Chapter 4

Linear Systems – Small Matrices

The solution of a **linear system of equations**

$$A\mathbf{x} = \mathbf{b}$$

with a square invertible matrix $A \in \mathbb{R}^{n\times n}$ and a right-hand side vector $\mathbf{b} \in \mathbb{R}^n$ is one of the most frequent tasks in numerical analysis. The matrix A and the vectors $\mathbf{x}$, $\mathbf{b}$ have the entries

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}.$$

Example 4.1 Consider the following system of 3 linear equations:

$$\begin{array}{rrrcr} 2x_1 & -2x_2 & +4x_3 & = & 6 \\ -5x_1 & +6x_2 & -7x_3 & = & -7 \\ 3x_1 & +2x_2 & +x_3 & = & 9 \end{array}$$

Using the definition of the matrix-vector product, this can be written in matrix-vector form as

$$\begin{pmatrix} 2 & -2 & 4 \\ -5 & 6 & -7 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 6 \\ -7 \\ 9 \end{pmatrix}.$$

Example 4.2 Figure 4.1 shows a hydraulic network[11] of 10 pipelines. It is fed by a water reservoir having a constant pressure of $p = 10$ bar. Here and in the following, pressure values refer to the difference between the real pressure and the atmospheric pressure. The flow rate Q_j (in m^3/s) of the jth pipeline is proportional

[11] Example taken from [A. Quarteroni, F. Saleri, and P. Gervasi. Springer, 2010].

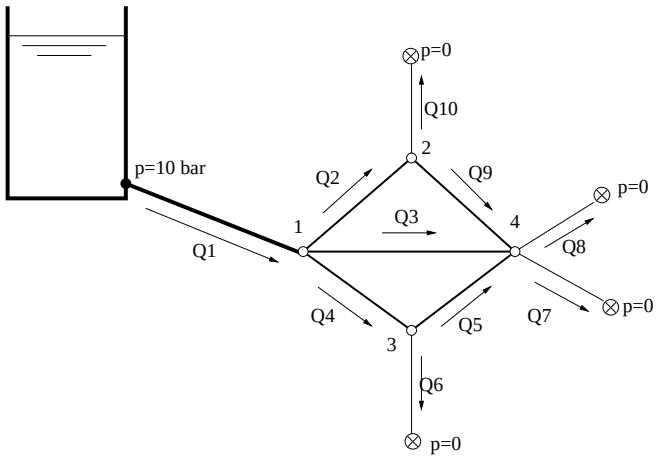


Figure 4.1: Hydraulic network from Example 4.2.

to the length L_j (in m) of the pipeline and the pressure difference Δp_j at both ends of the pipeline:

$$Q_j = k_j L_j \Delta p_j \tag{4.1}$$

The constant k_j denotes the hydraulic resistance (in m/(bar s)), which depends on the shape of the pipe and the fluid viscosity. It is assumed that the water flows from outlets (marked by $\otimes$ in the figure) at atmospheric pressure, and hence $p = 0$ at the exterior nodes of the network. To determine the pressure at the internal nodes $1, 2, 3, 4$, we can use that the flow rates at each internal node must sum up to zero. Denoting these pressures by $\mathbf{p} = [p_1, p_2, p_3, p_4]^T$, this implies for node 1:

$$\begin{aligned} & Q_1 - Q_2 - Q_3 - Q_4 = 0 \\ \overset{(4.1)}{\Longrightarrow} \quad & k_1 L_1 (p_1 - 10) - k_2 L_2 (p_2 - p_1) - k_4 L_4 (p_3 - p_1) - k_3 L_3 (p_4 - p_1) = 0 \\ \Longrightarrow \quad & -10 k_1 L_1 = -(k_1 L_1 + k_2 L_2 + k_3 L_3 + k_4 L_4) p_1 + k_2 L_2 p_2 + k_4 L_4 p_3 + k_3 L_3 p_4 \end{aligned}$$

To proceed, we choose concrete values for k_j and L_j as follows.

pipeline	k_j	L_j	pipeline	k_j	L_j	pipeline	k_j	L_j
1	0.01	20	2	0.005	10	3	0.005	14
4	0.005	10	5	0.005	10	6	0.002	8
7	0.002	8	8	0.002	8	9	0.005	10
10	0.002	8						

Inserting these values into the equation above gives

$$-2 = -0.37\, p_1 + 0.05\, p_2 + 0.05\, p_3 + 0.07\, p_4.$$

Similarly, linear equations can be derived for the other internal nodes $2, 3, 4$. In summary, this yields the linear system $A\mathbf{p} = \mathbf{b}$ with

$$A = \begin{pmatrix} -0.370 & 0.050 & 0.050 & 0.070 \\ 0.050 & -0.116 & 0 & 0.050 \\ 0.050 & 0 & -0.116 & 0.050 \\ 0.070 & 0.050 & 0.050 & -0.202 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} -2 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The solution of this linear system will be presented in Example 4.11. ◇

4.1 Triangular Matrices

Before coming to the solution of a general linear system $A\mathbf{x} = \mathbf{b}$, we first consider two special cases for the matrix A.
A **lower triangular matrix** A is a square matrix satisfying

$$a_{ij} = 0 \qquad \text{for all } i, j \text{ with } i < j.$$

An **upper triangular matrix** A is a square matrix satisfying

$$a_{ij} = 0 \qquad \text{for all } i, j \text{ with } i > j.$$

Often, we will denote lower triangular matrices with the letter L and upper triangular matrices with the letter U. The definitions imply the shapes

$$L = \begin{pmatrix} \ell_{11} & 0 & \cdots & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & \\ \ell_{31} & \ell_{32} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn-1} & \ell_{nn} \end{pmatrix}, \qquad U = \begin{pmatrix} u_{11} & u_{12} & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & \cdots & u_{2n} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & u_{nn} \end{pmatrix}.$$

Pictorially:

$$L = \text{◺}, \quad U = \text{◹}.$$

The solution of linear systems with (lower or upper) triangular matrices is quite simple. For example, consider

$$\begin{pmatrix} 1 & 2 & -1 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix} \qquad \Leftrightarrow \qquad \begin{aligned} x_1 + 2x_2 - x_3 &= 2 \\ 2x_2 + x_3 &= 3 \\ 2x_3 &= 2 \end{aligned}$$

The last equation can be immediately solved: $x_3 = 2/2 = 1$. Inserting this into the second equation gives

$$2x_2 + 2x_3 = 3 \quad \Rightarrow \quad x_2 = \frac{1}{2}(3 - x_3) = 1.$$

Finally, inserting $x_2 = 1$ and $x_3 = 1$ into the first equation gives

$$x_1 + 2x_2 - x_3 = 2 \quad \Rightarrow \quad x_1 = 2 - 2x_2 + x_3 = 1.$$

This process of eliminating the variables $x_n, x_{n-1}, \ldots$ is called *backward substitution*. Similarly, a linear system with a lower triangular matrix can be solved by eliminating the variables $x_1, x_2, \ldots$. This process is called *forward substitution*.

For general triangular matrices, forward and backward substitution are given by the following two algorithms.

Algorithm 4.3
Forward substitution

Input: Invertible lower triangular $L \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$.

Output: Solution $\mathbf{x}$ of $L\mathbf{x} = \mathbf{b}$.

for $i = 1, 2, \ldots, n$ **do**

$$x_i := \frac{1}{\ell_{ii}} \left(b_i - \sum_{k=1}^{i-1} \ell_{ik} x_k \right)$$

end for

Algorithm 4.4
Backward substitution

Input: Invertible upper triangular $U \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$.

Output: Solution $\mathbf{x}$ of $U\mathbf{x} = \mathbf{b}$.

for $i = n, n-1, \ldots, 1$ **do**

$$x_i := \frac{1}{u_{ii}} \left(b_i - \sum_{k=i+1}^{n} u_{ik} x_k \right)$$

end for

It is a good exercise to convince yourself that the right-hand side of the assignment in both algorithms only contains terms that are already known.

Remark 4.5 *In* PYTHON*, linear systems can be solved with the commands*

`numpy.linalg.solve(A, b)` *or* `scipy.linalg.solve(A, b)`

for a square, invertible matrix A. However, these functions do not automatically check whether A is triangular and, therefore, they are unnecessarily slow in such situations. To solve triangular linear systems, one should call `scipy.linalg.solve` *with the option* `assume_a` *set to 'upper triangular' or 'lower triangular'. Alternatively, one can also call* `scipy.linalg.solve_triangular`*.*

Let us perform a complexity analysis for Algorithm 4.3. The cost of an algorithm is determined by the number of elementary operations $+, -, *, /$ and elementary function evaluations. Each operation / evaluation is counted as one **flop** (floating point operation). The ith loop of Algorithm 4.3 performs 1 division, $i-1$ multiplications, and $i-1$ additions/subtractions; a total of $2i-1$ flops. Therefore, the total cost of Algorithm 4.3 is given by

$$\sum_{i=1}^{n} (2i - 1) = n^2 \text{ flops.} \tag{4.2}$$

The cost of Algorithm 4.4 is the same.

4.2 LU factorization

Knowing that a linear system with a triangular matrix is considerably simple to solve, we now try to reduce a general system to this case. To be more precise, we

will use a variant of **Gaussian elimination** to write the matrix A as a product of triangular matrices:

$$A = LU = \text{◺} \cdot \text{◹} \tag{4.3}$$

Once we know such a factorization, we can solve the linear system $A\mathbf{x} = \mathbf{b}$ with forward and backward substitution. If we introduce the auxiliary vector $\mathbf{y} = U\mathbf{x}$ then $\mathbf{b} = A\mathbf{x} = LU\mathbf{x} = L(U\mathbf{x}) = L\mathbf{y}$. Hence, we can solve $(LU)\mathbf{x} = \mathbf{b}$ in two steps:

1. solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ with Algorithm 4.3;
2. solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ with Algorithm 4.4.

The LU factorization of a matrix $A \in \mathbb{R}^{n\times n}$ proceeds in $n-1$ steps, by eliminating column-by-column the entries of A below the diagonal.

Example 4.6 We illustrate the computation of an LU factorization for the matrix from Example 4.1:

$$A = \begin{pmatrix} 2 & -2 & 4 \\ -5 & 6 & -7 \\ 3 & 2 & 1 \end{pmatrix}$$

In Step 1, we eliminate the entries below the first diagonal entry, by adding $5/2\times$ row 1 to row 2, and subtracting $3/2\times$ row 1 from row 3. As a result, we obtain the modified matrix

$$A^{(1)} := \begin{pmatrix} 2 & -2 & 4 \\ 0 & 1 & 3 \\ 0 & 5 & -5 \end{pmatrix}.$$

The crucial observation is that this step can be written as a matrix-matrix multiplication:

$$A^{(1)} := L_1 A = L_1 \begin{pmatrix} 2 & -2 & 4 \\ -5 & 6 & -7 \\ 3 & 2 & 1 \end{pmatrix}, \quad \text{with} \quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 5/2 & 1 & 0 \\ -3/2 & 0 & 1 \end{pmatrix}.$$

In Step 2, we eliminate the remaining entry 5 in $A^{(1)}$ below the second diagonal element. This can be achieved by subtracting $5\times$ row 2 from row 3, leading to

$$A^{(2)} := \begin{pmatrix} 2 & -2 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & -20 \end{pmatrix}$$

Again, this can be written as a matrix-matrix product

$$A^{(2)} := L_2 A^{(1)} = L_2 \begin{pmatrix} 2 & -2 & 4 \\ 0 & 1 & 3 \\ 0 & 5 & -5 \end{pmatrix}, \quad \text{with} \quad L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -5 & 1 \end{pmatrix}.$$

Setting $U := A^{(2)}$, the two steps above can be summarized as

$$A = LU, \quad \text{with} \quad L = L_1^{-1} L_2^{-1}.$$

It turns out that – due to their very special structure – the inverses of L_1 and L_2 can be simply obtained by negating the elements below the diagonal:

$$L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -5/2 & 1 & 0 \\ 3/2 & 0 & 1 \end{pmatrix}, \qquad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 5 & 1 \end{pmatrix}.$$

This can be easily verified by checking $L_1 L_1^{-1} = I_3$ and $L_2 L_2^{-1} = I_3$. Moreover, again due the very special structure, the product $L_1^{-1} L_2^{-1}$ is simply obtained by collecting all nonzero sub-diagonal elements:

$$L = L_1^{-1} L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -5/2 & 1 & 0 \\ 3/2 & 5 & 1 \end{pmatrix}$$

Hence,

$$A = LU, \quad \text{with} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ -5/2 & 1 & 0 \\ 3/2 & 5 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & -2 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & -20 \end{pmatrix}.$$

which is the LU factorization of A. ◇

The procedure from Example 4.6 easily extends to a general matrix A. *Before* Step k, the modified matrix A takes the form

$$A^{(k-1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} \\ & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ & & \ddots & \ddots & & \vdots \\ & & & a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{pmatrix}. \tag{4.4}$$

(For Step 1, we formally set $A^{(0)} := A$.) For performing Step k, the coefficients

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

are computed. Of course, this is only possible if the so called **pivot element** $a_{kk}^{(k-1)}$ is nonzero. We will come back to this limitation below, in Section 4.3.

The multiplication of the matrix

$$L_k := \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -\ell_{n\,k} & & 1 \end{pmatrix} \tag{4.5}$$

with $A^{(k-1)}$ performs Step k and eliminates all entries below the $(k-1)$th diagonal entry. The whole procedure is then repeated with the resulting matrix

$$A^{(k)} = L_k A^{(k-1)}. \tag{4.6}$$

After $n-1$ steps, we obtain

$$A^{(n-1)} = L_{n-1} L_{n-2} \cdots L_1 A^{(0)} = L_{n-1} L_{n-2} \cdots L_1 A.$$

This can be rewritten as

$$LU = A,$$

where

$$L := L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}, \qquad U = A^{(n-1)}.$$

Note that U is upper triangular by construction. The factors L_k^{-1} of the matrix L are given by

$$L_k^{-1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & \ell_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & \ell_{n\,k} & & & 1 \end{pmatrix}, \tag{4.7}$$

which can be verified by checking $L_k^{-1} L_k = I$. Due to the special structure of these factors, the sub-diagonal entries of L are obtained from collecting the subdiagonal entries of all L_k^{-1}:

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{pmatrix} 1 & & & & & \\ \ell_{21} & \ddots & & & & \\ \ell_{31} & \ddots & 1 & & & \\ \vdots & & \ell_{k+1,k} & 1 & & \\ \vdots & & \vdots & \ddots & \ddots & \\ \ell_{n1} & \cdots & \ell_{n\,k} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix}. \tag{4.8}$$

The procedure above is summarized in the following algorithm.

Algorithm 4.7 (Abstract form of *LU* factorization)
Input: Invertible matrix $A \in \mathbb{R}^{n \times n}$.
Output: LU factorization $A = LU$ with $U = A^{(n-1)}$ and L as in (4.8).

$A^{(0)} := A$
for $k = 1, \ldots, n-1$ **do**
Determine matrix L_k (see (4.5)) by computing the coefficients

$$\ell_{ik} := \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k+1, \ldots, n.$$

Set $A^{(k)} := L_k A^{(k-1)}$.
end for

Not every invertible matrix A has an LU factorization, that is $A = LU$ with L lower triangular with ones and the diagonal and U upper triangular. The following theorem characterizes which ones have.

Theorem 4.8 *Let $A \in \mathbb{R}^{n\times n}$ be invertible. Then A has an LU factorization if and only all leading principal submatrices*

$$A_k = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & \cdots & a_{nn} \end{pmatrix}, \quad k = 1, \ldots, n-1,$$

are also invertible.

Proof. Given an LU factorization $A = LU$, we partition

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{pmatrix}, \quad U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}, \quad L_{11}, U_{11} \in \mathbb{R}^{k\times k}.$$

Then

$$A_k = L_{11}U_{11} \quad \Rightarrow \quad \det(A_k) = \det(L_{11})\det(U_{11}) = \det(U_{11}),$$

where we used that L has ones on the diagonal. Because $0 \neq \det(A) = \det(U) = \prod_{i=1}^{n} u_{ii}$, it follows that $\det(U_{11}) = \prod_{i=1}^{k} u_{ii} \neq 0$ and hence A_k is invertible.

To show the other direction we use the construction above of the LU factorization. For Algorithm 4.7 to succeed we need to have $a_{kk}^{(k-1)} \neq 0$. Suppose that the first $k-1$ steps of Algorithm 4.7 have succeeded (that is, $a_{11} \neq 0, a_{22}^{(1)} \neq 0 \ldots, a_{k-1,k-1}^{(k-2)} \neq 0$). Then

$$A_k = \begin{pmatrix} 1 & & & \\ l_{21} & \ddots & & \\ \vdots & \ddots & 1 & \\ l_{k1} & \cdots & l_{k,k-1} & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{kk}^{(k-1)} \end{pmatrix}.$$

Hence it follows from $0 \neq \det(A_k) = a_{11}a_{22}^{(1)} \cdots a_{kk}^{(k-1)}$ that $a_{kk}^{(k-1)} \neq 0$. Therefore the kth step of Algorithm 4.7 succeeds as well and the claim follows by induction. ∎

To come up with a reasonable implementation of Algorithm 4.7, the matrix-matrix multiplications need to be replaced, as an explicit multiplication would be much too expensive. Moreover, to save memory, we will operate directly on the matrix A instead of creating temporary matrices $A^{(1)}, A^{(2)}, \ldots$.

Algorithm 4.9 (*LU* factorization)
Input: Invertible matrix $A \in \mathbb{R}^{n\times n}$.
Output: Factors L, U of LU factorization $A = LU$.

Set $L := I_n$.
for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $\ell_{ik} \leftarrow \dfrac{a_{ik}}{a_{kk}}$
    for $j = k+1, \ldots, n$ **do**
      $a_{ij} \leftarrow a_{ij} - \ell_{ik} a_{kj}$
    end for
  end for
end for
Set U to upper triangular part of A.

PYTHON

```
import numpy as np
def mylu(A):
    n = A.shape[0]
    L = np.eye(n)
    for k in range(n):
        L[k+1:n, k] = A[k+1:n, k] / A[k,k]
        A[k+1:n, k+1:n] = A[k+1:n, k+1:n]
                        - np.outer(L[k+1:n, k], A[k, k+1:n])
    U = np.triu(A)
    return L, U
```

The complexity of Algorithm 4.9 is given by

$$\sum_{k=1}^{n-1} \big(1 + 2(n-k)\big)(n-k) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n = \frac{2}{3}n^3 + O(n^2) \text{ flops.}$$

In summary, a linear system $A\mathbf{x} = \mathbf{b}$ is solved with the following procedure.

Algorithm 4.10 (Solution of $A\mathbf{x} = \mathbf{b}$ with LU factorization)

1. Compute LU factorization of $A = LU$ with Algorithm 4.9.
2. Solve $L\mathbf{y} = \mathbf{b}$ by forward substitution (Algorithm 4.3).
3. Solve $U\mathbf{x} = \mathbf{y}$ by backward substitution (Algorithm 4.4).

Example 4.11 We apply Algorithm 4.10 to Example 4.2:

```
import numpy as np, scipy as sp
A = np.array([[-0.370,  0.050,  0.050,  0.070],
```

```
              [ 0.050, -0.116,  0.000,  0.050],
              [ 0.050,  0.000, -0.116,  0.050],
              [ 0.070,  0.050,  0.050, -0.202]])
b = np.array([[-2], [0], [0], [0]])
L, U = mylu(A)
y = sp.linalg.solve_triangular(L, b, lower=True, unit_diagonal=True)
x = sp.linalg.solve_triangular(U, y)
```

This produces the output

```
x =
    8.1172
    5.9893
    5.9893
    5.7779
```

Of course, we could also have obtained the solution by directly calling `linalg.solve` from NumPy or SciPy. ◇

4.3 LU factorization with pivoting

Algorithm 4.9 will clearly fail whenever it encounters a zero pivot element. For example,

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

has a zero diagonal entry in the first position and immediately leads to a division by zero in Algorithm 4.9, although A is an invertible matrix. This situation is easy to spot, but it is important to keep in mind that – except for the first step – the pivot elements are computed in the course of the algorithm and it is in general impossible to "see" whether a matrix might lead to zero pivot elements.

The situation for the matrix A above is easy to resolve: We simply exchange rows 1 and 3 before attempting to compute the LU factorization. This corresponds to the multiplication of A with a permutation matrix[12]:

$$\widetilde{A} = PA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}, \quad \text{with} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

The LU factorization of $\widetilde{A}$ is then given by

$$\widetilde{A} = PA = LU \quad \text{with} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 2 \end{pmatrix}.$$

[12] A permutation matrix has exactly one entry 1 in each row and column, and is otherwise zero.

There is an even more important reason to perform such permutations. Consider a slight modification of A:

$$A = \begin{pmatrix} 10^{-16} & 1 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

Then Algorithm 4.9 does not fail and Python returns

```
L =
   1.0000e+00            0            0
            0   1.0000e+00            0
   1.0000e+16  -1.0000e+16   1.0000e+00
U =
   1.0000e-16   1.0000e+00   1.0000e+00
            0   1.0000e+00  -1.0000e+00
            0            0  -2.0000e+16
```

Moreover, solving the linear system for the right-hand side $b = [2,\ 2,\ 1]^T$ with these triangular factoriuation gives the "solution"

```
x =
     0
     2
     0
```

However, this is **totally wrong**; the exact solution is $x = [1, 2, 0]^T$ (up to roundoff error 10^{-16}). What happened? The very large entries in L and U gave rise to massive numerical cancellation, which eventually destroyed the numerical accuracy of the solution completely. The large entries are due to the very small pivot element 10^{-16} in the first step of the LU factorization. To avoid such small pivot elements, we perform **a row permutation in each step of the LU factorization such that the entry of largest magnitude in the active column below the diagonal becomes the pivot element.** The realization of this idea is illustrated by the following example.

Example 4.12 Let

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 2 & -7 & 2 \\ 1 & 24 & 0 \end{pmatrix}.$$

According to the idea above, we exchange the first and second rows by a permutation:

$$\widetilde{A}^{(0)} := P_1 A = \begin{pmatrix} 2 & -7 & 2 \\ 1 & 2 & 2 \\ 1 & 24 & 0 \end{pmatrix}, \quad \text{with} \quad P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We now apply the usual elimination step to $\widetilde{A}^{(0)}$, resulting in $\ell_{21} = 0.5$, $\ell_{31} = 0.5$, and

$$A^{(1)} = \begin{pmatrix} 2 & -7 & 2 \\ 0 & 5.5 & 1 \\ 0 & 27.5 & -1 \end{pmatrix}.$$

Since $a_{32}^{(1)}$ is larger than $a_{22}^{(1)}$, we have to exchange rows 2 and 3 before applying the next step of LU factorization:

$$\widetilde{A}^{(1)} := P_2 A^{(1)} = \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 5.5 & 1 \end{pmatrix}, \quad \text{with} \quad P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The usual elimination step applied to $\widetilde{A}^{(1)}$, results in $\ell_{32} = \frac{5.5}{27.5} = 0.2$, and

$$U = A^{(2)} = \begin{pmatrix} 2 & -7 & 2 \\ 0 & 27.5 & -1 \\ 0 & 0 & 1.2 \end{pmatrix},$$

Moreover,

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0.2 & 1 \end{pmatrix}.$$

It remains to collect the permutations:

$$P = P_2 P_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

This can and should be performed without explicit multiplication by applying the row exchange described by P_2 to the rows of P_1. In summary, we obtain the following **LU factorization with pivoting**:

$$PA = LU,$$

where P is a permutation matrix and L/U are lower/upper triangular matrices. $\diamond$

The extension of the procedure outlined in Example 4.12 to general matrices is given by the following algorithm.

Algorithm 4.13 (*LU* factorization with pivoting)
Input: Invertible matrix $A \in \mathbb{R}^{n\times n}$.
Output: Factors P, L, U of LU factorization with pivoting $PA = LU$.

for $k = 1, \dots, n-1$ **do**
  Search $i \in \{k, \dots, n\}$ such that $|a_{ik}| \geq |a_{i'k}|$ for all $i' \in \{k, \dots, n\}$
  Define P_k as permutation matrix that exchanges rows i and k.
  Exchange rows: $A \leftarrow P_k A$
  for $i = k+1, \dots, n$ **do**
    $a_{ik} \leftarrow \dfrac{a_{ik}}{a_{kk}}$
    for $j = k+1, \dots, n$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
    end for
  end for
end for
Set U to upper triangular part of A.
Set L to lower triangular part of A and diagonal entries 1.
Set $P := P_{n-1}P_{n-2}\cdots P_2 P_1$.

In PYTHON, Algorithm 4.13 is performed by the function

```
L,U,P = scipy.linalg.lu(A)
```

If implemented well, the cost of Algorithm 4.13 is essentially identical with Algorithm 4.9. An algorithm for solving a linear system $A\mathbf{x} = \mathbf{b}$ can be obtained from a slight modification of Algorithm 4.10:

1. Compute LU factorization with pivoting $PA = LU$ with Algorithm 4.13.
2. Set $\tilde{\mathbf{b}} := P\mathbf{b}$ and solve $L\mathbf{y} = \tilde{\mathbf{b}}$ by forward substitution (Algorithm 4.3).
3. Solve $U\mathbf{x} = \mathbf{y}$ by backward substitution (Algorithm 4.4).

4.4 Symmetric positive definite matrices⋆

Matrices from applications often have additional properties that facilitate the solution of the associated linear systems. A quite common property is positive definiteness.

Definition 4.14 *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called*

1. **positive definite**, *if* $\mathbf{x}^\mathsf{T} A\mathbf{x} > 0 \qquad \forall \mathbf{x} \in \mathbb{R}^n \backslash \{\mathbf{0}\}$;
2. **positive semi-definite**, *if* $\mathbf{x}^\mathsf{T} A\mathbf{x} \geq 0 \qquad \forall \mathbf{x} \in \mathbb{R}^n$.

A symmetric matrix is positive definite (semi-definite) if all its eigenvalues are positive (non-negative). The following theorem collects some simpler *necessary conditions for positive definiteness.*

Theorem 4.15 *Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then the following statements hold:*

1. *A is invertible;*
2. *$a_{ii} > 0$ for all $i \in \{1, \ldots, n\}$;*
3. *$|a_{ij}| < \frac{1}{2}(a_{ii} + a_{jj})$ for $i \neq j$ and hence $\max_{ij} |a_{ij}| = \max_i a_{ii}$;*
4. *if $X \in \mathbb{R}^{n\times n}$ is invertible, then $X^\mathsf{T}AX$ is again symmetric positive definite.*

Setting $A = I$, it follows from Theorem 4.15.4 that $X^\mathsf{T}X$ is symmetric positive for every invertible matrix X. The converse is also true: every symmetric positive matrix can be written in the form $X^\mathsf{T}X$.

Theorem 4.16 *Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then there exists an upper triangular matrix $R \in \mathbb{R}^{n\times n}$ such that $A = R^\mathsf{T}R$.*

The factorization $A = R^\mathsf{T}R$ from Theorem 4.16, with an upper triangular matrix R having positive diagonal entries, is called **Cholesky factorization**. Note that this is a special case of an LU factorization $A = LU$, with $L = R^\mathsf{T}$ and $U = R$. In fact, the Cholesky factorization could be extracted from an LU factorization. However, the following algorithm is more direct and more efficient.

Algorithm 4.17 (Cholesky factorization)
Input: Symmetric positive matrix $A \in \mathbb{R}^{n\times n}$.
Output: Cholesky factor R such that $A = R^\mathsf{T}R$.

for $j = 1, \ldots, n$ **do**
  for $i = 1, \ldots, j-1$ **do**
    $r_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki}r_{kj}\right)/r_{ii}$
  end for
  $r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2}$
end for

Algorithm 4.17 requires $n^3/3 + O(n^2)$ flops and is therefore half as expensive as computing the LU factorization. This is due to the fact that only one factor instead of two need to be computed.

Remark 4.18 *Algorithm 4.17 will break down if the matrix A is not positive definite. In this case, the expression $a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2$ will become negative at one point of the algorithm and no (real) square root can be determined.*

Note that Algorithm 4.17 does not perform any pivoting/permutations. It turns out that this is not necessary; the Cholesky factorization of a symmetric positive definite matrix can always be computed in a numerically reliable way without pivoting.

Recap on norms

Before we continue, we will recall basic definitions and facts about vector and matrix norms, which will be important for the rest of this chapter and next chapter. A (vector) norm on $\mathbb{R}^n$ is a real-valued function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ that satisfies the following three axioms:

Triangle inequality: $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$;

Homogeneity: $\|\alpha\mathbf{x}\| \leq |\alpha|\|x\|$ for all $\alpha \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$;

Positivity: $\|\mathbf{x}\| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$ and $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = 0$.

The most common examples are the Euclidean norm $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$, the 1-norm $\|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n|$, and the ∞-norm $\|\mathbf{x}\|_\infty = \max\{|x_i| : i = 1, \ldots, n\}$. On the other hand, the function $\|\mathbf{x}\|_0 = \#\{i : x_i \neq 0\}$, where $\#$ denotes the cardinality of a set, is *not* a norm.

A matrix norm on $\mathbb{R}^{m\times n}$ is a real-valued function $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ that satisfies the vector norm axioms on the vector space $\mathbb{R}^{m\times n} \cong \mathbb{R}^{m\cdot n}$:

Triangle inequality: $\|A+B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{R}^{m\times n}$;

Homogeneity: $\|\alpha A\| \leq |\alpha|\|A\|$ for all $\alpha \in \mathbb{R}$, $A \in \mathbb{R}^{m\times n}$;

Positivity: $\|A\| \geq 0$ for all $A \in \mathbb{R}^{m\times n}$ and $\|A\| = 0$ if and only if $A = 0$.

In order to be useful for the analysis of algorithms, a matrix norm should also be sub-multiplicative. More precisely, a family of matrix norms $\|\cdot\|$ on $\mathbb{R}^{m\times n}$, defined for all $m, n \in \mathbb{N}$, is called *sub-multiplicative* if

$$\|AB\| \leq \|A\|\|B\|, \quad \forall A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p},\ m, n, p \in \mathbb{N}.$$

The most simple way to define a matrix norm is to just take a vector norm on $\mathbb{R}^{m\times n} \cong \mathbb{R}^{m\cdot n}$. For example, the Euclidean norm gives rise to the Frobenius norm

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

One can show that the Frobenius norm is sub-multiplicative. In contrast, the maximum norm

$$\|A\|_{\max} = \max\{|a_{ij}| : i = 1, \ldots, n,\ j = 1, \ldots, m\}$$

is a matrix norm but it is *not* sub-multiplicative.

Another popular way to define a matrix norm on $\mathbb{R}^{m\times n}$ is the *operator norm* induced by a vector norm $\|\cdot\|$ on $\mathbb{R}^m, \mathbb{R}^n$:

$$\|A\| := \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \|\mathbf{x}\|=1}} \|A\mathbf{x}\|$$

One easily verifies the three matrix norm axioms. Additionally, any operator norm is sub-multiplicative because

$$\begin{aligned}\|AB\| &= \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|AB\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|AB\mathbf{x}\|}{\|\mathbf{x}\|}\frac{\|B\mathbf{x}\|}{\|B\mathbf{x}\|} = \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|AB\mathbf{x}\|}{\|B\mathbf{x}\|}\frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|}\\ &\leq \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|AB\mathbf{x}\|}{\|B\mathbf{x}\|} \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} \leq \sup_{\substack{\mathbf{y}\in\mathbb{R}^n\\ \mathbf{y}\neq 0}} \frac{\|A\mathbf{y}\|}{\|\mathbf{y}\|} \cdot \sup_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq 0}} \frac{\|B\mathbf{x}\|}{\|\mathbf{x}\|} = \|A\|\|B\|.\end{aligned}$$

The usefulness of an operator norm in practice also depends on how easy it is to determine the supremum. For the following three situations, the operator norm can be computed explicitly

Spectral norm: The operator norm induced by the Euclidean norm $\|\cdot\| \equiv \|\cdot\|_2$ is called spectral norm and satisfies $\|A\|_2 = \sigma_1(A)$, where $\sigma_1(\cdot)$ denotes the largest singular value of a matrix.

Maximum column sum: The operator norm induced by the 1-norm $\|\cdot\| \equiv \|\cdot\|_1$ can be shown to satisfy

$$\|A\|_1 = \max\Big\{\sum_{i=1}^{m} |a_{ij}| \colon j = 1,\ldots,n\Big\},$$

that is, the maximum 1-norm of the columns of A.

Maximum row sum: The operator norm induced by the ∞-norm $\|\cdot\| \equiv \|\cdot\|_\infty$ can be shown to satisfy

$$\|A\|_\infty = \max\Big\{\sum_{j=1}^{n} |a_{ij}| \colon i = 1,\ldots,m\Big\},$$

that is, the maximum 1-norm of the rows of A.

By definition, *any* operator norm satisfies $\|I_n\| = 1$, where I_n is the $n\times n$ identity matrix. This shows that the Frobenius norm cannot be expressed as an operator norm because $\|I_n\|_F = \sqrt{n}$.

4.5 Error analysis: Conditioning

Solving linear systems on the computer in finite precision arithmetic is subject to roundoff error. Already storing the matrix A and the vector $\mathbf{b}$ will usually cause

some error. To verify the accuracy of a *computed* (approximate) solution $\widehat{\mathbf{x}}$ one often computes the norm of the **residual** $\mathbf{r} = \mathbf{b} - A\widehat{\mathbf{x}}$. One major philosophy in numerical analysis is *backward error analysis*, to view a computed solution of a given linear system as the *exact solution of a perturbed linear system*. The norm of the residual corresponds to the minimal perturbation.

Theorem 4.19 *Let $\widehat{\mathbf{x}} \neq 0$ be an approximate solution of the linear system $A\mathbf{x} = \mathbf{b}$. Then*

$$\min\{\|\triangle A\|_2 : \ (A + \triangle A)\widehat{\mathbf{x}} = \mathbf{b}\} = \frac{\|\mathbf{r}\|_2}{\|\widehat{\mathbf{x}}\|_2},$$

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

Proof. The relation $(A + \triangle A)\widehat{\mathbf{x}} = \mathbf{b}$ yields

$$\|\triangle A\|_2\|\widehat{\mathbf{x}}\|_2 \geq \|\triangle A\widehat{\mathbf{x}}\|_2 = \|\mathbf{r}\|_2,$$

and hence $\min\{\|\triangle A\|_2 : \ (A + \triangle A)\widehat{\mathbf{x}} = \mathbf{b}\} \geq \|\mathbf{r}\|_2/\|\widehat{\mathbf{x}}\|_2$. To establish equality, we still need an admissible perturbation $\triangle_0 A$ with $\|\triangle_0 A\|_2 = \|\mathbf{r}\|_2/\|\widehat{\mathbf{x}}\|_2$. This is given by

$$\triangle_0 A = \frac{1}{\|\widehat{\mathbf{x}}\|_2^2}\mathbf{r}\,\widehat{\mathbf{x}}^{\mathsf{T}},$$

because $(A + \triangle_0 A)\widehat{\mathbf{x}} = A\widehat{\mathbf{x}} + \mathbf{r} = \mathbf{b}$. ∎

When storing the entries of A in double precision one causes an error of order $10^{-16}\|A\|_2$, which in turn leads to a residual norm $\|\mathbf{r}\|_2$ of order $10^{-16}\|A\|_2\|x\|_2$ even if the rest of the computations was carried out exactly. For this reason, the best we can reasonably hope from a numerical algorithm for solving linear system is that it achieves $\|r\|_2 \sim 10^{-16}\|A\|_2\|x\|_2$. Such algorithms are called *backward stable*. However, this does *not* necessarily imply a small error $\|\widehat{\mathbf{x}} - \mathbf{x}\|_2$ of the solution itself.

Example 4.20 Let $A = (a_{ij})$ with $a_{ij} = \dfrac{1}{i + j - 1}$ be the $n \times n$ **Hilbert matrix** and $b = (1, \ldots, 1)^{\mathsf{T}}$. We compute the solution of $A\mathbf{x} = \mathbf{b}$ with \ and compare it with the "exact" solution, which is computed in 100 decimal digits arithmetic, using the mpmath library for performing floating-point arithmetic with arbitrary precision.

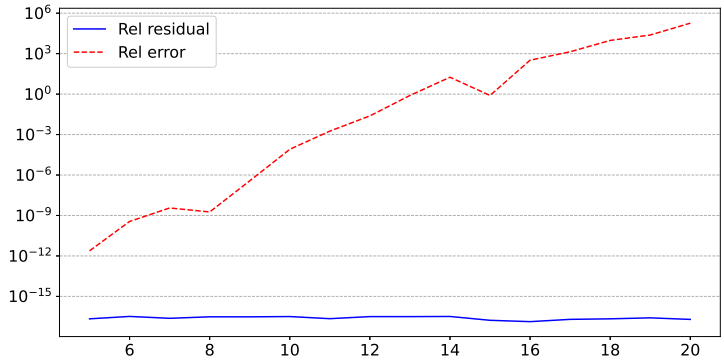
PYTHON

```
import matplotlib.pyplot as plt
import numpy as np, scipy.linalg as spla
from mpmath import hilbert, lu_solve, mp, ones

mp.dps = 100, ns = range(5, 21)
xrel, res = [], []
for n in ns:
    Anp, Amp = spla.hilbert(n), hilbert(n)
    rhsnp, rhsmp = np.ones(n), ones(n, 1)
    xnp, xmp = spla.solve(Anp, rhsnp), lu_solve(Amp, rhsmp)
    res.append(spla.norm(Anp @ xnp - rhsnp) / spla.norm(xnp))
```

```
    xrel.append(float(spla.norm(xnp - xmp)) / spla.norm(xnp))

plt.semilogy(ns, res, "b", ns, xrel, "r--")
plt.legend(["Rel residual", "Rel error"])
plt.grid(axis="y", which="major", linestyle="--")
plt.show()
```



The red dashed line shows the relative error of the solution $\widehat{\mathbf{x}}$ *computed in double precision* and the blue solid line shows the relative norm of the residual as n increases. The error of $\widehat{\mathbf{x}}$ grows rapidly, despite the fact that it is the solution of a very slightly perturbed linear system, according to Theorem 4.19. ◇

4.5.1 Condition of a function

To explain the phenomenon observed in Example 4.20 we will first consider the sensitivity analysis in a more abstract framework. A computation can be viewed as a function $f : \text{Input} \mapsto \text{Output}$. We assume that the admissible inputs are in a finite dimensional vector space V_i with norm $\|\cdot\|_{V_i}$. The Outputs are in a vector space V_o with norm $\|\cdot\|_{V_o}$. A computation is the evaluation $f(x) \in V_o$ for some admissible $x \in V_i$ (x can be a matrix; $V_i = \mathbb{R}^{n\times n}$). Because of roundoff error the representation of x is corrupted by an error $\triangle x$. We now want to understand how much this error is potentially increased when evaluating the function, that is, when considering $f(x+\triangle x)$ instead of $f(x)$. For this purpose we assume that f is defined and two times continuously differentiable in ab open ball containing $x \in V_i$. Using Taylor expansion, we have

$$x \mapsto f(x), \quad x + \triangle x \mapsto f(x + \triangle x) = f(x) + f'(x)\,\triangle x + O(\|\triangle x\|_{V_i}^2)$$

as $\triangle x \to 0$. The norm of the differential f' at x, that is,

$$\|f'(x)\|_{\mathcal{L}(V_i,V_o)} := \max\{\|f'(x)\,\triangle x\|_{V_o} : \|\triangle x\|_{V_i} = 1\}$$

measures the **absolute sensitivity** or *condition* of $f(x)$ with respect to a (small) perturbation $\triangle x$ of x.

Often, it is more reasonable to consider the **relative error**, which is independent of scaling of x or $f(x)$: For $f(x) \neq 0$, $x \neq 0$, $\triangle x \to 0$ we have

$$\frac{\|f(x+\triangle x)-f(x)\|_{V_o}}{\|f(x)\|_{V_o}} = \frac{\|f'(x)\triangle x\|_{V_o}+O(\|\triangle x\|_{V_i}^2)}{\|f(x)\|_{V_o}} \implies$$

$$\underbrace{\frac{\|f(x+\triangle x)-f(x)\|_{V_o}}{\|f(x)\|_{V_o}}}_{\text{rel. error in } f} \leq \Bigg(\frac{\|f'(x)\|_{\mathcal{L}(V_i,V_o)}}{\|f(x)\|_{V_o}}\|x\|_{V_i}\Bigg)\underbrace{\frac{\|\triangle x\|_{V_i}}{\|x\|_{V_i}}}_{\text{rel. error in } x} + O(\|\triangle x\|_{V_i}^2).$$

Definition 4.21 *The (relative)* **condition of the function** $x \mapsto f(x)$ *at* x *is*

$$\operatorname{cond}(f,x) := \frac{\|f'(x)\|_{\mathcal{L}(V_i,V_o)}}{\|f(x)\|_{V_o}}\|x\|_{V_i}\,.$$

As an example, consider the subtraction of two numbers $a, b \in \mathbb{R}$. Then $V_i = \mathbb{R}^2$ (with norm $\|\cdot\|_1$) and $V_o = \mathbb{R}$. Moreover, $f : V_i \to V_o$ with $f : (a,b) \mapsto a-b$ and

$$\|f'(a_0,b_0)\|_{\mathcal{L}(V_i,V_o)} = \left\|\left(\frac{\partial f}{\partial a}|_{(a_0,b_0)}, \frac{\partial f}{\partial b}|_{(a_0,b_0)}\right)\right\|_\infty = \|(1,-1)\|_\infty = 1.$$

Hence the relative condition of f at (a_0,b_0) is

$$\operatorname{cond}(f,(a_0,b_0)) = \frac{|a_0|+|b_0|}{|a_0-b_0|}.$$

As expected, this is large when $a_0 \approx b_0$.

4.5.2 Condition number of a matrix

In the following, $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ denotes the operator norm induced by a vector norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ (e.g., the spectral norm induced by the Euclidean norm). The following proposition studies the relative condition of matrix inversion, that is, the $f(X) = X^{-1}$ at an invertible matrix $X = A$.

Proposition 4.22 (Sensitivity of matrix inversion) *Let $A \in \mathbb{R}^{n\times n}$ be invertible. If $\|A^{-1}\triangle A\| < 1$ then $A+\triangle A$ is invertible and*

$$\underbrace{\frac{\|(A+\triangle A)^{-1}-A^{-1}\|}{\|A^{-1}\|}}_{\text{rel. error in } A^{-1}} \leq \frac{\|A\|\,\|A^{-1}\|}{1-\|A^{-1}\triangle A\|}\underbrace{\frac{\|\triangle A\|}{\|A\|}}_{\text{rel. error in } A}.$$

Proof. For $\|X\| < 1$, the so called **Neumann series**

$$\sum_{k=0}^{\infty} X^k = I + X + X^2 + \cdots = I + Y, \quad Y := \sum_{k=1}^{\infty} X^k \tag{4.9}$$

converges because $\| \sum_{k=0}^{\infty} X^k \| \leq \sum_{k=0}^{\infty} \|X\|^k = 1/(1 - \|X\|)$. This series is the inverse of $I - X$ because

$$(I - X)(I + Y) = (I - X)\Big(\sum_{k=0}^{\infty} X^k\Big) = \sum_{k=0}^{\infty} X^k - \sum_{k=1}^{\infty} X^k = I.$$

Moreover,

$$\|Y\| = \Big\|X \sum_{k=0}^{\infty} X^k\Big\| \leq \frac{\|X\|}{1 - \|X\|}.$$

Now we can write

$$A + \triangle A = A(I + A^{-1}\triangle A).$$

Setting $X = -A^{-1}\triangle A$, the discussion above shows that $(I + A^{-1}\triangle A)$ is invertible when $\|A^{-1}\triangle A\| < 1$ with the inverse given by $I + Y$ for some Y with $\|Y\| \leq \|A^{-1}\triangle A\|/(1 - \|A^{-1}\triangle A\|)$. In turn $A + \triangle A$ is also invertible and

$$\|(A+\triangle A)^{-1} - A^{-1}\| = \|(I+Y)A^{-1} - A^{-1} \leq \|Y\|\|A^{-1}\| \leq \|A^{-1}\|^2 \frac{\|\triangle A\|}{1 - \|A^{-1}\triangle A\|}.$$

□

Proposition 4.22 shows that $\|A\|\,\|A^{-1}\|$ governs the condition of matrix inversion. This quantity is called **condition number** of A:

$$\kappa(A) := \|A\|\,\|A^{-1}\|. \tag{4.10}$$

Note that the condition number depends on the operator norm $\|\cdot\|$. We write $\kappa_2(A) = \|A\|_2\,\|A^{-1}\|_2$.

PYTHON

```
from numpy import linalg as LA
LA.cond(A) # condition number of A in the spectral norm
LA.cond(A,p) # condition number of A in the p-norm with p = 1,2,inf
LA.cond(A,'fro') # condition number of A in the Frobenius norm
```

4.5.3 Sensitivity of linear systems

As a direct consequence of Proposition 4.22 we obtain the following result on the sensitivity of the solution of a linear system to perturbations in A and $\mathbf{b}$.

Theorem 4.23 *Let $A \in \mathbb{R}^{n\times n}$ be invertible and consider $\triangle A \in \mathbb{R}^{n\times n}$ satisfying $\|A^{-1}\triangle A\| < 1$. Then*

$$(A + \triangle A)\widehat{\mathbf{x}} = \mathbf{b} + \triangle\mathbf{b} \tag{4.11}$$

has a unique solution and

$$\frac{\|\widehat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)}{1 - \|A^{-1}\triangle A\|}\Big(\frac{\|\triangle A\|}{\|A\|} + \frac{\|\triangle\mathbf{b}\|}{\|\mathbf{b}\|}\Big). \tag{4.12}$$

Proof. Using the notation from the proof of Proposition 4.22 we have

$$\widehat{\mathbf{x}} = (A + \triangle A)^{-1}(\mathbf{b} + \triangle\mathbf{b}) = (I + Y)(\mathbf{x} + A^{-1}\triangle\mathbf{b}).$$

Hence, $\widehat{\mathbf{x}} - \mathbf{x} = Y\mathbf{x} + (I + Y)A^{-1}\triangle\mathbf{b}$ and, in turn,

$$\begin{aligned}
\|\widehat{\mathbf{x}} - \mathbf{x}\| &\leq \|Y\|\,\|\mathbf{x}\| + (1 + \|Y\|)\|A^{-1}\|\|\triangle\mathbf{b}\| \\
&\leq \frac{\|A^{-1}\triangle A\|}{1 - \|A^{-1}\triangle A\|}\|\mathbf{x}\| + \frac{1}{1 - \|A^{-1}\triangle A\|}\|A^{-1}\|\|\triangle\mathbf{b}\| \\
&\leq \frac{\kappa(A)}{1 - \|A^{-1}\triangle A\|}\Big(\frac{\|\triangle A\|}{\|A\|} + \frac{\|\triangle b\|}{\|A\|\|\mathbf{x}\|}\Big)\|\mathbf{x}\|
\end{aligned}$$

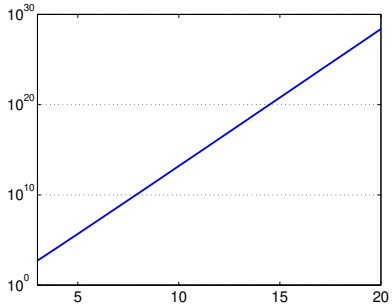
The proof is completed using $\|A\|\|\mathbf{x}\| \geq \|\mathbf{b}\|$. ∎

Remark 4.24 *The inequality* (4.12) *can be weakened to the asymptotic bound*

$$\frac{\|\widehat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\,\Big(\frac{\|\triangle A\|}{\|A\|} + \frac{\|\triangle\mathbf{b}\|}{\|\mathbf{b}\|}\Big) + O(\epsilon^2)$$

for $\epsilon = \max\{\|\triangle A\|, \|b\|\} \to 0$.

This analysis explains the strong growth of the error of $\widehat{\mathbf{x}}$ in Example 4.20. As n increases, the condition number of the Hilbert matrix increases exponentially fast.[13] The following figure shows the condition number for the spectral norm vs n.



[13] See Beckermann, B. The condition number of real Vandermonde, Krylov and positive definite Hankel matrices. Numer. Math. 85 (2000), no. 4, 553–577.

4.6 Error analysis: Solution by LU factorization

We have now understood the effect of error in the data (matrix A and vector b) on the quality of the solution. This effect is independent of any algorithm; in fact, no algorithm can reasonably undo the effect of rounding A and b. Therefore, the best we can hope for is that our algorithm does not introduce a much larger error. This leads to the idea of *backward error analysis*; relate the error conducted during the algorithm back to the original data and hope that it is not much larger than what happened due to rounding anyway.

In the following, we will provide the analysis of the forward/backward substitution step and only state the result for the solution via the LU decomposition. In the following lemma, $|A|$ will denote the matrix obtained from A by replacing each entry by its absolute value. Comparison between matrices is understood elementwise. Also, we recall the quantity $\gamma_n \equiv \gamma_n(\mathbb{F}) := \frac{nu(\mathbb{F})}{1-nu(\mathbb{F})} = nu(\mathbb{F}) + O(u(\mathbb{F})^2)$ defined in Lemma 1.16.

Lemma 4.25 *Let $\widehat{\mathbf{x}}$ denote the solution computed by Algorithm 4.3 in floating point arithmetic $\mathbb{F}$. Then there exists a matrix $\triangle L$ with*

$$|\triangle L| \leq \gamma_n |L|,$$

such that

$$(L + \triangle L)\widehat{\mathbf{x}} = \mathbf{b}.$$

Proof. To simplify the notation, we let θ_i denote an undetermined generic variable satisfying $|\theta_i| \leq \gamma_i$. Using the models (1.5) and (1.6) of floating point arithmetic, the first loop of Algorithm 4.3 satisfies

$$\ell_{11}(1 + \theta_1)\widehat{x}_1 = b_1,$$

and the second loop satisfies

$$\ell_{22}(1 + \theta_2)\widehat{x}_2 = b_2 - \ell_{21}\widehat{x}_1(1 + \theta_1) .$$

More generally, in the ith loop we have

$$\ell_{ii}(1 + \theta_2)\widehat{x}_i = b_i - \ell_{i1}\widehat{x}_1(1 + \theta_{i-1}) - \ell_{i2}\widehat{x}_2(1 + \theta_{i-2}) - \cdots - \ell_{i,i-1}\widehat{x}_{i-1}(1 + \theta_1) .$$

Setting $\triangle \ell_{ii} := \ell_{ii}\theta_2$ and $\triangle \ell_{ik} := \ell_{ik}\theta_{i-k}$ yields the result. ∎

An immediate consequence of the result of Lemma 4.25 is that

$$\|\triangle L\|_2 \leq c_L u \|L\|_2$$

for some constant c_L mildly growing with n. An analogous result holds, of course, for Algorithm 4.4. More importantly, this also holds for $Ax = b$; the solution $\hat{x}$

computed by the LU factorization (with or without pivoting) satisfies $(A+\triangle A)\hat{x} = b$ for some $\triangle A$ with

$$\|\triangle A\|_2 \leq c_A u \|L\|_2 \|U\|_2$$

for some constant c_A mildly growing with n; see [3]. It is important to observe that $\|L\|_2\|U\|_2$ can be significantly larger than $\|A\|$. When using pivoting (that is, Algorithm 4.13) then all entries of L are bounded by one in magnitude, while the norm of U is critically determined whether the algorithm encounters small pivot elements. Even with pivoting, there are examples for which the pivot elements can become very small (much smaller than $\|A^{-1}\|$) but this is a rare event; see [3] for more details.