} + 2\,\omega - 2\Big)\, t + (\omega^2 - 2\,\omega + 1).$$

The spectral radius $\rho(R_{\text{SOR}(\omega)})$ is the maximum of the absolute value of the zeros of this polynomial, and its graph is shown for $\omega \in [0,2]$ is shown in Figure 1.5.3.

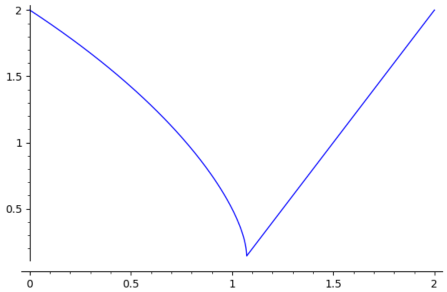

FIGURE 1.5.3. The spectral radius of SOR(ω)

The optimal value of the relaxation parameter is $\omega_{\text{opt}} = 1.0717$ up to 4 decimal digits, and the corresponding spectral radius is $\rho(R_{\text{SOR}(1.0717)}) = 0.1535$.

1.5.4. Convergence of the basic iterative schemes. We will give some criteria that, in some specific situation, guarantee the convergence of the basic iterative schemes. Since the spectral radius is difficult to compute, these convergence criteria can be useful in practice.

As in §1.5.3, we assume that all the diagonal entries of A are nonzero and we consider the decomposition

$$A = D - \widetilde{L} - \widetilde{U}$$

with D diagonal, $\widetilde{L}$ strictly lower triangular and $\widetilde{U}$ strictly upper triangular.

The matrix A is *(column) diagonally dominant* if the absolute value of each diagonal entry is greater the sum of the absolute values of the rest of the entries in its column, that is

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

If A is diagonally dominant then both the Jacobi and the Gauss-Seidel iterations will converge for every choice of initial vector x_0. To see this, let

$$R_{\mathrm{J}} = D^{-1}(\widetilde{L} + \widetilde{U}) \quad \text{and} \quad R_{\mathrm{GS}} = (D - \widetilde{L})^{-1}\widetilde{U}$$

be the corresponding iteration matrices as in (1.37) and (1.38). For each $\lambda \in \Lambda(R_{\mathrm{J}})$ let x be a corresponding eigenvector for it, and k the index of its largest component. Up to a normalization, we can suppose that

$$x_k = 1 \quad \text{and} \quad |x_j| \leq 1 \text{ for all } j.$$

Then the equality $R_{\mathrm{J}}\, x = \lambda\, x$ implies that $(\widetilde{L} + \widetilde{U})\, x = \lambda\, x$. The k-th entry of this later equality is

$$-\sum_{j\neq k} \frac{a_{k,j}}{a_{k,k}}\, x_j = \lambda,$$

which implies that

$$|\lambda| \leq \sum_{j\neq k} \frac{|a_{k,j}|}{|a_{k,k}|}\, |x_j| < 1,$$

because $|x_j| \leq 1$ for all j and A is diagonally dominant. Since this inequality holds for every eigenvalue of R_{J}, we deduce that $\rho(R_{\mathrm{J}}) < 1$, which gives the result for the Jacobi method.

Similarly, for each $\lambda \in \Lambda(R_{\mathrm{GS}})$ let x a corresponding eigenvector, and k the index of the largest component of x. Again, we assume that $x_k = 1$ and $|x_j| \leq 1$ for all j. From the equality $R_{\mathrm{GS}}\, x = \lambda\, x$ we deduce that $\widetilde{U}\, x = \lambda\, (D - \widetilde{L})\, x$ and so

$$\sum_{j<k} a_{k,j}\, x_j = \lambda \Big(a_{k,k} + \sum_{j>k} a_{k,j}\, x_j \Big).$$

This implies that

$$|\lambda| \leq \frac{\sum_{j<k} |a_{k,j}|\, |x_j|}{|a_{k,k}| - \sum_{j>k} |a_{k,j}|\, |x_j|} \leq \frac{\sum_{j<k} |a_{k,j}|}{|a_{k,k}| - \sum_{j>k} |a_{,j}|} < 1,$$

again because $|x_j| \leq 1$ for all j and A is diagonally dominant, which implies that $|a_{k,k}| - \sum_{j>k} |a_{k,j}| > \sum_{j<k} |a_{k,j}|$. Since this inequality holds for every eigenvalue of R_{GS}, we deduce the convergence of the Gauss-Seidel method in this case.

For the successive overrelaxation method, the condition on the relaxation parameter

$$0 < \omega < 2 \tag{1.40}$$

is necessary for the convergence of the method. Indeed, by (1.39) the corresponding iteration matrix is

$$R_{\mathrm{SOR}(\omega)} = (\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\,\mathbb{1}_n + \omega\, U)$$

and so its determinant can be computed as

$$\begin{aligned} \det(R_{\mathrm{SOR}(\omega)}) &= \det((\mathbb{1}_n - \omega\, L)^{-1}((1-\omega)\,\mathbb{1}_n + \omega\, U)) \\ &= \det(\mathbb{1}_n - \omega\, L)^{-1} \det((1-\omega)\,\mathbb{1}_n + \omega\, U) = (1-\omega)^n. \end{aligned} \tag{1.41}$$

The determinant of a matrix coincides with the product of its eigenvalues, repeated with the corresponding multiplicity, and so

$$|\det(R_{\mathrm{SOR}(\omega)})| \le \rho(R_{\mathrm{SOR}(\omega)})^n. \tag{1.42}$$

The inequalities (1.42) and (1.41) readily imply the lower bound

$$\rho(R_{\mathrm{SOR}(\omega)}) \ge |1-\omega|,$$

and so the condition for the spectral radius in (1.33) can only be satisfied whenever the condition (1.40) holds.

On the other hand, if A is a symmetric and positive definite matrix with real coefficients then $\mathrm{SOR}(\omega)$ converges for every $0 < \omega < 2$ [**Dem97**, Theorem 6.4]. In particular, the Gauss-Seidel method ($\omega = 1$) always converges in this situation.