Linear systems - Direct methods

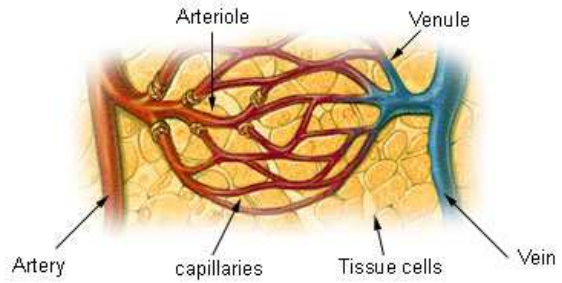
Numerical Analysis

Profs. Gianluigi Rozza - Luca Heltai

2018-SISSA mathLab Trieste

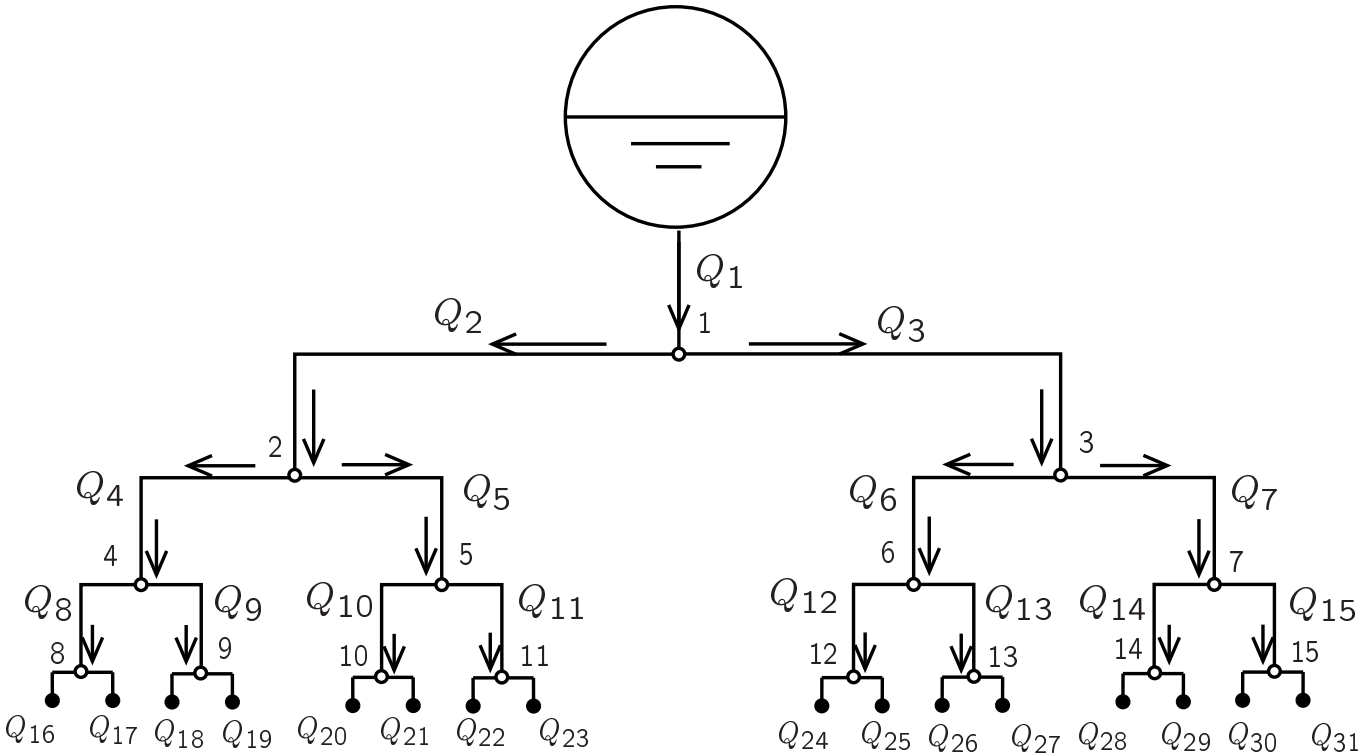
Examples and motivation

Example 1 (Capillary bed)**.** Capillaries are microscopic blood vessels, the smallest part of the circulatory system. The capillaries are grouped in networks called capillary beds, made of 10 to 100 capillaries, depending on the tissue.
Blood arrives to the capillary bed through arterioles. In the capillary bed an exchange of oxigen and waste molecules takes place.
Then, the capillaries become venules and collect the blood in veins to be transfered back to the heart.



We consider a model of capillary bed:

- We can model a certain portion of capillary system as a hydraulic network where every capillary is represented as a pipe.
- We call *nodes* (small empty circles in the figure on the next page) the points where several capillaries meet.
- The artery that feeds the system is represented as source at a constant pressure of 50mmHg.
- We suppose the blood leaves the system through the venules (small black circles in the figure) at a constant pressure (venous pressure), fixed to the reference value zero (all other pressures will be referenced to this one).
- The blood flows from the source to the sinks (we suppose in a constant way) due to the pressure gradient.


Q_1
1
Q_2
Q_3
2
3
Q_4
Q_5
Q_6
Q_7
4
5
6
7
Q_8
Q_9
Q_{10}
Q_{11}
Q_{12}
Q_{13}
Q_{14}
Q_{15}
8
9
10
11
12
13
14
15
Q_{16} Q_{17} Q_{18} Q_{19} Q_{20} Q_{21} Q_{22} Q_{23} Q_{24} Q_{25} Q_{26} Q_{27} Q_{28} Q_{29} Q_{30} Q_{31}

We want to find the pressure distribution p_j, $j = 1, \cdots, 15$, and the flows Q_m, $m = 1, \cdots, 31$, in the capillary network. To do this, we consider that:

- in every branch m of the network, $m = 1, \cdots, 31$, we have the following relation, called constitutive relation, between the blood flow Q_m (mm^3/s) and the pressure (in mmHg) in the two nodes i and j at the end-points of each capillary

$$Q_m = \frac{1}{R_m L_m}(p_i - p_j), \tag{1}$$

  where R_m is the hydraulic resistance per unit length ($\mathrm{mmHg\,s/mm}^4$) and L_m is the length of the capillary m;

- at every node j of the network, $j = 1, \cdots 15$, we impose the balance equation between inflow and outflow:

$$\left(\sum_{m \text{ entering}} Q_m \right) - \left(\sum_{m \text{ exiting}} Q_m \right) = 0,$$

  where outflows have a negative value.

For example, at the node 2 in the figure, the flow Q_2 is an inflow and the flows Q_4 and Q_5 are outflows, being the balance:

$$Q_2 - Q_4 - Q_5 = 0.$$

By using for every flow the constitution relation (1) in the previous balance, we have

$$\frac{1}{R_2 L_2}(p_1 - p_2) - \frac{1}{R_4 L_4}(p_2 - p_4) - \frac{1}{R_5 L_5}(p_2 - p_5) = 0.$$

One such equation is obtained for every node in the network.
Remark: if we consider the balance for the node 1, we get

$$\frac{1}{R_1 L_1}(p_r - p_1) - \frac{1}{R_2 L_2}(p_1 - p_2) - \frac{1}{R_3 L_3}(p_1 - p_3) = 0.$$

Being the pressure p_r constant, we move it to the right-hand side.

$$-\frac{1}{R_1 L_1}p_1 - \frac{1}{R_2 L_2}(p_1 - p_2) - \frac{1}{R_3 L_3}(p_1 - p_3) = -\frac{1}{R_1 L_1}p_r.$$

In a similar way for the node 8

$$\frac{1}{R_8 L_8}(p_4 - p_8) - \frac{1}{R_{16} L_{16}} p_8 - \frac{1}{R_{17} L_{17}} p_8 = 0,$$

where we see that $p_{16} = p_{17} = 0$. Same process for the nodes $9, \cdots, 15$.

Writing the balance for every node after having substituted the constitution relation, we get a system of linear equations for the pressures:

$$A\mathbf{p} = \mathbf{b}, \tag{2}$$

where $\mathbf{p} = [p_1, p_2, \cdots, p_{15}]^T$ is the unknown vector pressures at the nodes of the network, $A \in \mathbb{R}^{15\times 15}$ is the coeficient matrix of the system and $\mathbf{b} \in \mathbb{R}^{15}$ is the array of known data.

Lets suppose that all capillaries have the same hydraulic resistance $R_m = R = 1$ and that capillary length halves at each bifurcation (if $L_1 = 20$, we'll have $L_2 = L_3 = 10$, $L_4 = L_5 = L_6 = L_7 = 5$ etc..), the following matrix A is generated:

$$\begin{pmatrix}
-\frac{1}{4} & \frac{1}{10} & \frac{1}{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{10} & -\frac{1}{2} & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{10} & 0 & -\frac{1}{2} & 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{5} & 0 & -1 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{5} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{5} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.4 & 0 & 0 \\
0 & 0 & \frac{1}{5} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0.4 \\
0 & 0 & 0 & 0.4 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2
\end{pmatrix}$$

and $\mathbf{b} = [-5/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$.

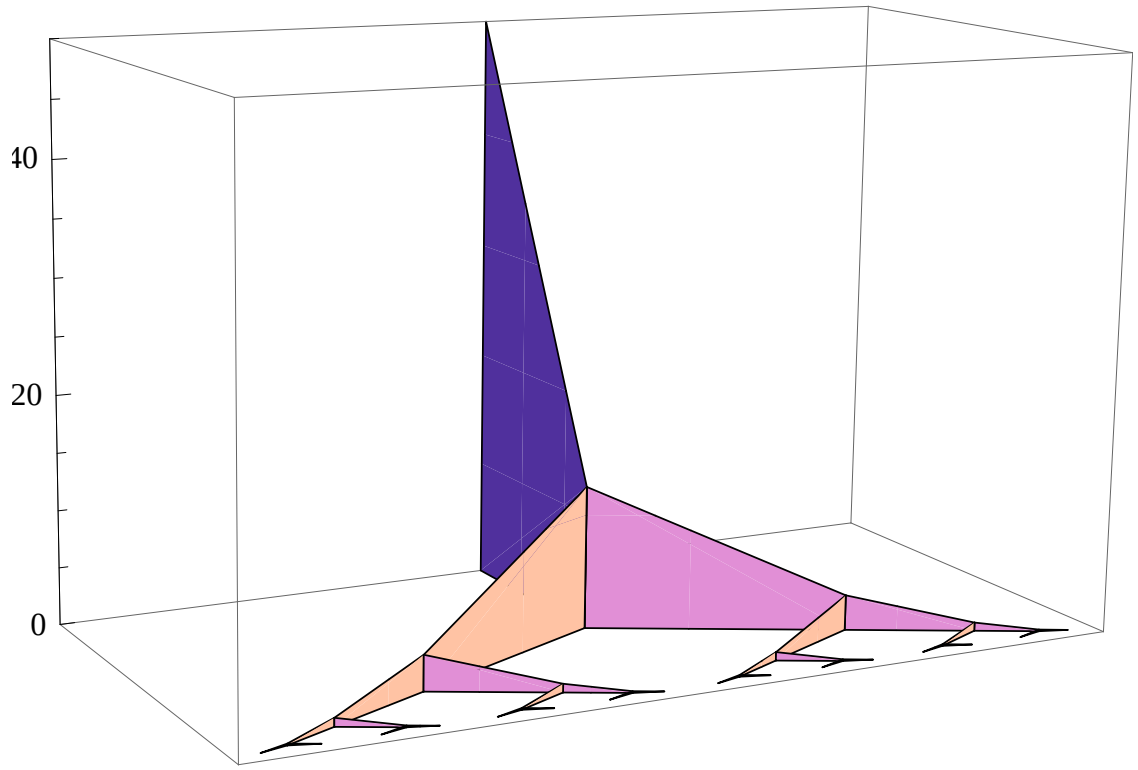
The problem of determining the pressures and flows in the network requires to solve a linear system $A\mathbf{p} = \mathbf{b}$.
In this case, the matrix A is symmetric and definite positive. This last property ensures that the matrix A is not singular and thus that the system has a unique solution.
The solution is given by the vector

$$\mathbf{p} = [12.46, 3.07, 3.07, 0.73, 0.73, 0.73, 0.73, 0.15, 0.15, 0.15, 0.15, 0.15, 0.15, 0.15, 0.15]^T.$$

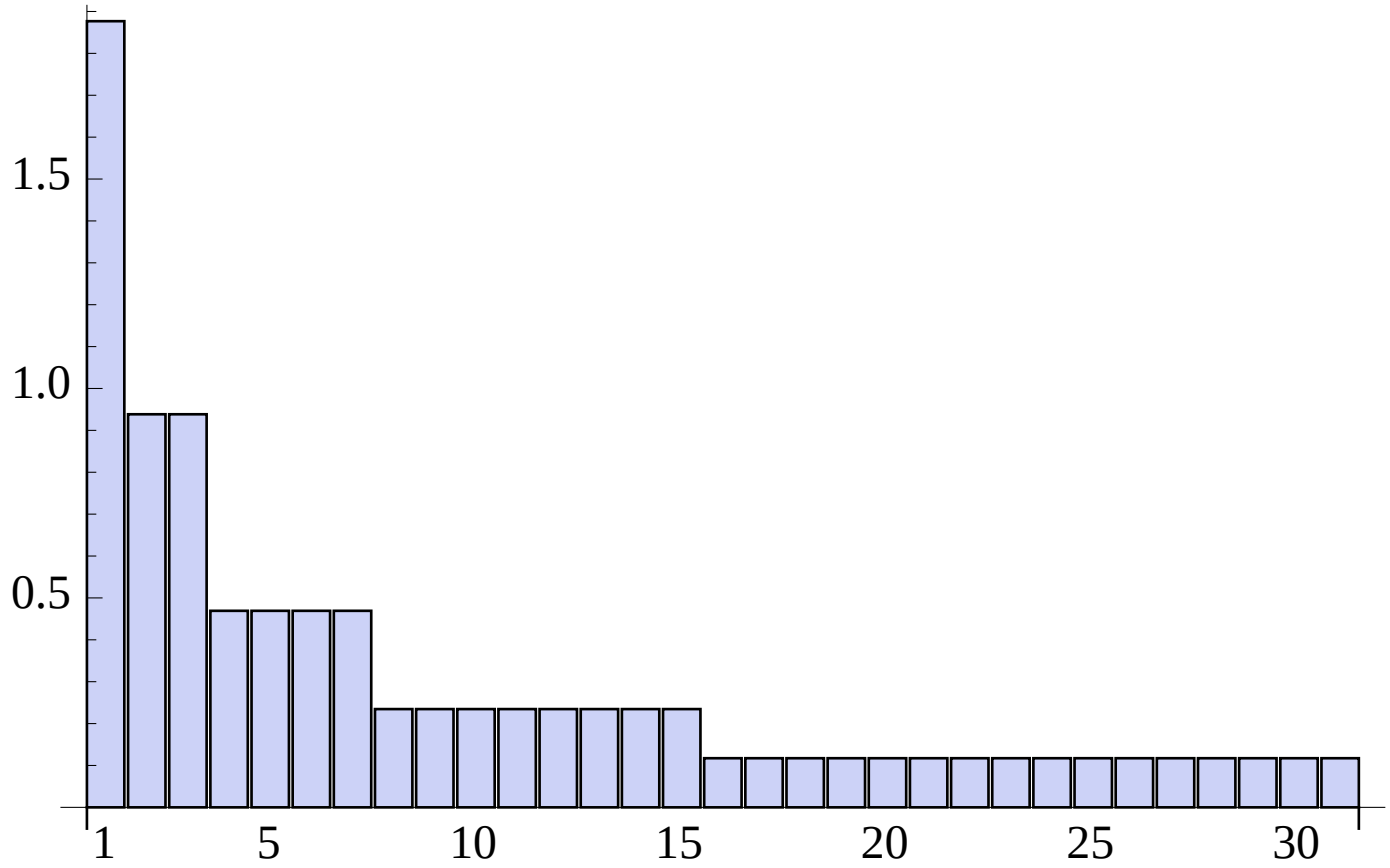
Once we have the pressures, we can compute, using the relation (1), the flows:

$$\begin{aligned} Q_1 &= 1.88 \\ Q_{2,3} &= 0.94 \\ Q_{4,\cdots,7} &= 0.47 \\ Q_{8,\cdots,15} &= 0.23 \\ Q_{16,\cdots,31} &= 0.12 \end{aligned}$$

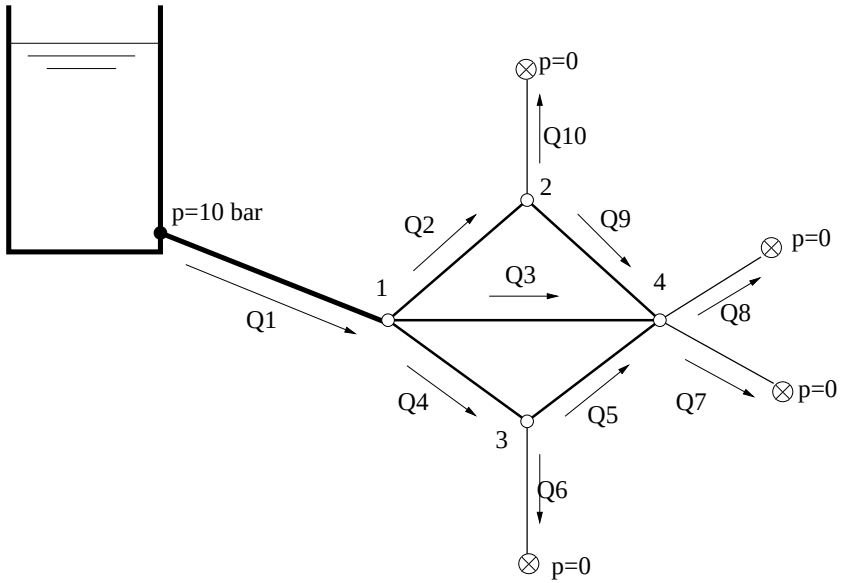
Linear distribution of pressures in the capillary bed computed from the pressures at every node (solution of the system)



Distribution of the flows in the capillary bed



Example 2 (Hydraulic network)**.** Let us consider a hydraulic network made of 10 pipes.



The network is fed by a reservoir of water at constant pressure $p_r = 10$ bar (pressure values in this exercise are the difference between the "real" pressure and the atmospheric pressure, in bar). For each pipe in the network, the following relation between the water flow Q (m^3/s) and the pressure (in bar) holds at both pipe-ends

$$Q = \frac{1}{RL}(p_{\text{in}} - p_{\text{out}}), \tag{3}$$

where R is the hydraulic resistance per unit length ((bar s)/m^4) and L is the legth (in m) of the pipeline. The resistance R depends on the pipe radius r and the dynamic viscosity μ of the liquid, following the relation $R = 8\mu/(\pi r^4)$.

At every node of the network, the balance between inflows and outflows can be set; for example for the node 2 in the figure, we have $Q_2 - Q_9 - Q_{10} = 0$ (outflows have a negative sign).

We assume atmospheric pressure at the outlets ($p = 0$ bar). A typical problem consists the pressure values and flows in the network. Using both given relations, a linear system can be built for the pressure of a form:

$$A\mathbf{p} = \mathbf{b}, \tag{4}$$

where $\mathbf{p} = [p_1, p_2, p_3, p_4]^T$ is the vector of pressures (unknown) at the 4 nodes of the network.

The following table contains the relevant characteristics of the pipelines:

Pipe	R	L	*Pipe*	R	L	*Pipe*	R	L
1	0.2500	20	2	2.0000	10	3	1.0204	14
4	2.0000	10	5	2.0000	10	6	7.8125	8
7	7.8125	8	8	7.8125	8	9	2.0000	10
10	7.8125	8						

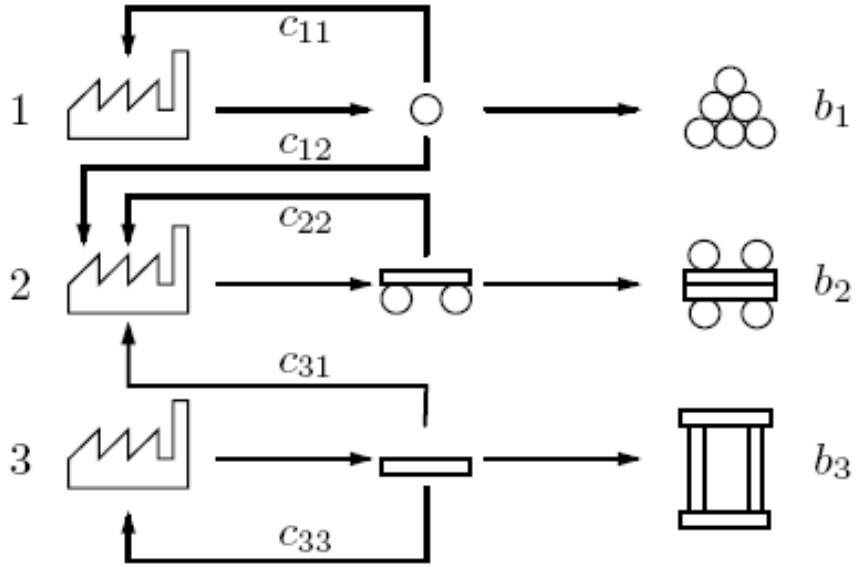
Correspondingly, A and $\mathbf{b}$ are:

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

The array of pressure values at the nodes is:
$\mathbf{p} = A^{-1}\mathbf{b} = [8.1172, 5.9893, 5.9893, 5.7779]^T$. We can finally compute, using the relation (1), the flows (in m^3/s):

Q1	0.3766	Q2	1.0640	Q3	0.1638
Q4	0.1064	Q5	0.0106	Q6	0.0958
Q7	0.0924	Q8	0.0924	Q9	0.0106
Q10	0.0958				

Example 3 (Economy/Logistic)**.** We want to determine the situation of equilibrium between demand and offer of certain goods. Let us consider that $m >= n$ factories produce n different products. They have to adapt their productions to the internal demand (i.e. the goods needed as input by the other factories) as well as to the external demand, from the consumers.
$x_i,\ i = 1, \ldots, n$ is the total number of goods made by the factory i,
$b_i,\ i = 1, \ldots, n$ is the corresponding demand from the market and
c_{ij} the amount produced by the factory i needed for the factory j to make one unit of product.



Interaction scheme between 3 factories and the market

If we suppose that the relation between the different products is linear, the equlibrium is reached when the vector $\mathbf{x} = [x_1, \ldots, x_n]^T$ satisfies

$$\mathbf{x} = C\mathbf{x} + \mathbf{b},$$

where $C = (c_{ij})$ and $\mathbf{b} = [b_1, \ldots, b_n]^T$. Consequently, the total production $\mathbf{x}$ is solution of the linear system :

$$A\mathbf{x} = \mathbf{b}, \quad \text{where } A = I - C.$$

Formulation of the problem

We call linear system of order n (n positive integer), an expression of the form

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

where $A = (a_{ij})$ is a given matrix of size $n \times n$, $\mathbf{b} = (b_j)$ is a given vector and $\mathbf{x} = (x_j)$ is the unknown vector of the system. The previous relation is quivalent to the n equations

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \ i = 1, \ldots, n.$$

The matrix A is called *non-singular* if $\det(A) \neq 0$; the solution $\mathbf{x}$ will be unique (for any given vector $\mathbf{b}$) if and only if the matrix associated to the linear system is non-singular.

In theory, if A is non-singular, the solution is given by the *Cramer's rule*:

$$x_i = \frac{\det(B_i)}{\det(A)}, \ i = 1, \ldots, n,$$

where B_i is the matrix by substituting the i-th column of A by the vector $\mathbf{b}$:

$$B_i = \begin{bmatrix} a_{11} & \ldots & b_1 & \ldots & a_{1n} \\ a_{21} & \ldots & b_2 & \ldots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \ldots & b_n & \ldots & a_{nn} \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \uparrow \\ i \end{matrix}$$

Unfortunately, the application of this rule is unacceptable for the practical solution of systems because the computational cost is of the order of $(n+1)!$ *floating point operations* (*flops*). In fact, every determinant requires $n!$ *flops*.

For example, the following table gives the time required by different computers to solve a linear system using the Cramer rule (o.r. stands for “out of reach”):

	Number of flops of the computer				
n	10^9 (Giga)	10^{10}	10^{11}	10^{12} (Tera)	10^{15} (Peta)
10	10^{-1} sec	10^{-2} sec	10^{-3} sec	10^{-4} sec	negligible
15	17 hours	1.74 hours	10.46 min	1 min	$6\ 10^{-2}$ sec
20	4860 years	486 years	48.6 years	4.86 years	1.7 days
25	o.r.	o.r.	o.r.	o.r.	38365 years

Alternative algorithms have to be developed. In the following sections, several methods will be analysed.

Triangular systems

A matrix $U = (u_{ij})$ is *upper triangular* if

$$u_{ij} = 0 \quad \forall i, j \ : \ 1 \leq j < i \leq n$$

and a matrix $L = (l_{ij})$ is *lower triangular* if

$$l_{ij} = 0 \quad \forall i, j \ : \ 1 \leq i < j \leq n.$$

Respectively, the system to be solved is called *upper or lower triangular system*.
Remark: If a matrix A in non-singular and triangular, knowing that

$$\det(A) = \prod_{i=1}^{n} \lambda_i(A) = \prod_{i=1}^{n} a_{ii}$$

($\lambda_i(A)$ being the i-th eigenvalue of A), we can deduce that $a_{ii} \neq 0$, for all $i = 1, \ldots, n$.

If L is lower triangular and non-singular, the linear system $L\mathbf{y} = \mathbf{b}$ corresponds to

$$\begin{cases} l_{11}y_1 & = & b_1 \\ l_{21}y_1 + l_{22}y_2 & = & b_2 \\ \vdots & & \\ l_{n1}y_1 + l_{n2}y_2 + \ldots + l_{nn}y_n & = & b_n \end{cases}$$

Thus:

$$\boxed{y_1 = b_1/l_{11}}, \qquad \text{[1 operation]}$$

and for $i = 2, 3, \ldots, n$

$$\boxed{y_i = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right).} \qquad \text{[} 1 + 2(i-1) \text{ operations]}$$

This algorithm is called *forward substitutions algorithm.*

The forward substitutions algorithm requires n^2 operations, whare n is the size of the system:

$$\begin{aligned} 1 + \sum_{i=2}^{n} (1 + 2(i-1)) &= 1 + (n-1) + 2 \sum_{i=2}^{n} (i-1) \\ &= 1 + (n-1) + 2 \frac{n(n-1)}{2} \\ &= n^2. \end{aligned}$$

If U is upper triangular and non-singular, the system $U\mathbf{x} = \mathbf{y}$ is:

$$\begin{cases} u_{11}x_1 + \ldots + u_{1,n-1}x_{n-1} + u_{1n}x_n &= y_1 \\ \vdots & \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1} \\ u_{nn}x_n &= y_n \end{cases}$$

Thus:

$$\boxed{x_n = y_n/u_{nn}},$$

and for $i = n-1, n-2, \ldots, 2, 1$

$$\boxed{x_i = \frac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right).}$$

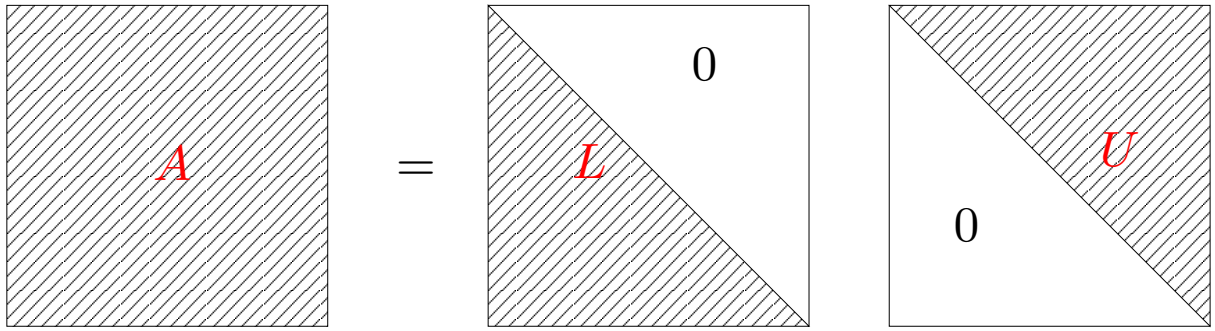
This algorithm is called *backward substitutions algorithm* The cost is, once again, n^2 operations.

The LU factorization method

(Sec. 5.3 of the book)

Let $A = (a_{ij})$ be a non-singular $n \times n$ matrix. Assume that there exist a matrix $U = (u_{ij})$, upper triangular and a matrix $L = (l_{ij})$, lower triangular such that

$$A = LU. \tag{5}$$



We call (5) a *factorization LU of A*.

If we know the factorization LU of A, solving the system $A\mathbf{x} = \mathbf{b}$ is equivalent to solving two systems defined by *triangular* matrices. Indeed,

$$A\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad LU\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \begin{cases} L\mathbf{y} = \mathbf{b}, \\ U\mathbf{x} = \mathbf{y}. \end{cases}$$

We can easily calculate the solutions of both systems:

- first, we use the forward substitutions algorithm to solve $L\mathbf{y} = \mathbf{b}$ (order n^2 *flops*);
- then, we use the backward substitutions algorithm to solve $U\mathbf{x} = \mathbf{y}$ (order n^2 *flops*).

It is required to find first (if possible) the matrices L and U (what requires a number of operations of the order $\frac{2n^3}{3}$ *flops*).

Example 4. Lets try to find a factorization LU in the case case where the size of the matrix A is $n = 2$. We can write the equation (5) as

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix},$$

Or equivalently:

$$\begin{aligned} &(a) \quad l_{11}u_{11} = a_{11}, \quad &&(b) \quad l_{11}u_{12} = a_{12}, \\ &(c) \quad l_{21}u_{11} = a_{21}, \quad &&(d) \quad l_{21}u_{12} + l_{22}u_{22} = a_{22}. \end{aligned}$$

We have then a system (non-linear) with 4 equations and 6 unknowns; in order to have the same number of equations and unknowns, we fix the diagonal of L by taking $l_{11} = l_{22} = 1$. Consequently, from (a) and (b) we have $u_{11} = a_{11}$ and $u_{12} = a_{12}$; finally, if we assume $a_{11} \neq 0$, we obtain $l_{21} = a_{21}/a_{11}$ and $u_{22} = a_{22} - l_{21}u_{12} = a_{22} - a_{21}a_{12}/a_{11}$ using the equations (c) and (d).

To determine a factorization LU of the matrix A of any size n, we apply the following method.

1. The elements of L and U satisfy the non-linear system

$$\sum_{r=1}^{\min(i,j)} l_{ir}u_{rj} = a_{ij}, \quad i,j = 1,\ldots,n; \tag{6}$$

2. The system (6) has n^2 equations and n^2+n unknowns. We can wipe out n unknowns if we set the n diagonal elements of L equal to 1:

$$l_{ii} = 1, \quad i = 1,\ldots,n.$$

We will see that in this case there exists an algorithm (*Gauss factorization*) allowing us to efficiently compute the factors L and U.

The Gauss elimination method

The Gauss elimination method transforms the system

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

in an equivalent system (i.e. with the same solution) of the form:

$$U\mathbf{x} = \widehat{\mathbf{b}},$$

where U is an upper triangular matrix and $\widehat{\mathbf{b}}$ is a properly modified second member.

This system can be solved by a backward substitutions method.

In the transformation, we essentially use the property that says that we do not change the solution of the system if we add to a given equation a linear combination of other equations.

Let us consider an invertible matrix $A \in \mathbb{R}^{n\times n}$ in which the diagonal element a_{11} is assumed to be non-zero. we set $A^{(1)} = A$ and $\mathbf{b}^{(1)} = \mathbf{b}$. We introduce the *multiplier*

$$l_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, 3, \ldots, n, \qquad A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & \ldots & a_{1j}^{(1)} & \ldots & a_{1n}^{(1)} \\ \vdots & & \vdots & & \vdots \\ a_{i1}^{(1)} & \ldots & a_{ij}^{(1)} & \ldots & a_{in}^{(1)} \\ \vdots & & \vdots & & \vdots \\ a_{n1}^{(1)} & \ldots & a_{nj}^{(1)} & \ldots & a_{nn}^{(1)} \end{bmatrix}$$

where the $a_{ij}^{(1)}$ represent the elements of $A^{(1)}$. Example:

$$A = \begin{bmatrix} 2 & 4 & 6 \\ 4 & 8 & 10 \\ 7 & 8 & 9 \end{bmatrix} \quad \Longrightarrow \quad l_{21} = \frac{4}{2}, l_{31} = \frac{7}{2}.$$

The unknown x_1 can be removed from the rows $i = 2, \ldots, n$ by substracting l_{i1} times the first row and doing the same at the right-hand side.

Let us define

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

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

where the $b_i^{(1)}$ are the components of $\mathbf{b}^{(1)}$ and we get a new system of the form

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix},$$

which will be written as $A^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$ and that is equivalent to the system we had at the beginning.

Once again we can transform this system by removing the unknown x_2 from the rows $3, \ldots, n$. By repeating this step we obtain a finite series of systems

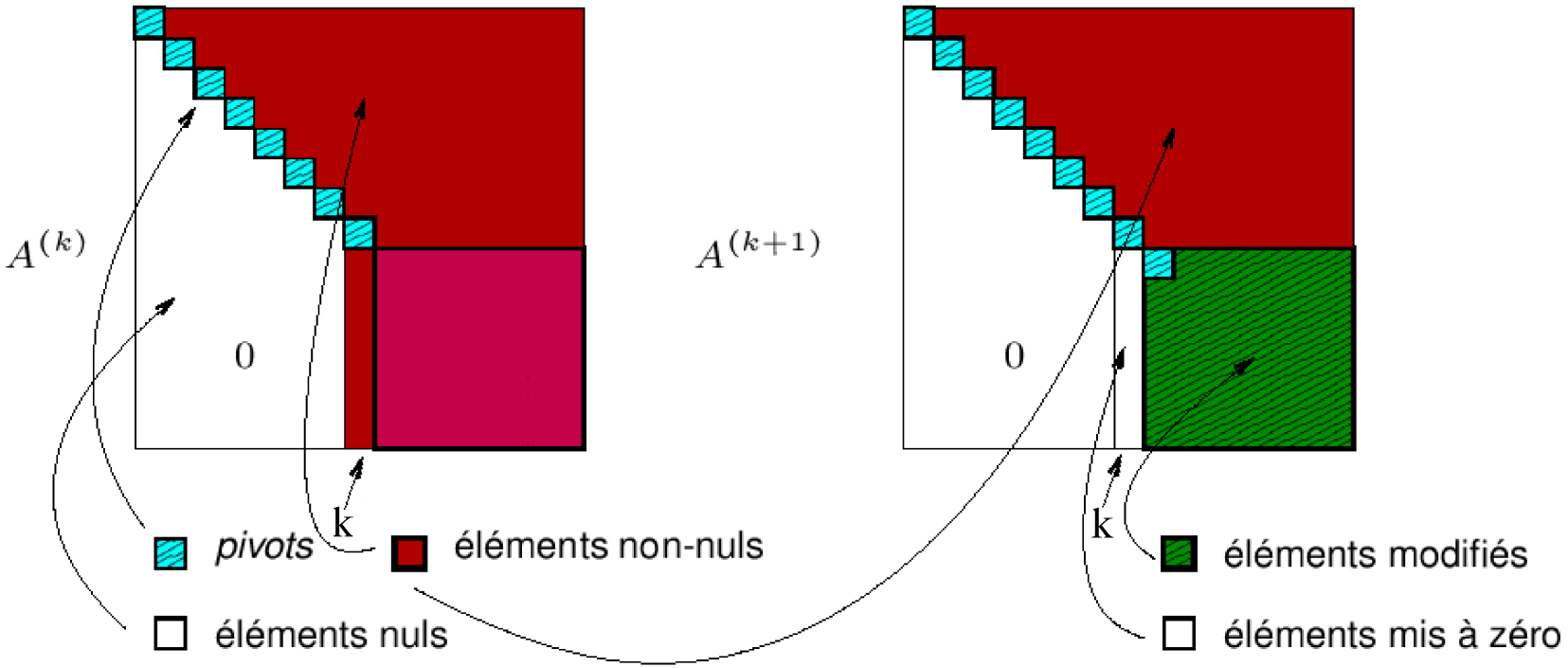
$$A^{(k)}\mathbf{x} = \mathbf{b}^{(k)}, \quad 1 \leq k \leq n, \tag{7}$$

where, for $k \geq 2$, the matrix $A^{(k)}$ is of the form

$$A^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & \ldots & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \ldots & 0 & a_{kk}^{(k)} & \ldots & a_{kn}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \ldots & 0 & a_{nk}^{(k)} & \ldots & a_{nn}^{(k)} \end{bmatrix},$$

where we assume $a_{ii}^{(i)} \neq 0$ for $i = 1, \ldots, k-1$.

Gauss elimination method: diagram showing how the matrix $A^{(k+1)}$ is obtained from the matrix $A^{(k)}$.



It is clear that for $k = n$ we obtain the following upper triangular system $A^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ 0 & & & & \vdots \\ 0 & & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}.$$

To be consistent with the previous notation, we write as U upper triangular matrix $A^{(n)}$. The elements $a_{kk}^{(k)}$ are called $pivots$ and have to be non-zero for $k = 1, \dots, n-1$.

In order to make explicit the formulae to get from the k-th systm to the $k+1$-th, for $k = 1, \ldots, n-1$, we asssume that $a_{kk}^{(k)} \neq 0$ and we define the multiplier

$$\boxed{l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, n,} \qquad [(n-k) \text{ operations}] \qquad (8)$$

we set then

$$\boxed{a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}, \quad i, j = k+1, \ldots, n,} \qquad [2(n-k)^2 \text{ operations}]$$

$$\boxed{b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}, \quad i = k+1, \ldots, n.} \qquad [2(n-k) \text{ operations}] \qquad (9)$$

Remark 1. *To perform the Gauss elimination,*

$$2\sum_{k=1}^{n-1}(n-k)^2 + 3\sum_{k=1}^{n-1}(n-k) \quad =$$

$$2\sum_{p=1}^{n-1}p^2 + 3\sum_{p=1}^{n-1}p \quad = \quad 2\frac{(n-1)n(2n-1)}{6} + 3\frac{n(n-1)}{2}$$

operations are required, plus n^2 *operations for the resolution with the backward substitutions method of the triangular system* $U\,\mathbf{x} = \mathbf{b}^{(n)}$. *By keeping only the dominant elements (of order* n^3*), we can say that the Gauss elimination method has a cost of around*

$$\frac{2}{3}n^3 \text{ operations.}$$

The following table shows the estimated computation time to solve a system using the LU factorization in different computers:

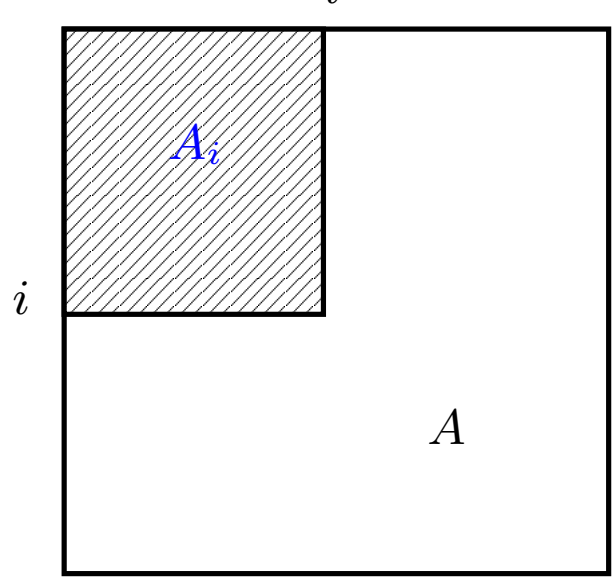
	Number of flops of the computer		
n	10^9 (Giga)	10^{12} (Tera)	10^{15} (Peta)
10^2	$7\ 10^{-4}$ sec	negligible	negligible
10^4	11 min	0.7 sec	$7\ 10^{-4}$ sec
10^6	21 years	7.7 months	11 min
10^8	o.r.	o.r.	21 years

The Gauss method is only properly defined if the pivots $a_{kk}^{(k)}$ are non-zero for $k = 1, \ldots, n-1$. Unfortunately, knowing that the diagonal elements of A are not zero is not enough to avoid null pivots during the elimination phase. For example, the matrix A in (10) is invertible and its diagonal elements are non-zero

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix}, \text{ but we find } A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & \boxed{0} & -1 \\ 0 & -6 & -12 \end{bmatrix}. \qquad (10)$$

Nevertheless, we have to stop the Gauss method at the second step, because $a_{22}^{(2)} = 0$.

Let A_i be the i-th main submatrix of A ($i = 1, \ldots, n-1$), i.e. the submatrix made of the i first rows and columns of A:



and let d_i be the principal minor of A defined as $d_i = det(A_i)$. We have the following result.

Proposition 1. *(Proposition* 5.1 *in the book) For a given matrix* $A \in \mathbb{R}^{n\times n}$, *its Gauss factorization exists and is unique iff the principal submatrices* A_i $(i = 1, \ldots, n-1)$ *are non-singular (i.e. the principal minors* d_i *are non-zero:* $d_i \neq 0$*).*

Remark: If $d_i \neq 0$ $(i = 1, \ldots, n-1)$, then the pivots $a_{ii}^{(i)}$ are also non-zero.

The matrix of the previous example does not satisfy this condition because $d_1 = 1$ but $d_2 = 0$.

There are some categories of matrices for which the hypothesis of the proposition (1) are fulfilled. In particular, we mention:

1. *(Strictly >) diagonal dominant by row* matrices. A matrix A is said diagonal dominant by row if

$$|a_{ii}| \geq \sum_{j=1,\ldots,n;j\neq i} |a_{ij}|, \quad i = 1,\ldots,n.$$

2. *(Strictly >) diagonal dominant by column* matrices. A matrix A is said diagonal dominant by column if

$$|a_{jj}| \geq \sum_{i=1,\ldots,n;i\neq j} |a_{ij}|, \quad j = 1,\ldots,n.$$

Examples: $\begin{bmatrix} -4 & 1 & 2 \\ 2 & 5 & 0 \\ -2 & 1 & 7 \end{bmatrix}$ is diagonal dominant by row and by column, whereas

$\begin{bmatrix} -3 & 1 & 2 \\ 2 & 5 & 0 \\ -2 & 1 & 7 \end{bmatrix}$ is only diagonal dominant by row.

3. *Symmetric definite positive* matrices. A matrix A is symmetric if $A = A^T$; it is definite positive if all its eigenvalues are positive, i.e.:

$$\lambda_i(A) > 0, \quad i = 1, \ldots, n.$$

Gauss $\sim LU$

We can show that the Gauss method is equivalent to the factorization $A = LU$ of the matrix A, with $L =$ multiplier matrix and $U = A^{(n)}$.

More exactly:

$$A = \underbrace{\begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ l_{21} & 1 & & & 0 \\ \vdots & l_{32} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{n1} & & & l_{n,n-1} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & & & \ddots & \vdots \\ 0 & & & & a_{nn}^{(n)} \end{bmatrix}}_{U}.$$

The matrices L and U only depend on A (and not on $\mathbf{b}$), the same factorization can be resused for solving several linear systems that share the same matrix A but different vectors $\mathbf{b}$.

The number of operations is then considerably reduced, since most of the computational weight, around $\frac{2}{3}n^3 flops$, is due to the Gaussian elimination process. Indeed, let us consider the M linear systems:

$$A\mathbf{x}_m = \mathbf{b}_m \qquad m = 1, \ldots, M$$

Donc:

- the cost of the factorization $A = LU$ is $\frac{2}{3}n^3 flops$;
- the cost of the resolution of both triangular systems, $L\mathbf{y}_m = \mathbf{b}_m$ and $U\mathbf{x}_m = \mathbf{y}_m$ $(m = 1, \ldots, M)$ is $2Mn^2 flops$,

for a total of $\frac{2}{3}n^3 + 2Mn^2 flops$ which is much smaller than $\frac{2}{3}Mn^3 flops$ required to solve all the systems Gauss elimination method.

The pivoting technique

It has been already noted that the Gauss method fails if a pivot becomes zero.

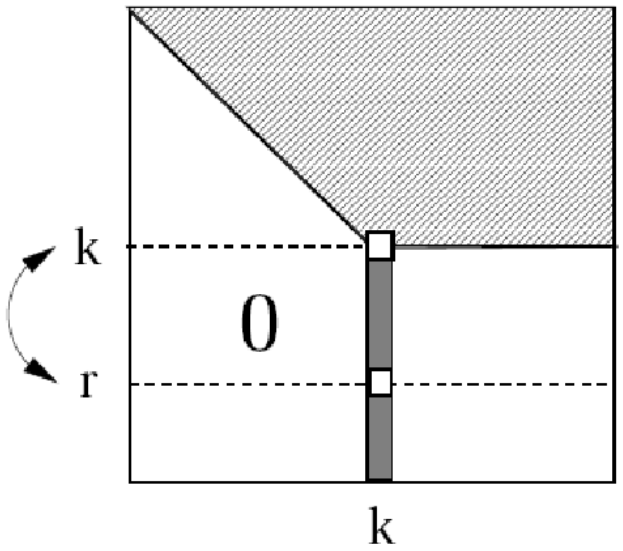
In that case, we can use a technique called pivoting that consists in exchanging the rows (or the columns) of the system in such a way that no pivot is zero.

Example 5. Let us go back to the matrix (10) for which the Gauss method gives a null pivot at the second step. By just exchanging the second and the third rows, we get a non-zero pivot and can execute one step further. Indeed,

$$A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & \boxed{0} & -1 \\ 0 & -6 & -12 \end{bmatrix} \quad \Longrightarrow \quad P_2 A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & \boxed{-6} & -12 \\ 0 & 0 & -1 \end{bmatrix},$$

where $P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ is called permutation matrix.

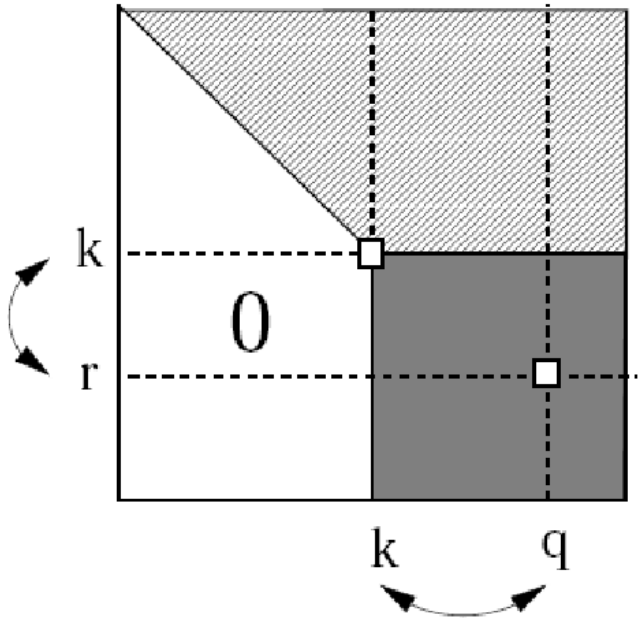
The pivoting strategy used for the example 5 can be generalized by finding, at every step k of the elimination, a non-zero pivot among the elements of the subcolumn $A^{(k)}(k:n,k)$. This is called a $partial\ pivot\ change$ (by row).



From (8) we know that a big value of l_{ik} (coming for instance from a small $a_{kk}^{(k)}$) can amplify the rounding errors affecting the elements $a_{kj}^{(k)}$.

Consequently, in order to ensure a better stability, we choose as pivot the biggest element in module of the column $A^{(k)}(k:n,k)$, and the partial pivoting is performed at every step, even if it is not strictly necessary (i.e. even if the pivot is non-zero).

An alternative method consists looking for the pivot in the whole submatrix $A^{(k)}(k:n, k:n)$, performing what is called *complete pivoting*.



Remark that, whereas partial pivoting requires just an additional cost of n^2 tests, complete pivoting needs some $2n^3/3$, what considerably increases the cost of the Gauss method.

In general, if at the step k we have to exchange the rows k and r, we will have to multiply $A^{(k)}$ by the following permutation matrix P_k before continuing:

$$
\begin{array}{c}
\\
k \rightarrow \\
\\
r \rightarrow \\
\\
\\
\\
\\
\end{array}
\begin{pmatrix}
1 & & & & & & \\
& \ddots & & & & & \\
& & 0 & \dots & 1 & & \\
& & & \vdots & & & \\
& & 1 & \dots & 0 & & \\
& & & & & \ddots & \\
& & & & & & 1
\end{pmatrix}
= P_k
$$

$$
\begin{array}{cc}
\uparrow & \uparrow \\
r & k
\end{array}
$$

This means we will consider $P_k A^{(k)}$ insted of $A^{(k)}$.

We can prove that the result obtained is of the form:

$$\boxed{PA = LU}, \tag{11}$$

being $P = P_{n-1}P_{n-2}\dots P_2P_1$ (global permutation matrix). L is the multiplier matrix (the new ones!) and $U = A^{(n)}$.

Once the matrices L, U and P have been calculated, the resolution of the initial system is transformed into the resolution of the triangular systems

$$A\mathbf{x} = \mathbf{b} \implies PA\mathbf{x} = P\mathbf{b} \implies \begin{cases} L\mathbf{y} = P\mathbf{b}, \\ U\mathbf{x} = \mathbf{y}. \end{cases}$$

Remark that the coefficients of the matrix L have the same values as the multipliers calculated by a factorization LU of the matrix PA without pivoting.

If we use complete pivoting, it can be proved that we arrive to the following result:

$$\boxed{PAQ = LU}$$

where $P = P_{n-1} \dots P_1$ is a permutation matrix that takes into account all permutations by row, and $Q = Q_1 \dots Q_{n-1}$ is a permutation matrix that takes into account all permutations by column. By construction, la matrix L is still lower triangular, and its elements have a module lower or equal to 1.

As for the partial pivoting, the elements of L are the multipliers generated by the factorization LU of the matrix PAQ with no pivoting.

Once the matrices L, U, P and Q have been calculated, for solving the linear system we notice that we can write

$$A\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \underbrace{PAQ}_{LU}\underbrace{Q^{-1}\mathbf{x}}_{\mathbf{x}^*} = P\mathbf{b}\,.$$

What brings us to the resolution of triangular systems

$$\begin{cases} L\mathbf{y} = P\mathbf{b}, \\ U\mathbf{x}^* = \mathbf{y}. \end{cases}$$

and finally we compute

$$\mathbf{x} = Q\mathbf{x}^*.$$

Remark 2. *In Matlab/Octave, we can get the factorization of a matrix A with the command*

```
>> [L,U,P] = lu(A);
```

The matrix P is a permutation matrix. In the case where the matrix P is the identity, the matrices L and U are the matrices we are looking for (such that $LU = A$). Otherwise, we have $LU = PA$.

Example. 2 (continued) In Matlab/Octave, we first need to define the matrix `A` and the vector `b` of the linear system:

```
>> A = [-0.37, 0.05, 0.05, 0.07; 0.05, -0.116, 0, 0.05;...
        0.05, 0, -0.116, 0.05; 0.07, 0.05, 0.05, -0.202];
>> b = [-2; 0; 0; 0];
```

Then, we can use the command \ as follows:

```
>> x = A\b
x =
    8.1172
    5.9893
    5.9893
    5.7779
```

This command computes the solution of the system with *ad hoc* algorithms (it tests the matrix to choose an optimal algorithm).

Remark: the pressures at the nodes 2 and 3 are the same (by symmetry).

If we wanted to use the LU factorization:

```
>> [L,U,P] = lu(A);
>> y = L\(P*b);
>> x = U\y
x =
    8.1172
    5.9893
    5.9893
    5.7779
```

The solution is the same. We can verify that, in this case, no permutation takes place (P is the identity matrix):

```
>> P
P =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Remark 3. *Using the LU factorization, obtained by fixing the value* 1 *for the* n *diagonal elements of* L*, we can calculate the determinant of a square matrix with* $O(n^3)$ *operations, because*

$$\det(A) = \det(L)\det(U) = \det(U) = \prod_{k=1}^{n} u_{kk};$$

indeed, the determinant of a triangular matrix is is the product of the diagonal elements. The Matlab/Octave command `det(A)` *makes use of this.*

The inverse matrix

If A is a $n \times n$ non-singular matrix, let us call $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ the columns of its inverse matrix A^{-1} i.e. $A^{-1} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)})$.
The relation $AA^{-1} = I$ can be expressed by the following n systems : for $1 \leq k \leq n$,

$$A\mathbf{x}^{(k)} = \mathbf{e}^{(k)}, \tag{12}$$

where $\mathbf{e}^{(k)}$ is the column vector with all the elements equal to 0 except the one corresponding to the k-th row, which equals 1.
Once we know the matrices L and U that factorizes A, solving the n systems (12) defined by the same matrix A requires $2n^3$ operations.

The Cholesky factorization

In the case where the $n \times n$ matrix A is symmetric and positive definite, there exists a unique upper triangular matrix R with positive diagonal elements such that

$$\boxed{A = R^T R.}$$

This factorization is called $Cholesky\ factorization$. In Matlab/Octave, the command

```
>> R = chol(A)
```

can be used to compute R.

The elements r_{ij} of R can be calculated using the expressions $r_{11} = \sqrt{a_{11}}$ and for $i = 2, \ldots, n$:

$$r_{ji} = \frac{1}{r_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} r_{ki} r_{kj} \right), \quad j = 1, \ldots, i-1, \tag{13}$$

$$r_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2} \tag{14}$$

The Cholesky factorization needs around $\frac{n^3}{3}$ operations (half the operations for a LU factorization).

Example 6. In Matlab there are several families of predefined matrices (see `help gallery`). Let us consider the family of the *Lehmer* matrices, that are positive definite and thus candidates for a Cholesky factorization. We want to calculate the cost of the Cholesky factorization for matrices of size $n = 10, 20, 30, 40$ and 50.

Remark : In the version 5 of Matlab there was available the command `flops` which allowed to calculate the number of operations executed. Unfortunately, in the new version 6 of Matlab, this command has been removed (due to the integration in Matlab of certain linear algebra libraries). This example can't be executed on Matlab 6:

```
>> fl=[];
>> for n=10:10:50
    A=gallery('lehmer',n);
    flops(0)
    R=chol(A);
    fl = [fl, flops];
   end
```

We get the number of *flops* for the different sizes

```
>> fl
fl =
         385         2870         9455        22140        42925
```

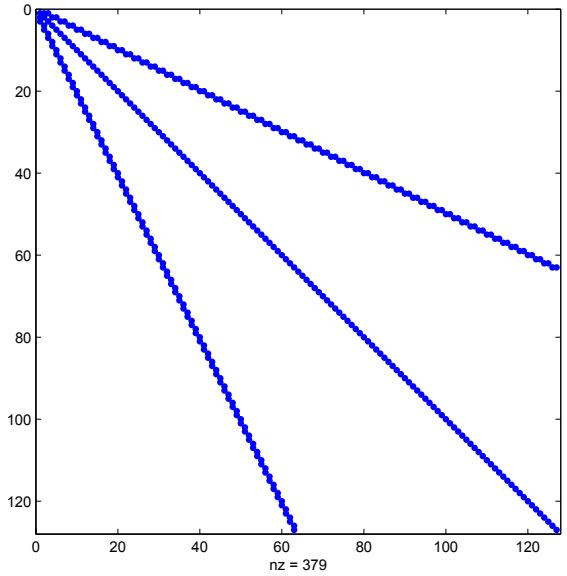
that grow proportionally to n^3. Indeed, if we calculate the ratio $fl(n)/n^3$ we find a "quasi" constant value.

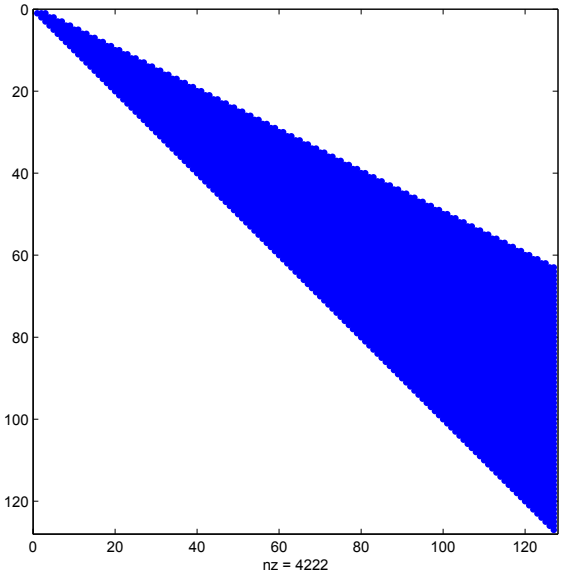
```
>> n=[10, 20, 30, 40, 50];
>> fl./(n.^3)
ans =
    0.3850    0.3588    0.3502    0.3459    0.3434
```

Remark that this value is approximately $\frac{1}{3}$. Thus, we can say that the computational cost of the Cholesky factorization is of order $\frac{1}{3}n^3$, or half the cost of the Gauss elimination method.

Memory space limitations

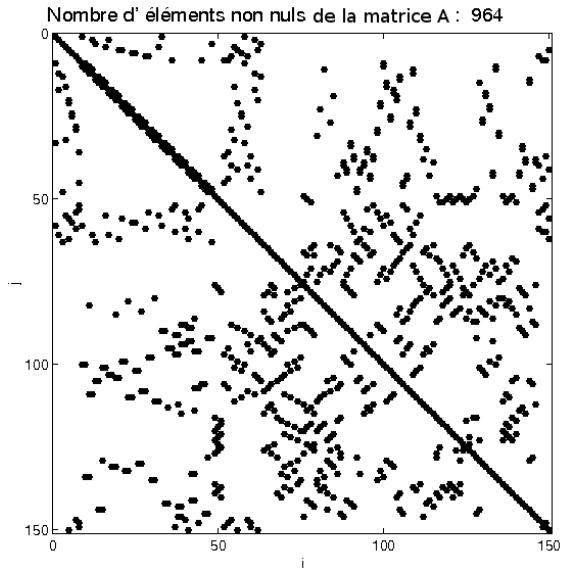
Example. 1 (continued) Let A be a matrix of size 127×127 corresponding to capillary bed with 8 bifurcation levels. This matrix is symmetric definite positive. The number of non-null entries of A is 379 and thus much smaller than $(127)^2 = 16129$. It is a *sparse* matrix. The figure on the left shows the disposition of the non-null entries of A, whereas the one on the right shows the non-null entries of the matrix R.

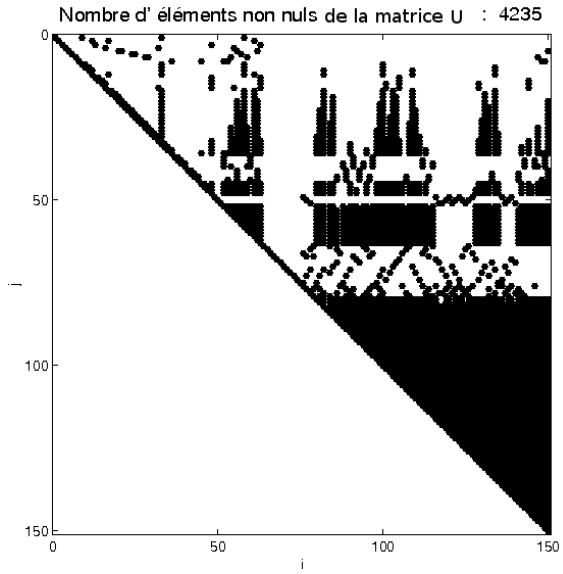




Example 7. Let us consider the problem of calculating the deformations in a structure subject to a given set of forces. The discretization using the finite elements method generates a matrix A of size 150×150. (The same matrix would have been produced by the approximation of an electric potential field.) This matrix is symmetric definite positive. The number of non-null entries of A is 964, and thus much smaller than $(150)^2 = 22500$. It is a *sparse* matrix.

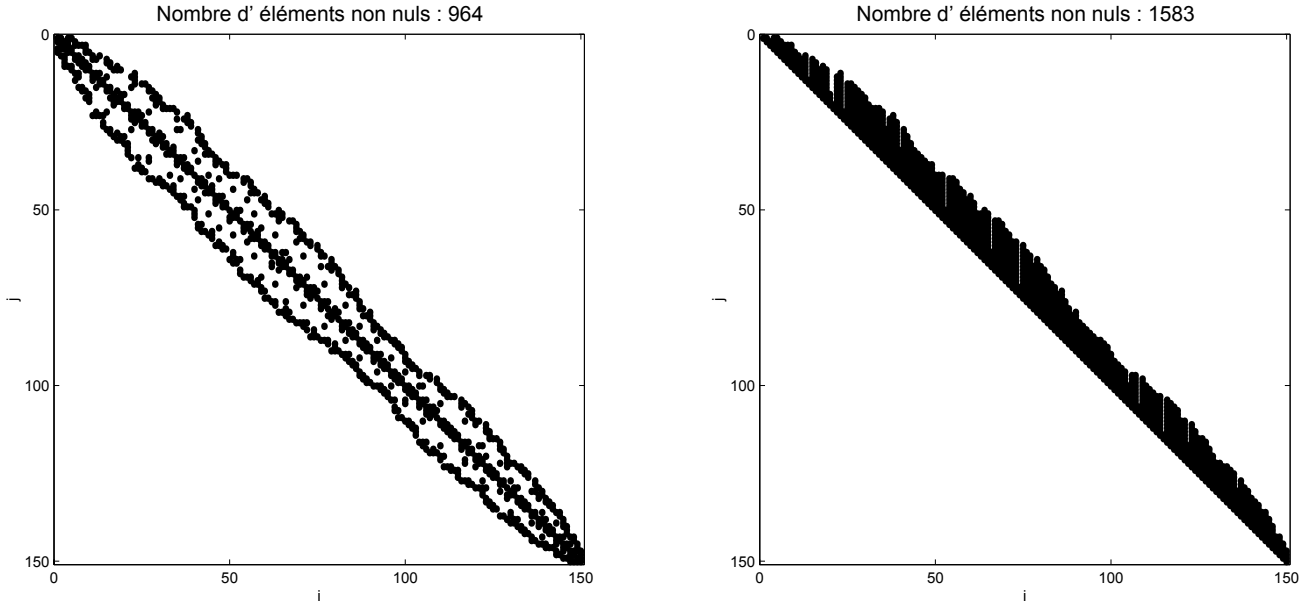
The figure on the left shows the disposition of the non-null entries of A, whereas the one on the right shows the non-null entries of the matrix R.





We notice that the number of non-null entries of R is much bigger than those of A (*fill-in phenomenon*). This leads to a bigger memory usage. To reduce the fill-in phenomenon, we can we can re-order rows and columns of A in a particular fashion; this is called *re-ordering* of the matrix. There are several algorithms that allow us to do this (Matlab command: `symand`).

For example, the following figure shows, on the left, one possibility of reordering A, while the one on the right shows the disposition of the non-null entries of the Cholesky factorization of the reordered matrix A.



Precision limitations

Example 8. Rounding errors can induce important differences between the calculated solution using the Gauss elimination method (GEM) and the exact solution. This happens when the *conditioning* of the matrix of the system is very big.

The Hilbert matrix of size $n \times n$ is a symmetric matrix defined by:

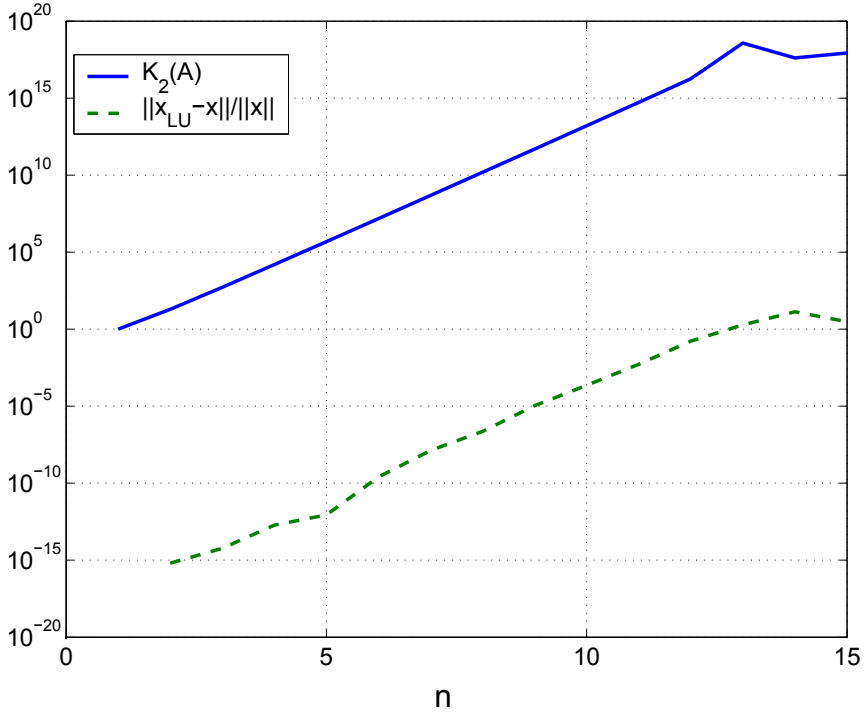
$$A_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,\ldots,n$$

In Matlab/Octave, we can build a Hilbert matrix of any size n with the command `A = hilb(n)`. For example, for $n = 4$, we get:

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

We consider the linear systems $A_n \mathbf{x}_n = \mathbf{b}_n$ where A_n is the Hilbert matrix of size n with $n = 4, 6, 8, 10, 12, 14, \ldots$, whereas $\mathbf{b}_n$ is chosen such that the exact solution is $\mathbf{x}_n = (1, 1, \cdots, 1)^T$.

For every n, we calculate the conditioning of the matrix, we solve the linear system by LU factorization and we get $\mathbf{x}_n^{LU}$ as the found solution. The obtained conditioning as well as the error $\|\mathbf{x}_n - \mathbf{x}_n^{LU}\|/\|\mathbf{x}_n\|$ (where $\|\cdot\|$ is the euclidian norm of a vector, $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \cdot \mathbf{x}}$) are shown in the figure below.



Considerations on the precision (Sect. 5.5

The methods we have seen until now allow us to find the solution of a linear system in a finite number of operations. That is why they are called *direct methods*. However, there are cases where these methods are not satisfactory.

Definition 1. *We call* ***conditioning*** *of a matrix M, symmetric definite positive, the ratio between the maximum and minimum of its eigenvalues, i.e.*

$$K(M) = \frac{\lambda_{max}(M)}{\lambda_{min}(M)}$$

It can be shown that, the bigger the conditioning of a matrix, the worse the solution obtained by a direct method.
For example, let us consider a linear system $A\mathbf{x} = \mathbf{b}$.
If we solve this system with a computer, due to rounding errors, we will not find the exact solution $\mathbf{x}$ but an approximate solution $\hat{\mathbf{x}}$. The following relationship can be shown :

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \tag{15}$$

where $\mathbf{r}$ is the residual $\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}$; we write as $\|\mathbf{v}\| = \left(\sum_{k=1}^{n} v_k^2\right)^{1/2}$ the Euclidean norm of a vector $\mathbf{v}$.
Remark that, if the conditioning of A is big, the distance $\|\mathbf{x} - \hat{\mathbf{x}}\|$ between the exact solution and the numerically computed solution can be very big even if the residual is very small.

Proof for (15) : Let A be a symmetric definite positive matrix, we can consider the n eigenvalues $\lambda_i > 0$ and the associated unitary eigenvectors $\{\mathbf{v}_i\}$, $i = 1, \ldots, n$: $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$, $i = 1, \ldots, n$. These vectors form an orthonormal base of $\mathbb{R}^n$, what means $\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}$ for $i, j = 1, \ldots, n$. For any $\mathbf{w} \in \mathbb{R}^n$, if we write it as

$$\mathbf{w} = \sum_{i=1}^{n} w_i \mathbf{v}_i,$$

we have

$$\begin{aligned} \|A\mathbf{w}\|^2 &= (A\mathbf{w})^T (A\mathbf{w}) \\ &= (\lambda_1 w_1 \mathbf{v}_1^T + \ldots \lambda_n w_n \mathbf{v}_n^T)(\lambda_1 w_1 \mathbf{v}_1 + \ldots \lambda_n w_n \mathbf{v}_n) \\ &= \sum_{i,j=1}^{n} \lambda_i \lambda_j w_i w_j \mathbf{v}_i^T \mathbf{v}_j = \sum_{i,j=1}^{n} \lambda_i \lambda_j w_i w_j \delta_{ij} = \sum_{i=1}^{n} \lambda_i^2 w_i^2. \end{aligned}$$

And yet, as $\|\mathbf{w}\|^2 = \sum_{i=1}^{n} w_i^2$, we get $\|A\mathbf{w}\|^2 \leq \lambda_{max}^2 \|\mathbf{w}\|^2$, i.e. $\|A\mathbf{w}\| \leq \lambda_{max}\|\mathbf{w}\|$ where λ_{max} is the biggest eigenvalue of A.

As the eigenvalues of A^{-1} are $1/\lambda_i$, we also get $\|A^{-1}\mathbf{w}\| \leq \frac{1}{\lambda_{min}}\|\mathbf{w}\|$ $\forall \mathbf{w} \in \mathbb{R}^n$, where λ_{min} is the smallest eigenvalue of A. Thus, we have

$$\begin{aligned} \|\mathbf{x} - \hat{\mathbf{x}}\| &= \|A^{-1}\mathbf{r}\| \leq \frac{1}{\lambda_{min}}\|\mathbf{r}\|, \\ \|\mathbf{b}\| &= \|A\mathbf{x}\| \leq \lambda_{max}\|\mathbf{x}\|, \end{aligned}$$

from where we directly find the inequality (15). ■