Least Squares

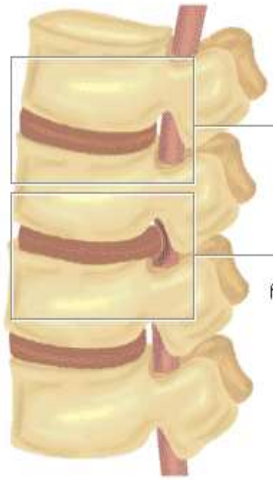
Numerical Analysis

Profs. Gianluigi Rozza- Luca Heltai

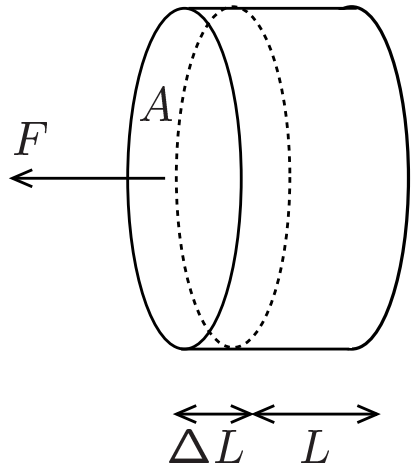
2019-SISSA mathLab Trieste

Examples and motivations

Example 1. We consider a mechanical test to establish the link between stresses ($MPa = 100N/cm^2$) and relative deformations (cm/cm) of a sample of biological tissue (an intervertebral disc, taken from P. Komarek, Chapt. 2 of *Biomechanics of Clinical Aspects of Biomedicine*, 1993, J. Valenta ed., Elsevier).



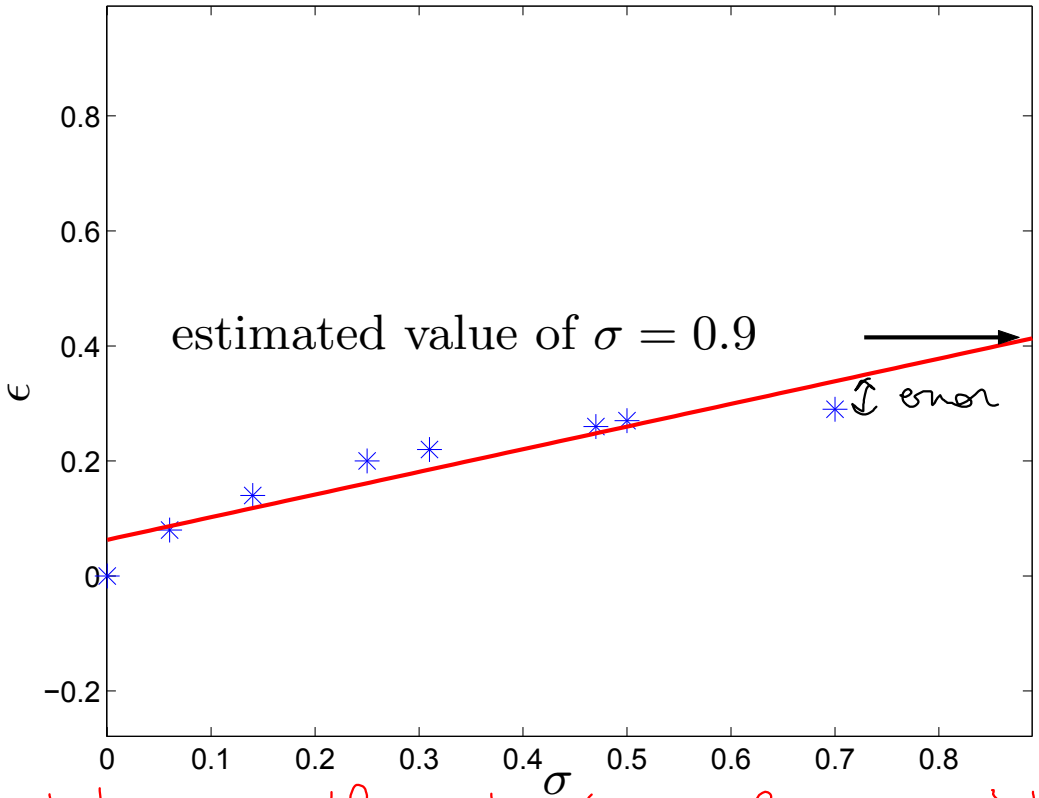
test i	pressure σ	deformation ϵ
1	0.00	0.00
2	0.06	0.08
3	0.14	0.14
4	0.25	0.20
5	0.31	0.23
6	0.47	0.25
7	0.60	0.28
8	0.70	0.29



$$\sigma = F/A$$
$$\epsilon = \Delta L/L$$

From these data, we want to estimate the deformation corresponding to a stress $\sigma = 0.9$ MPa.

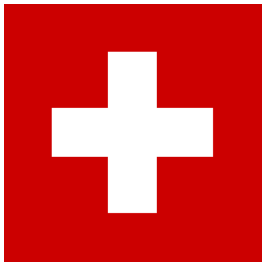
By the method of least squares, we get that best approximation for the data is $p(x) = 0.3938x - 0.0629$. The approximation can be used (called *a linear regression*) to estimate ϵ when $\sigma = 0.9$ MPa: We get $p(0.9) \simeq 0.4$.



→ we build a TREND by minimizing the distance between data and line

! If you want to approx this data (could be $10^2, 10^3$ data collected) with std interpolation you'll get a poly with high degree → NOT GOOD!

Example 2. The result of census of the population of Switzerland between 1900 and 2010 (in thousands):



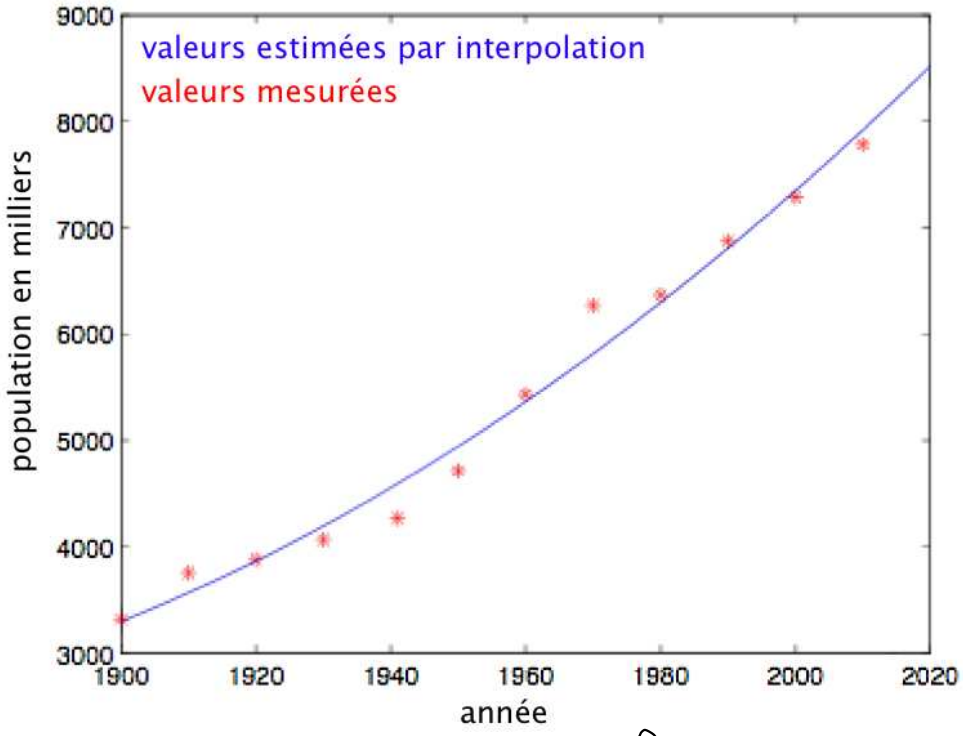
year	1900	1910	1920	1930	1941	1950
population	3315	3753	3880	4066	4266	4715

year	1960	1970	1980	1990	2000	2010
population	5429	6270	6366	6874	7288	7783

- Is it possible to estimate the number of inhabitants of Switzerland during the year when there has not been census, for example in 1945 and 1975?
- Is it possible to predict the number of inhabitants of Switzerland in 2020?

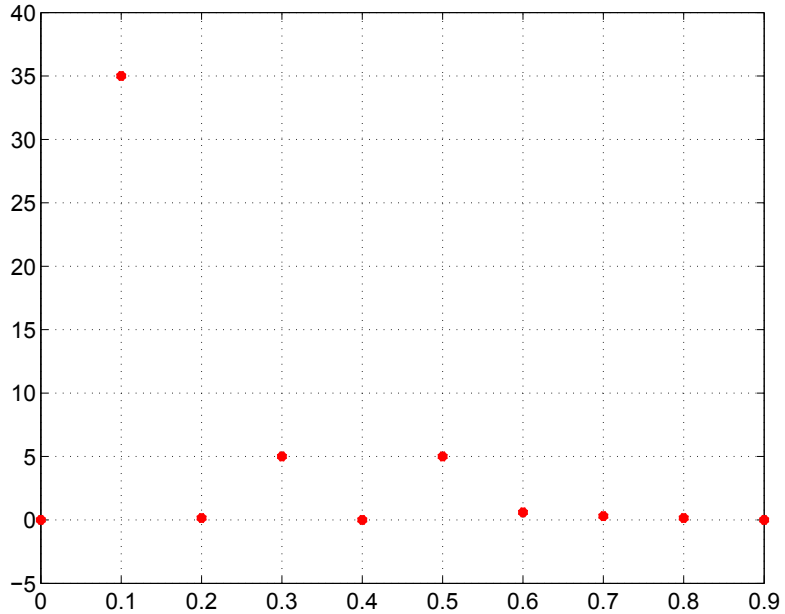
The polynomial of degree two (parabola) which approximates the data by the method of least squares is $p(x) = 0.15x^2 - 549.9x + 501600$.

↑ II ord poly. LEAST SQUARES



When fitting we should guarantee the minimization of a qty, like errors

Example 3. Points on the following figure represent the measurements of blood flow in a part of the common carotid artery during a heartbeat. The frequency of data collection is constant and equal to 10 / T where T = 1 sec. is the period of the beat.
We want to deduce from this discrete signal a continuous signal represented by a linear combination of known functions (eg. trigonometric functions - then we can adequately approximate a periodic signal).



reconstruction of the behavior of the flow rate

Let $f(t)$ be a signal that we know. $N = 10$ is a size of a sample $[f(t_0), \ldots, f(t_{N-1})]$, where $t_j = jT/N$. We are looking for $\{c_k\} \in \mathbb{C}$, $k \in [0, N-1] \subset \mathbb{N}$ such that:

vector of weights is the only thing we have to get in order to get the expression of reconstructed flow signal

basis function

$$f(t_j) = \frac{1}{N} \sum_{k=0}^{N-1} c_k \omega_N^{-kj}, \quad j = 0, \ldots, N-1, \tag{1}$$

where

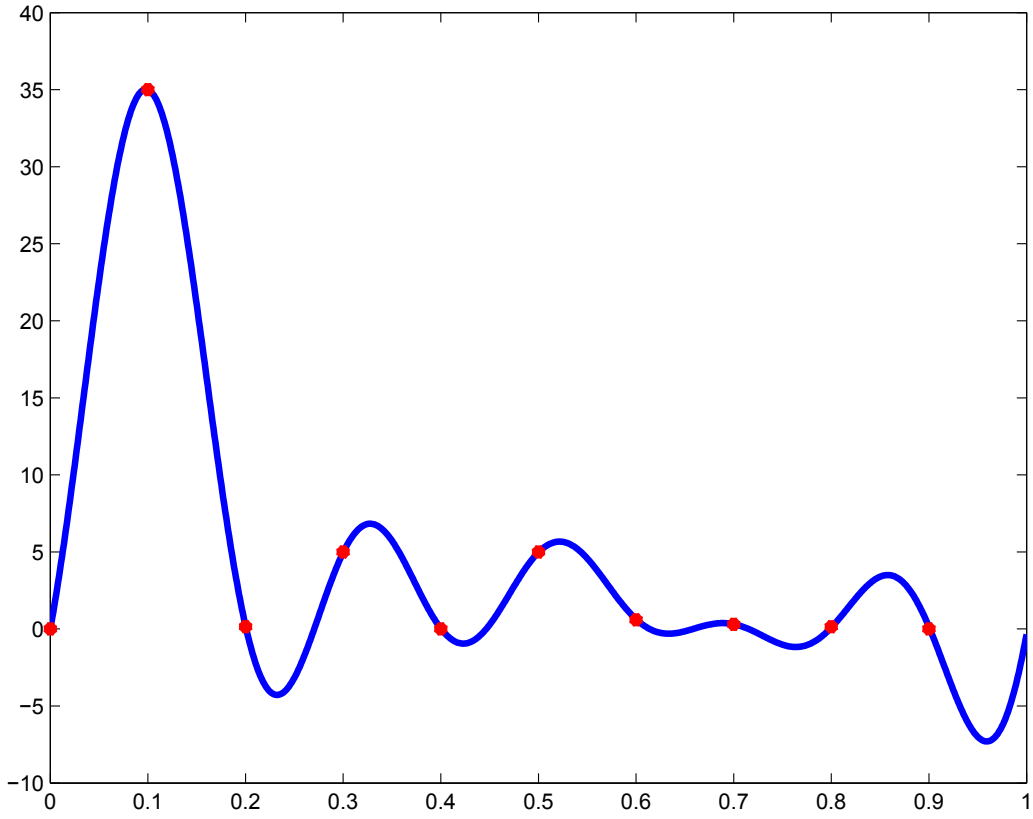
$$\omega_N = e^{(-2\pi i)/N} = \cos(2\pi/N) - i \sin(2\pi/N),$$

i is the imaginary unit. We can calculate a vector of the coefficients $\mathbf{c} = [c_1, \ldots, c_{10}]^T$ by the FFT algorithm (Fast Fourier Transform, Cooley & Tuckey, 1965), implemented in Matlab/Octave by the `fft` command.

# of data

look for this in python library

Using the formula (1), there are several techniques for obtaining the value of f in each point $t \in [0, T]$.
Using one of techniques we obtain a result that is plotted on a following figure:



If you use
Eigen latest
expect Eigen
costs; but if
you reduce it,
expect Eigen
accurate results

Approximation by the least square metho

(Chapt. 3.4 of the book)

Suppose we have $n+1$ points $x_0, x_1, \ldots, x_n$ and $n+1$ values $y_0, y_1, \ldots, y_n$.
We have seen that if n is large, the interpolating polynomial may show large oscillations (high degree!)

Instead of interpolating the values, it is possible to define a polynomial of degree $m < n$ that approximates the data "at best"

Definition 1. *We call least squares polynomial approximation of degree m the polynomial $\tilde{f}_m(x)$ of degree m such that*

$$\sum_{i=0}^{n} |y_i - \tilde{f}_m(x_i)|^2 \leq \sum_{i=0}^{n} |y_i - p_m(x_i)|^2 \qquad \forall p_m(x) \in \mathbb{P}_m$$

Remark 4. *Where $y_i = f(x_i)$ (f is a continuous function) then $\tilde{f}_m$ is called the approximation of f in the least squares sense.*

↳ minimizing the error by selecting the best polynomial of some degree

In other words, the least squares polynomial approximation is the polynomial of degree m that minimizes the distance to the data.

Let note $\tilde{f}_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$ and define the function

$$\Phi(a_0, a_1, \ldots, a_m) = \sum_{i=0}^{n} \left| y_i - \left(a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m\right)\right|^2$$

Then the coefficients of $\tilde{f}^m$ can be determined by the relation

$$\frac{\partial \Phi}{\partial a_k} = 0, \qquad k = 0, \ldots, m, \tag{5}$$

i.e., $m+1$ linear equations in with $m+1$ unknowns a_k, $k = 0, \ldots, m$.

Example 8. Let $n = 2$ and $m = 1$, nodes $x_0 = 1$, $x_1 = 3$, $x_2 = 4$ with values $y_0 = 0$, $y_1 = 2$, $y_2 = 7$. We want to compute the (*regression line*), i.e., the the least squares polynomial approximation of degree 1, $\tilde{f}_1(x) = a_0 + a_1 x$.

We set $\Phi(a_0, a_1) = \sum_{i=0}^{2} [y_i - (a_0 + a_1 x_i)]^2$ and impose $\dfrac{\partial \Phi}{\partial a_0} = 0$ and $\dfrac{\partial \Phi}{\partial a_1} = 0$:

$$\begin{aligned}
\frac{\partial \Phi}{\partial a_0} &= -2 \sum_{i=0}^{2} [y_i - (a_0 + a_1 x_i)] = -2 \left(\sum_{i=0}^{2} y_i - 3a_0 - a_1 \sum_{i=0}^{2} x_i \right) \\
&= -2(9 - 3a_0 - 8a_1) \\
\frac{\partial \Phi}{\partial a_1} &= -2 \sum_{i=0}^{2} x_i [y_i - (a_0 + a_1 x_i)] = -2 \left(\sum_{i=0}^{2} x_i y_i - a_0 \sum_{i=0}^{2} x_i - a_1 \sum_{i=0}^{2} x_i^2 \right) \\
&= -2(34 - 8a_0 - 26a_1)
\end{aligned}$$

Hence the coefficients a_0 and a_1 are the solution of the system

$$\begin{cases} 3a_0 + 8a_1 = 9 \\ 8a_0 + 26a_1 = 34 \end{cases}$$

Generalize to odd m

Ideally, for $\tilde{f}_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots$ we would like to impose $\tilde{f}_m(x_i) = y_i$ pour $i = 0, \ldots, n$.
This can be written as a linear system with unknowns a_k, $k = 0, \ldots, m$: $B\mathbf{a} = \tilde{\mathbf{y}}$, where B is a matrix of dimension $(n+1) \times (m+1)$

dim of sol of approx

$$B = \begin{pmatrix} 1 & x_0 & \ldots & x_0^m \\ 1 & x_1 & \ldots & x_1^m \\ \vdots & & & \vdots \\ 1 & x_n & \ldots & x_n^m \end{pmatrix}$$

dim of dataset

A rectangular B matrix is NOT solvable as a linear system

Since $m < n$, the system is oversized. The solution to (5) is equivalent to the square system (*system of normal equations*)

$$\boxed{B^T B\mathbf{a} = B^T \tilde{\mathbf{y}}.}$$

premultiply by B^T → you get a square matrix, now you can solve shit

Example 9. We have 8 measures (σ-ϵ):

```
>> sigma = [0.00 0.06 0.14 0.25 0.31 0.47 0.50 0.70];
>> epsilon = [0.00 0.08 0.14 0.20 0.22 0.26 0.27 0.29];
```

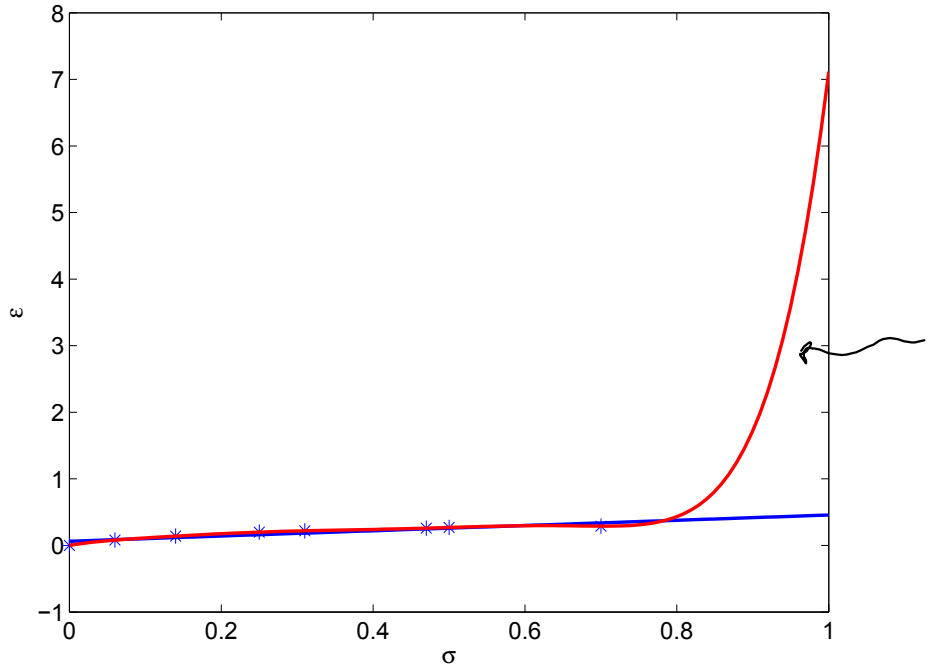
We want to extrapolate the value of ϵ for$\sigma = 0.4$. We consider two ways:

- compute the interpolating polynomial Π_7 of degree 7
- compute the least squares polynomial approximation of degree 1

On peut utiliser les commandes suivantes:

```
>>  plot(sigma, epsilon, '*'); hold on;  % plotting the known values
>>  sigma_sample = linspace(0,1.0,100);
>>  p7 = polyfit(sigma, epsilon, 7);
>>  pol = polyval(p7, sigma_sample);     % interpolating polynomial
>>  plot(sigma_sample,pol, 'r');
>>  p1 = polyfit(sigma,epsilon,1);
>>  pol_mc = polyval(p1, sigma_sample);  % least square
>>  plot(sigma_sample, pol_mc, 'g'); hold off;
```

- the interpolating polynomial Π_7 of degree 7 is in red
- the least squares polynomial approximation of degree 1 is in blue



degree 7 gives a very bad approx at the end

For $\sigma > 0.7$,the behavior of the two polynomials are very different.

In particular, for $\sigma = 0.9$ the values of $\epsilon(\sigma)$ extrapolated with the two methods are

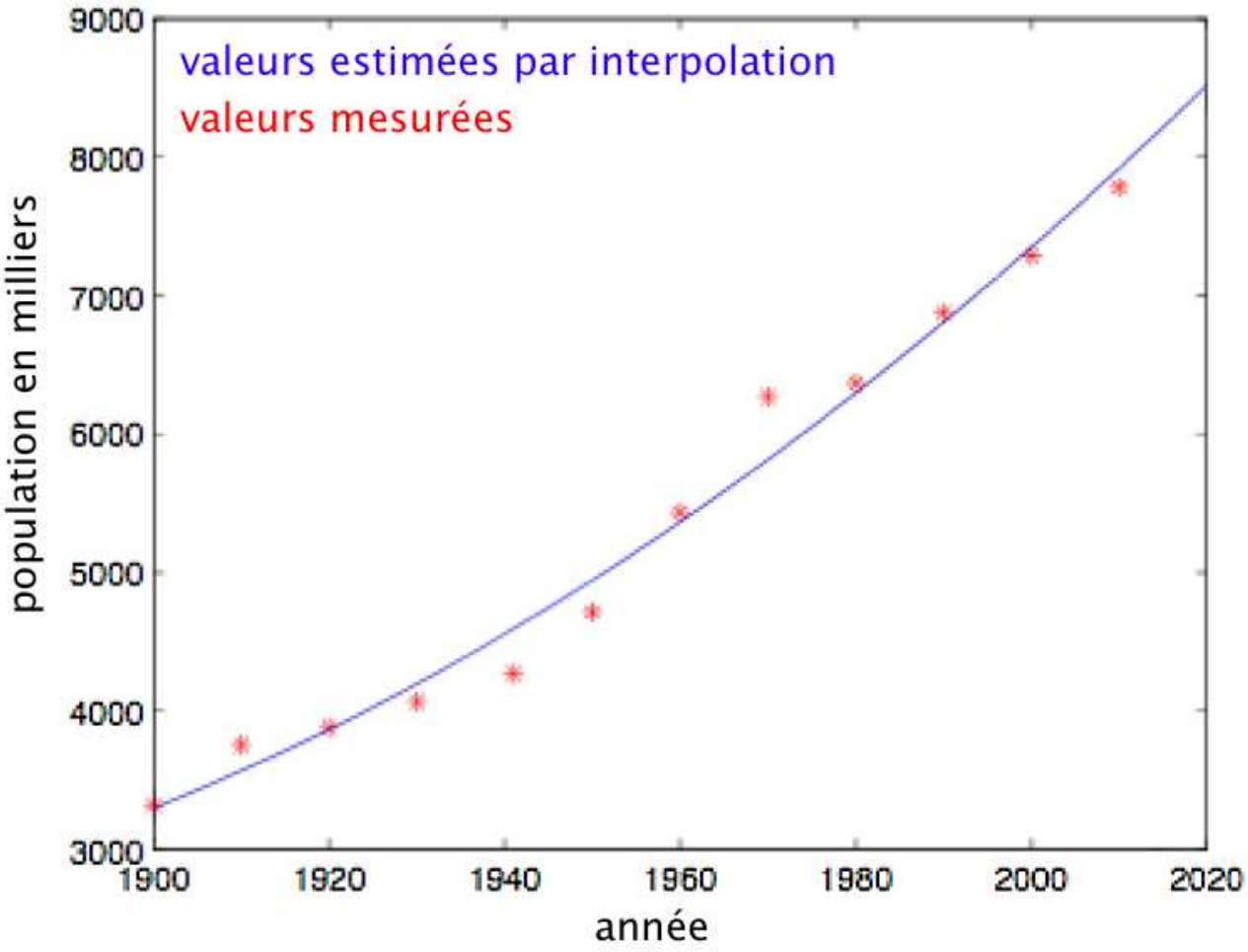
```
>> polyval(p7, 0.9)
ans =
    1.7221

>> polyval(p1, 0.9)
ans =
    0.4173
```

- The value obtained by Π_7 (172.21%) is unrealistic
- On the contrary, the value obtained with the regression line is more appropriate to compute the value at $\sigma = 0.9$.

Example 10. Swiss population Starting from the population at the 20th century decades, we extrapolate the Swiss popululation this year. We use a least square polynomial approximation of degree 2

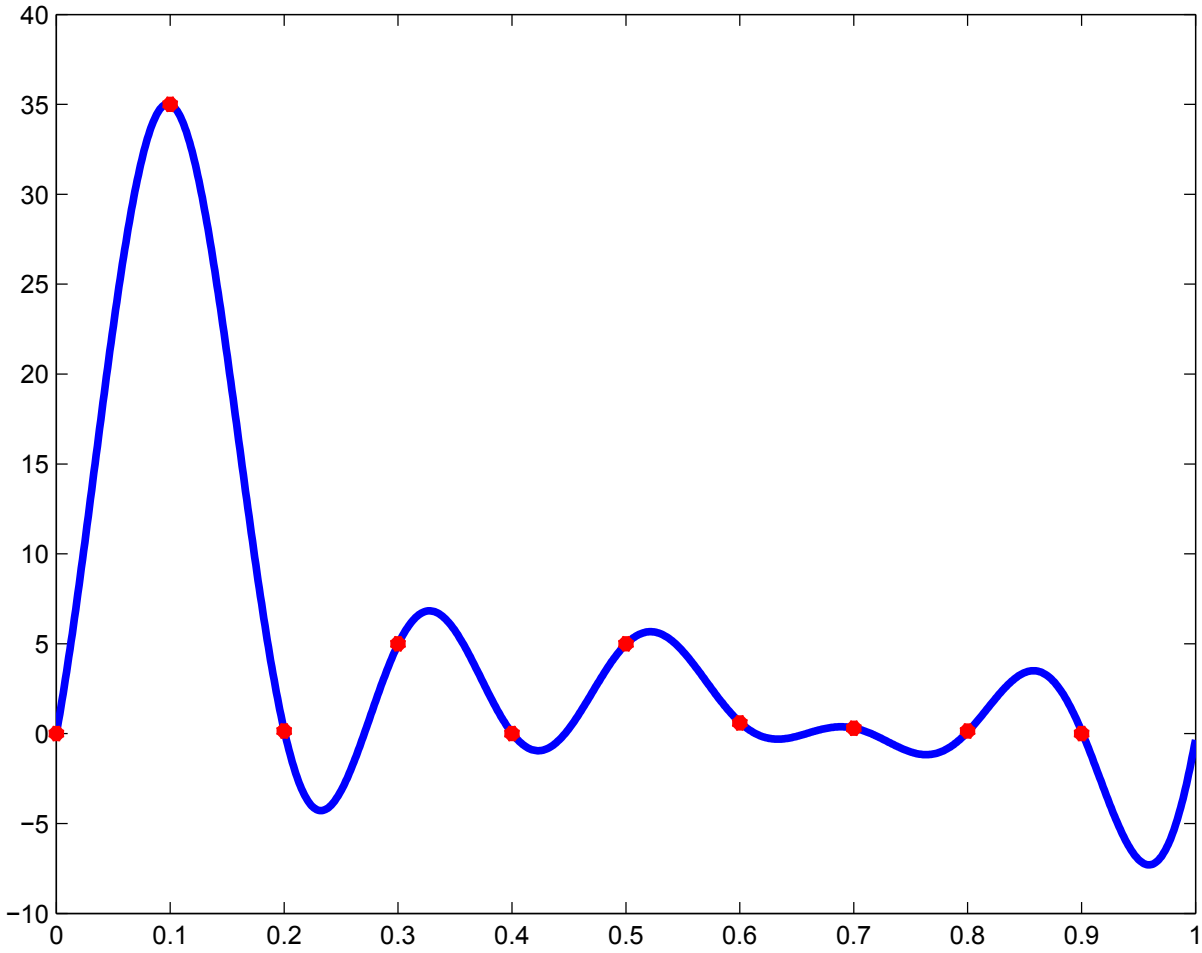
```
>> year = [1900, 1910, 1920, 1930, 1941, 1950,...
           1960, 1970, 1980, 1990, 2000];
>> population = [3315, 3753, 3880, 4066, 4266, 4715,...
           5429, 6270, 6366, 6874, 7288];
>> p2 = polyfit(year, population, 2);
>> year_sample = [1900:2:2010];
>> vp2 = polyval(p2, year_sample);
>> plot(year, population, '*r', ...
       year_sample, vp2, 'b');
```


valeurs estimées par interpolation
valeurs mesurées
population en milliers
année
9000
8000
7000
6000
5000
4000
3000
1900
1920
1940
1960
1980
2000
2020

Example. Carotide flow rate We define 10 couples of data (time vs flowrate) with the vectors **t** and **deb**. With the help of the command **interpft**, we compute the value of the interpolating function at 1000 equidistributed points in the periodicity interval [0,1]:

```
t = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9];
deb = [0 35 .15 5 0 5 .6 .3 .15 0];
f = interpft(deb, 1000);
plot(linspace(0, 1, 1000), f); hold on;
plot(t, deb, 'r*')
```

interpolation function


40
35
30
25
20
15
10
5
0
−5
−10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Processing of polynomials

In Matlab/Octave, there are specific commands for doing calculations with polynomials. Let `x` be a vector of abscissas, `y` be a vector of ordinates and `p` (respectively $\texttt{p}_i$) be the vector of coefficients of a polynomial $P(x)$ (respectively P_i); then, we have following commands:

command	action
`y=polyval(p,x)`	`y` $=$ values of $P(x)$
`p=polyfit(x,y,n)`	`p` $=$ coefficients of the interpolating polynomial Π_n
`z=roots(p)`	`z` $=$ zeros of P such that $P(\texttt{z}) = 0$
`p=conv(`$\texttt{p}_1$`,`$\texttt{p}_2$`)`	`p` $=$ coefficients of the polynomial $P_1 P_2$
`[q,r]=deconv(`$\texttt{p}_1$`,`$\texttt{p}_2$`)`	`q` $=$ coefficients of Q, `r` $=$ coefficients of R such that $P_1 = QP_2 + R$
`y=polyderiv(p)`	`y` $=$ coefficients of $P'(x)$
`y=polyinteg(p)`	`y` $=$ coefficients of $\int P(x)\, dx$

Linear systems - Direct methods

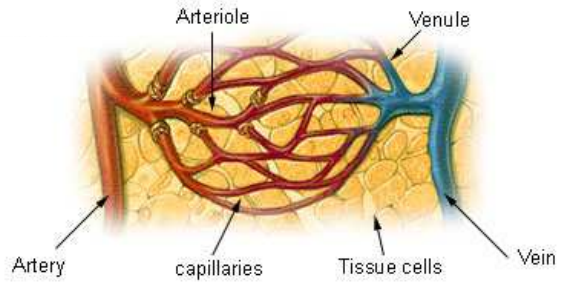
Numerical Analysis

Profs. Gianluigi Rozza - Luca Heltai

2019-SISSA mathLab Trieste

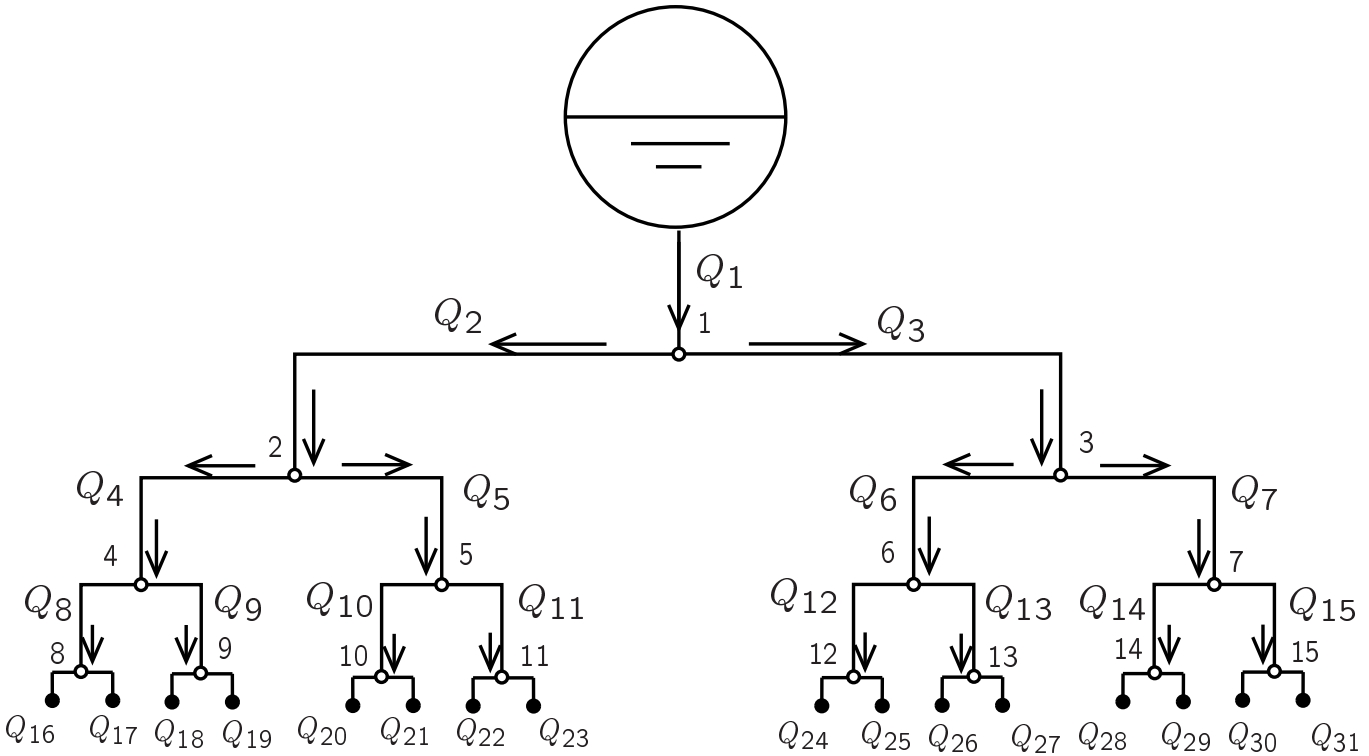
Examples and motivation

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 ($\text{mmHg}\,\text{s}/\text{mm}^4$) 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:

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

A contains all the properties of our system

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.

! Do evaluations of ybies you save for multiple use, in order to be more efficient and avoid "repetitivity" → save time and computational costs

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

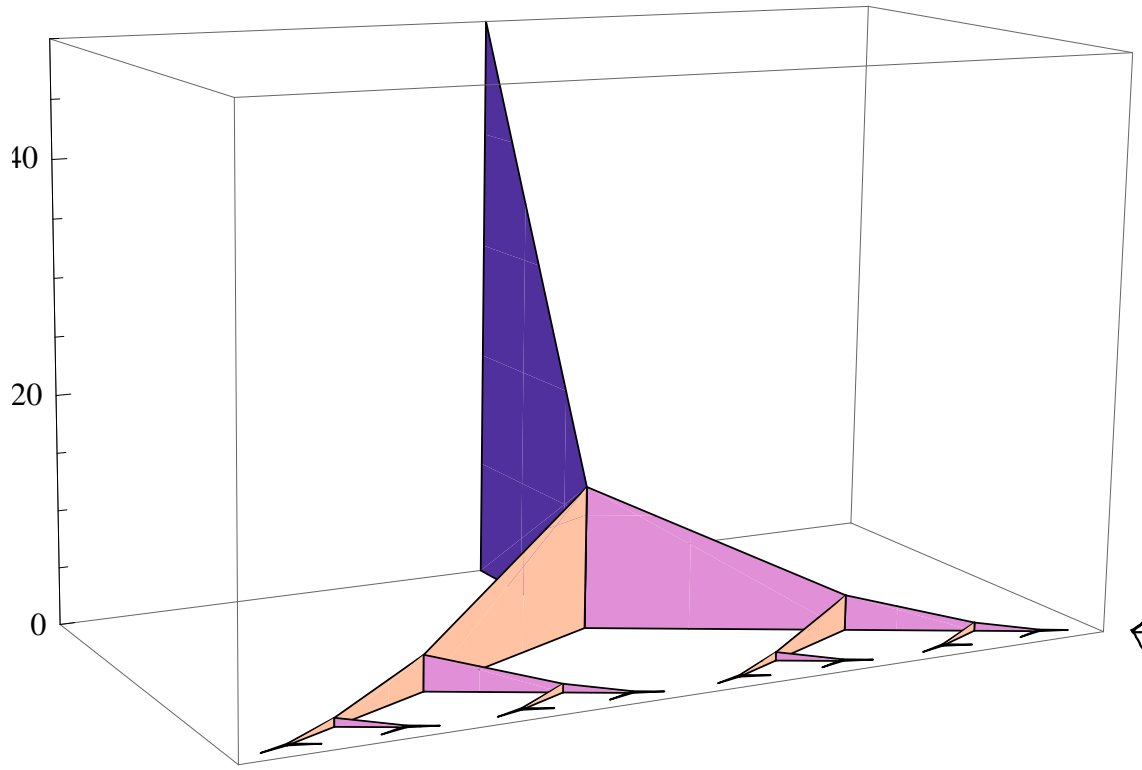
The problem of determining the pressures and flows in the network requires to solve a linear system $A\mathbf{p} = \mathbf{b}$.
In this case, the matrix A is symmetric and definite positive. This last property ensures that the matrix A is not singular and thus that the system has a unique solution.
The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

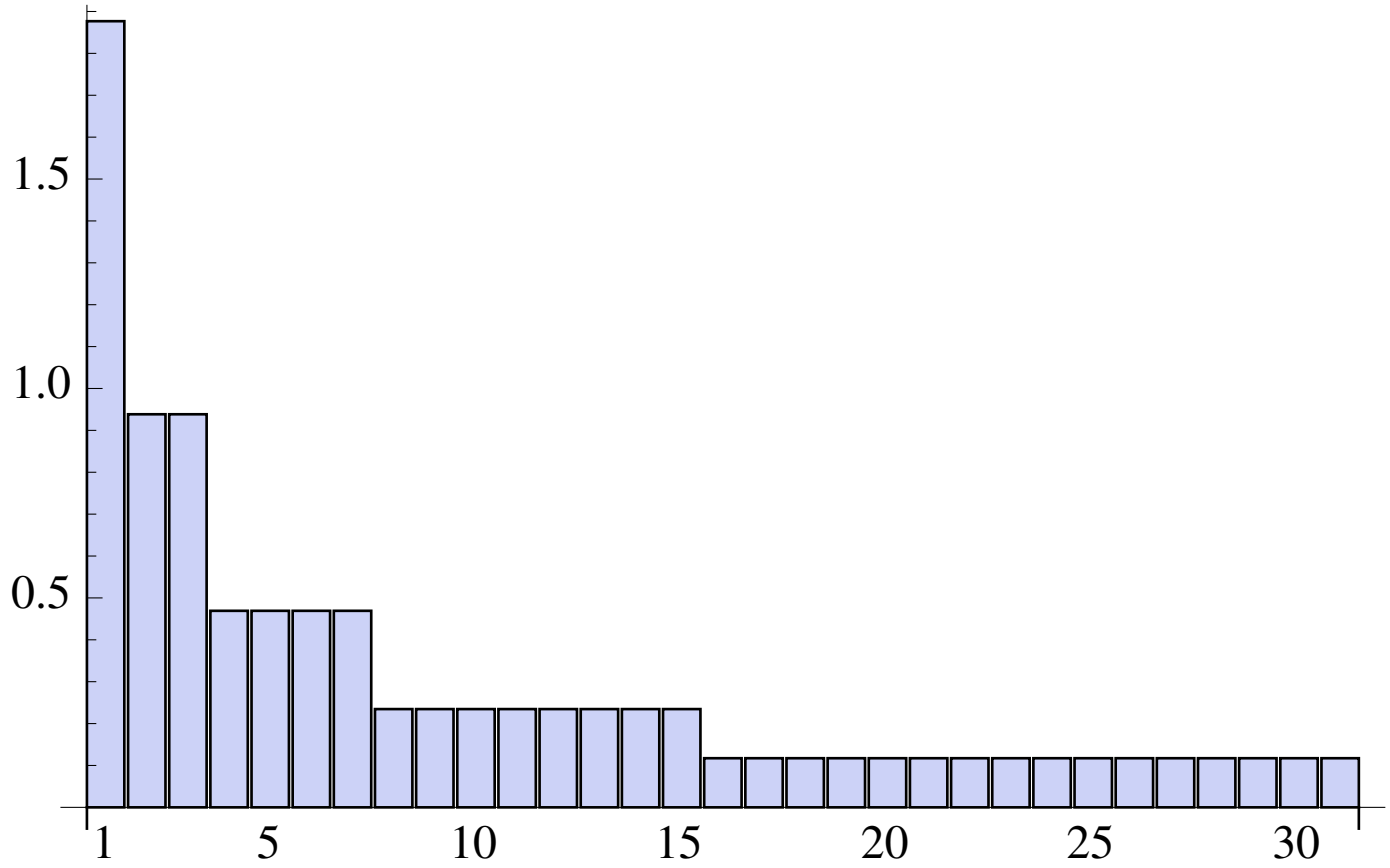
$$\begin{aligned} Q_1 &= 1.88 \\ Q_{2,3} &= 0.94 \\ Q_{4,\cdots,7} &= 0.47 \\ Q_{8,\cdots,15} &= 0.23 \\ Q_{16,\cdots,31} &= 0.12 \end{aligned}$$

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

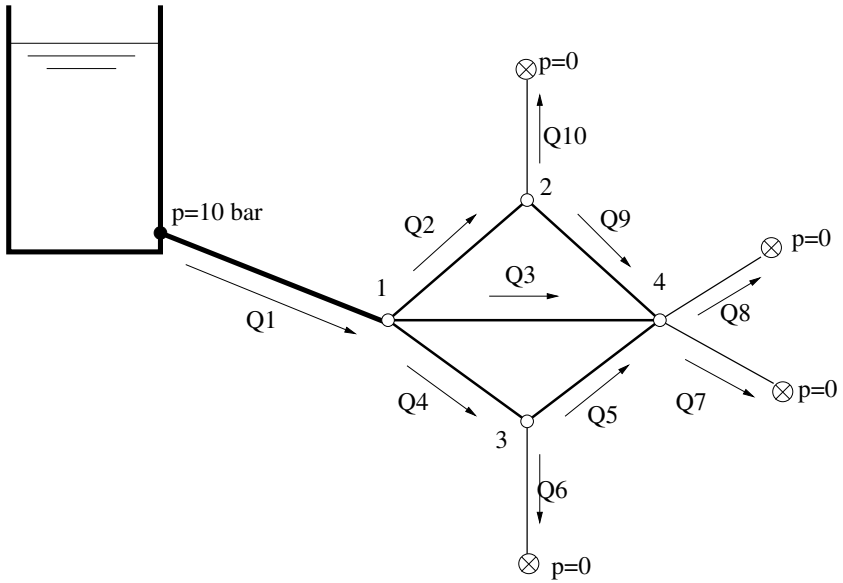


system is dissipating energy ⇓ we have a drop in pressure

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.

unknowns

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:

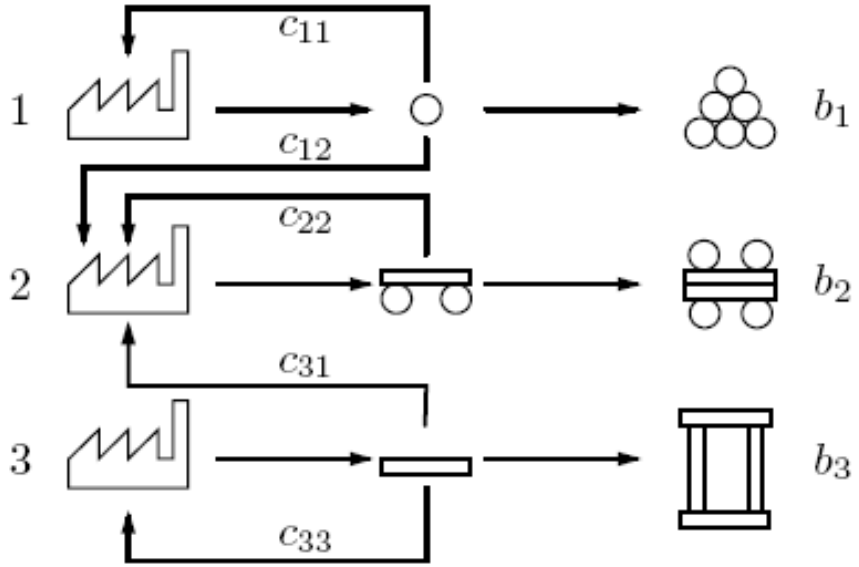
symmetrical

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

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):

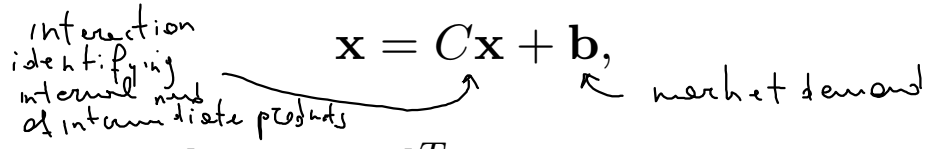
Q1	0.3766	Q2	1.0640	Q3	0.1638
Q4	0.1064	Q5	0.0106	Q6	0.0958
Q7	0.0924	Q8	0.0924	Q9	0.0106
Q10	0.0958				

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



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 $\mathbf{b}$:

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

NOT EASY TO COMPUTE det(Bi) because it requires ~ N! flops (floating point operation)

=> COMPUTATIONAL COST OF CRAMER'S RULE IS of (N+1)!

↑ N=20 => you boy is FUCKED!

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

already computed at previous steps

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

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:

we proceed with inverted order

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

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

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

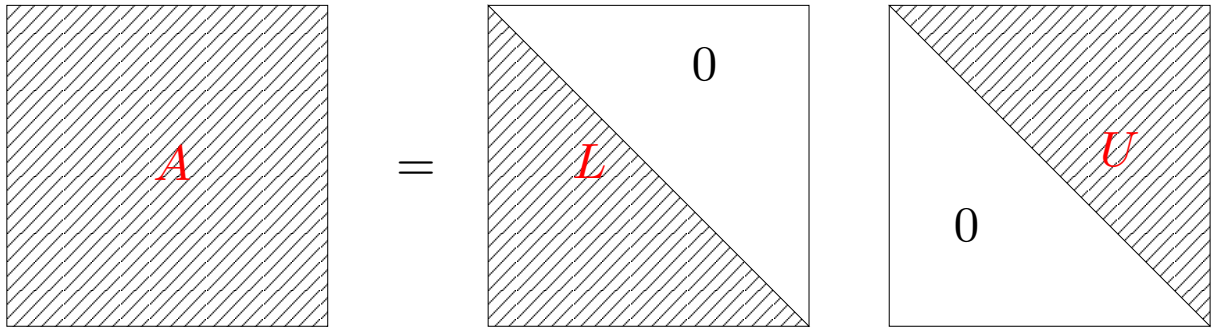
This algorithm is called *backward substitutions algorithm* The cost is, once again, n^2 operations.

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

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

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:

we have the diagonals of L and U

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

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:

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

If systems have also infinite solutions!

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

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

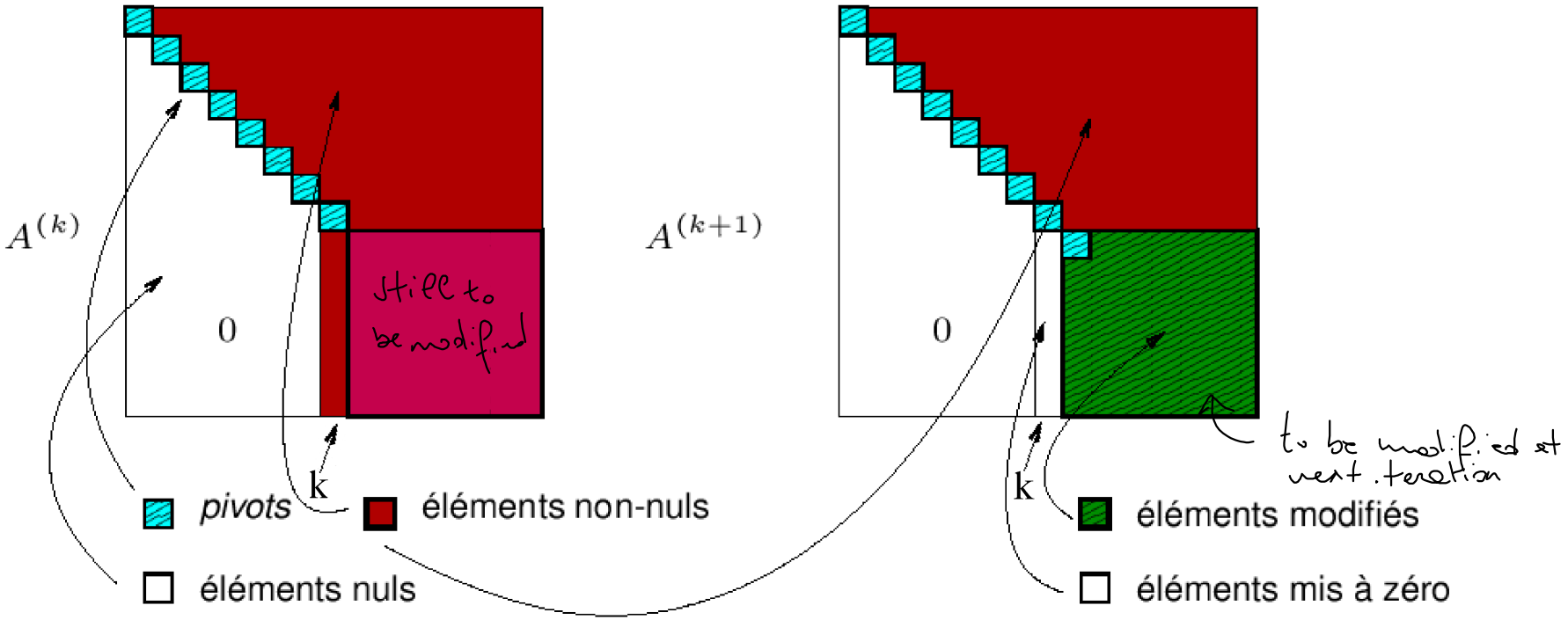
where, for $k \geq 2$, the matrix $A^{(k)}$ is of the form

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

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

↳ otherwise the multiplier gives division by zero

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

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

number of multiplies needed

we set then

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

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

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:

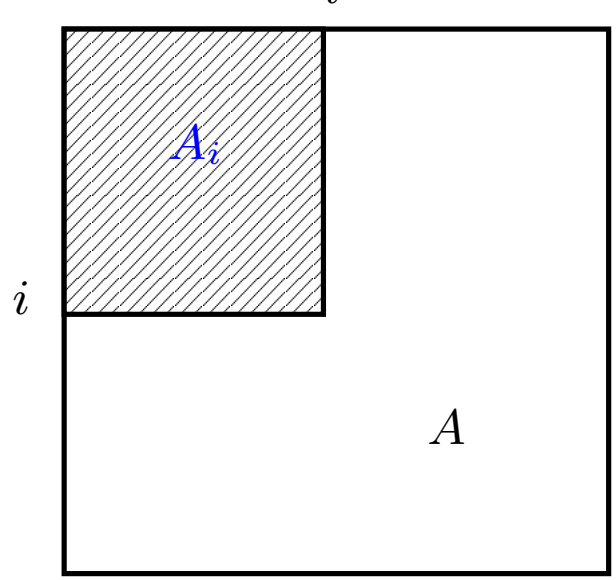
	Number of flops of the computer		
n	10^9 (Giga)	10^{12} (Tera)	10^{15} (Peta)
10^2	$7\ 10^{-4}$ sec	negligible	negligible
10^4	11 min	0.7 sec	$7\ 10^{-4}$ sec
10^6	21 years	7.7 months	11 min
10^8	o.r.	o.r.	21 years

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.:

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

$\geqslant \longrightarrow$ semidefinite positiveness

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 $U = A^{(n)}$.

↖ now we know how to find L

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.

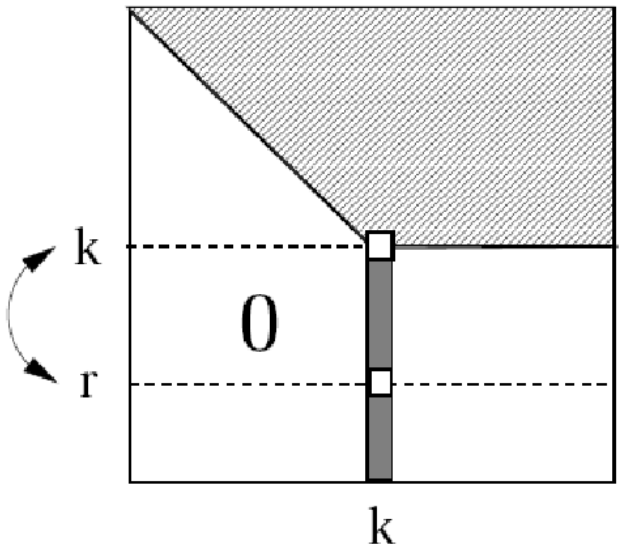
Example 5. Let us go back to the matrix (10) for which the Gauss method gives a null pivot at the second step. By just exchanging the second and the third rows, we get a non-zero pivot and can execute one step further. Indeed,

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

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

↑ if you exchange rows
⇒ $P_2 A$: PREMULT
if column
⇒ $A P_2$: POSTMULTIPLIC

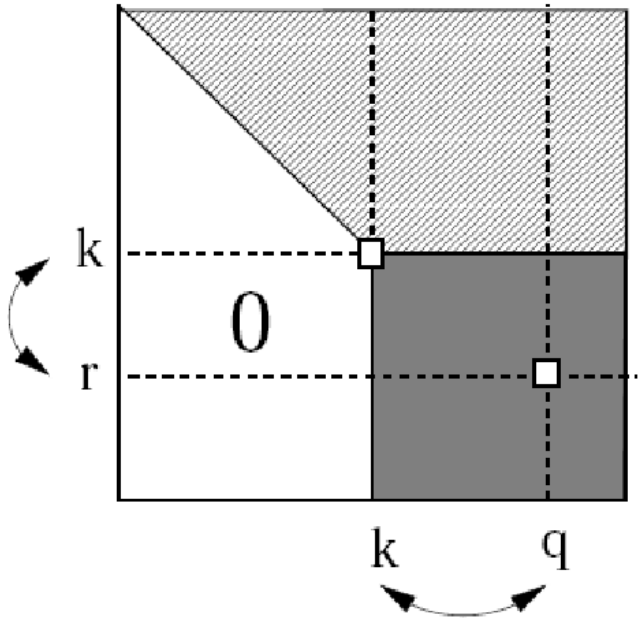
The pivoting strategy used for the example 5 can be generalized by finding, at every step k of the elimination, a non-zero pivot among the elements of the subcolumn $A^{(k)}(k:n,k)$. This is called a *partial pivot change* (by row).



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.

Full permutation <=> doubling of cost

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:

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

$$
\begin{array}{c}
\uparrow \\ r
\end{array}
\qquad
\begin{array}{c}
\uparrow \\ k
\end{array}
$$

to use on both sides of the system!

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 $LU = PA$.

P is not needed

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
```

LU fact gives you the possibility to store LU in smarter ways

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*

binet theorem

L has diagonal elements = 1

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

n operation

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 $1 \leq k \leq n$,

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

k-th column of identity

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.

requires only 1/2 memory of LU factorization (not 2 matrices but only 1 R matrix)

upper triangular matrix and symmetric

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
```

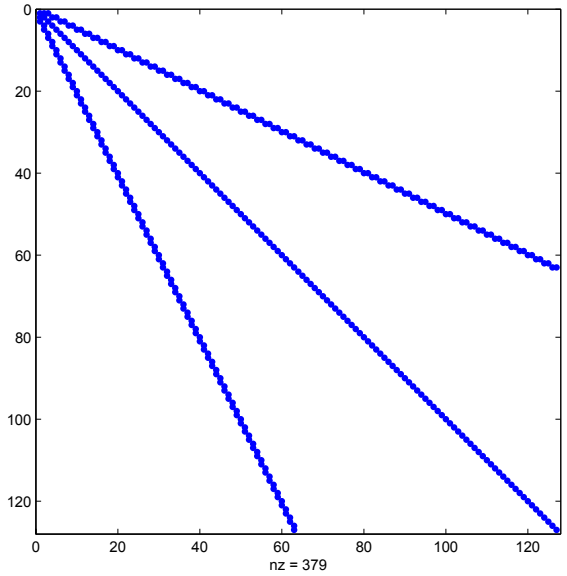
that grow proportionally to n^3. Indeed, if we calculate the ratio $fl(n)/n^3$ we find a "quasi" constant value.

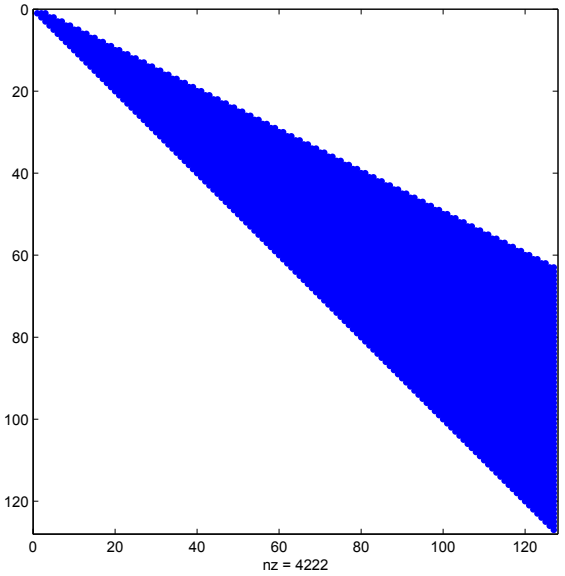
```
>> n=[10, 20, 30, 40, 50];
>> fl./(n.^3)
ans =
    0.3850    0.3588    0.3502    0.3459    0.3434
```

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

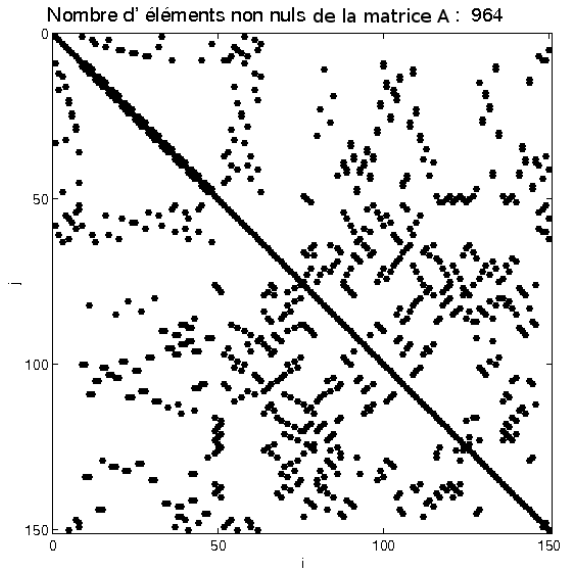
Example. 1 (continued) Let A be a matrix of size 127×127 corresponding to capillary bed with 8 bifurcation levels. This matrix is symmetric definite positive. The number of non-null entries of A is 379 and thus much smaller than $(127)^2 = 16129$. It is a *sparse* matrix. The figure on the left shows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

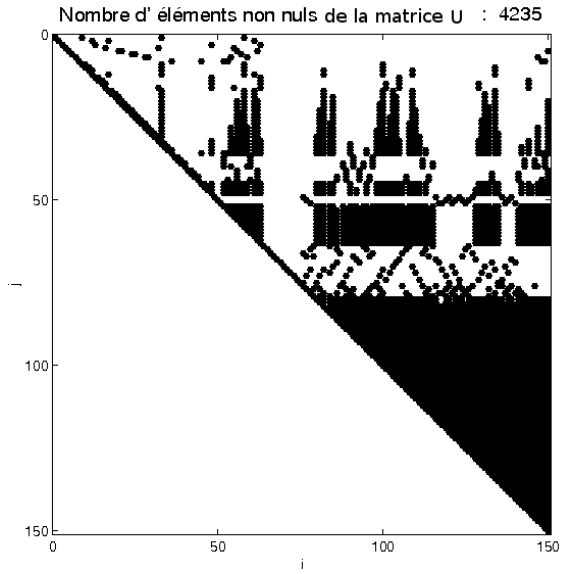




Example 7. Let us consider the problem of calculating the deformations in a structure subject to a given set of forces. The discretization using the finite elements method generates a matrix A of size 150×150. (The same matrix would have been produced by the approximation of an electric potential field.) 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.

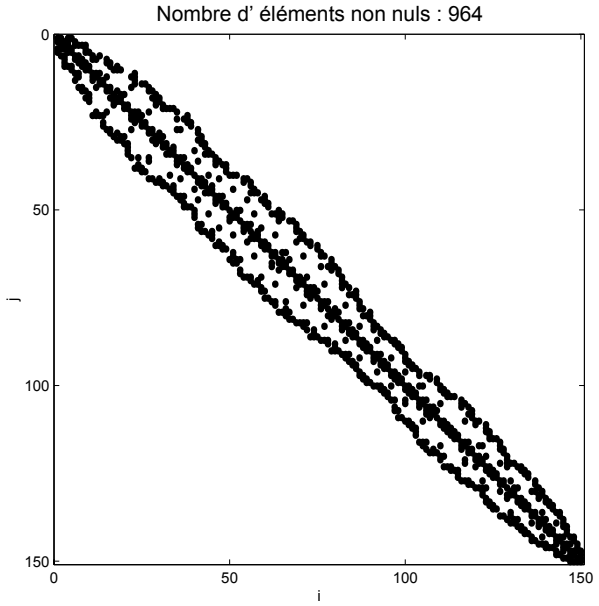
The figure on the left shows the disposition of the non-null entries of A, whereas the one on the right shows the non-null entries of the matrix R.

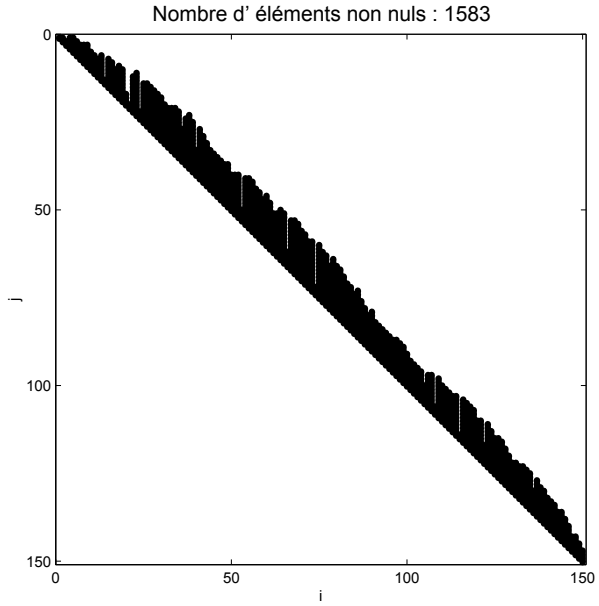




We notice that the number of non-null entries of R is much bigger than those of A (*fill-in phenomenon*). This leads to a bigger memory usage. To reduce the fill-in phenomenon, we can we can re-order rows and columns of A in a particular fashion; this is called *re-ordering* of the matrix. There are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





concentrating elements around diagonals gives stability

to be defined

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:

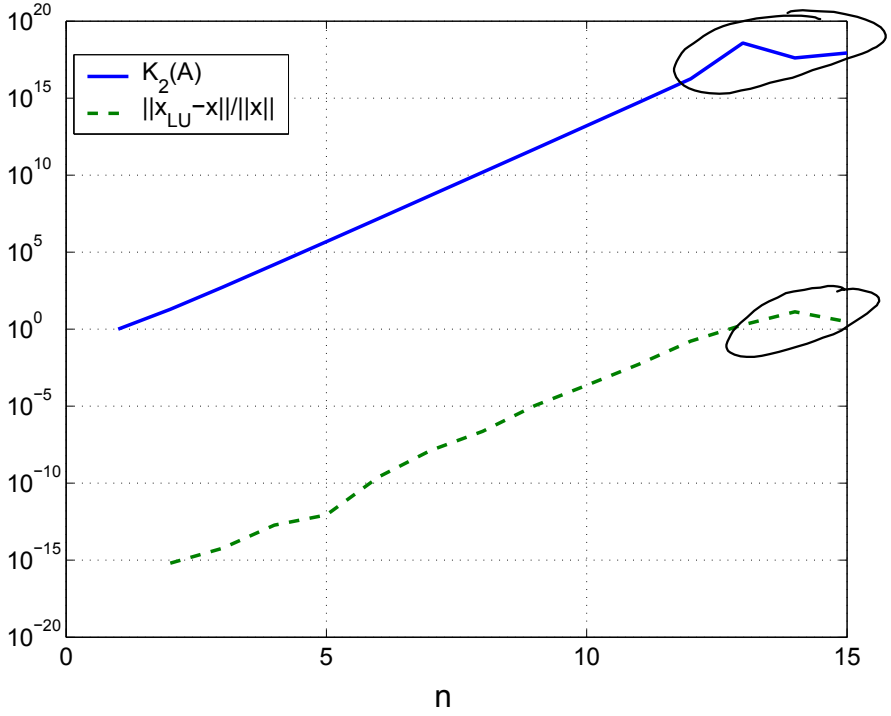
$$A_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,\ldots,n$$

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.



tells us matrix is "uncontrollable"

relative error in order of magnitude of the solution => it is NOT reliable

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.*

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

If h is too big => got shitty behavior

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 :

needs to be estimated because x is generally NOT known

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

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). ■

Linear Systems - Iterative Methods

Numerical Analysis

Profs. Gianluigi Rozza-Luca Heltai

2019-SISSA mathLab Trieste

(Sec. 5.9 of the book)

Solve the linear system $A\mathbf{x} = \mathbf{b}$ using an iterative method consists in building a series of vectors $\mathbf{x}^{(k)}$, $k \geq 0$, in $\mathbb{R}^n$ that converge at the exact solution $\mathbf{x}$, i.e.:

$$\lim_{k \to \infty} \mathbf{x}^{(k)} = \mathbf{x}$$

for any initial vector $\mathbf{x}^{(0)} \in \mathbb{R}^n$.
We can consider the following recurrence relation:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{g}, \quad k \geq 0 \tag{1}$$

where B is a well chosen matrix (depending on A) and $\mathbf{g}$ is a vector (that depends on A and $\mathbf{b}$), satisfying the relation (of consistence)

$$\mathbf{x} = B\mathbf{x} + \mathbf{g}. \tag{2}$$

Given $\mathbf{x} = A^{-1}\mathbf{b}$, we get $\mathbf{g} = (I - B)A^{-1}\mathbf{b}$; the iterative method is therefore completely defined by the matrix B, known as *iteration matrix*.
By defining the error at step k as

$$\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)},$$

error for each iteration
$(2) - (1)$

we obtain the following recurrence relation:

$$\mathbf{e}^{(k+1)} = B\mathbf{e}^{(k)} \quad \text{and thus} \quad \mathbf{e}^{(k+1)} = B^{k+1}\mathbf{e}^{(0)}, \quad k = 0, 1, \ldots.$$

We can show that $\lim_{k\to\infty} \mathbf{e}^{(k)} = \mathbf{0}$ for all $\mathbf{e}^{(0)}$ (and thus for all $\mathbf{x}^{(0)}$) if and only if

$$\boxed{\rho(B) < 1,}$$

or $\rho(B)$ is the *spectral radius* of the matrix B, defined by

$$\boxed{\rho(B) = \max |\lambda_i(B)|}$$

and $\lambda_i(B)$ are the eigenvalues of the matrix B.

The smaller the value of $\rho(B)$, the less iterations are needed to reduce the initial error of a given factor.

Construction of an iterative method

A general way of setting up an iterative method is based on the decomposition of the matrix A:

$$A = P - (P - A)$$

where P in an invertible matrix called *preconditioner* of A.
Hence,

$$A\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad P\mathbf{x} = (P - A)\mathbf{x} + \mathbf{b}$$

which is of the form (2) leaving

$$B = P^{-1}(P - A) = I - P^{-1}A \quad \text{and} \quad \mathbf{g} = P^{-1}\mathbf{b}.$$

We can define the corresponding iterative method

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{r}^{(k)} \qquad k \geq 0$$

where $\mathbf{r}^{(k)}$ represents the *residual* at the iteration k: $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$

We can generalise this method as follows:

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)} \qquad k \geq 0 \tag{3}$$

where $\alpha_k \neq 0$ is a parameter that improves the convergence of the series $\mathbf{x}^{(k)}$. The method (3) is called *Richardson's method*.

The matrix P has to be chosen in such a way that renders the cost of solving (3) small enough. For example a diagonal or triangular P matrix would comply with this criterion.

Jacobi method

If the elements of the diagonal of A are non-zero, we can write

$$P = D = diag(a_{11}, a_{22}, \ldots, a_{nn})$$

D with the diagonal part of A being:

$$D_{ij} = \begin{cases} 0 & \text{si } i \neq j \\ a_{ij} & \text{if } i = j. \end{cases}$$

The Jacobi method corresponds to this choice with $\alpha_k = 1$ for all k.
We deduce:

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (A - D)\mathbf{x}^{(k)} \qquad k \geq 0.$$

By components:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, \ldots, n. \tag{4}$$

NO DIAGONAL!

The Jacobi method can be written under the general form

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{g},$$

with

$$B = B_J = D^{-1}(D - A) = I - D^{-1}A, \qquad \mathbf{g} = \mathbf{g}_J = D^{-1}\mathbf{b}.$$

Gauss-Seidel method

This method is defined as follows:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, \ldots, n.$$

Take advantage also of partial x^{k+1} at current iteration already computed

This method corresponds to (1) with $P = D - E$ and $\alpha_k = 1$ ($\forall k \geq 0$) where E is the lower triangular matrix

D: diagonal; $-E$: lower triangular

$$\begin{cases} E_{ij} = -a_{ij} & \text{if } i > j \\ E_{ij} = 0 & \text{if } i \leq j \end{cases}$$

(lower triangular part of A without the diagonal and with its elements' sign inverted).

We can write this method under the form (3), with the iteration matrix $B = B_{GS}$ given by

$$B_{GS} = (D - E)^{-1}(D - E - A)$$

and

$$\mathbf{g}_{GS} = (D - E)^{-1}\mathbf{b}.$$

why not only E?
Library (historically)
builds matrix by block,
using also the diagonal

Example 1. Given the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}.$$

We have then

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 16 \end{pmatrix};$$

Thus, the iteration matrix for the Jacobi method is

Not symmetric

$$B_J = D^{-1}(D - A) = I - D^{-1}A = \begin{pmatrix} 0 & -2 & -3 & -4 \\ -5/6 & 0 & -7/6 & -4/3 \\ -9/11 & -10/11 & 0 & -12/11 \\ -13/16 & -14/16 & -15/16 & 0 \end{pmatrix}.$$

For defining the matrix A and extracting its diagonal D and its lower triangular part E (without the diagonal and the sign inverted) with Matlab/Octave, we use the commands

```
>> A = [1,2,3,4;5,6,7,8;9,10,11,12;13,14,15,16];
>> D = diag(diag(A));
>> E = - tril(A,-1);
```

These allow us, for exemple, to compute the iteration matrix B_{GS} for the Gauss-Seidel method in the following way:

```
>> B_GS = (D-E)\(D-E-A);
```

We find:

$$B_{GS} = \begin{pmatrix} 0.0000 & -2.0000 & -3.0000 & -4.000 \\ 0.0000 & 1.6667 & 1.3333 & 2.0000 \\ 0.0000 & 0.1212 & 1.2424 & 0.3636 \\ 0.0000 & 0.0530 & 0.1061 & 1.1591 \end{pmatrix}.$$

Convergence

We have the following convergence results:

- (Prop 5.3) If the matrix A is strictly diagonally dominant by row, i.e.,

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

  then the Jacobi and the Gauss-Seidel methods converge.

- If A is symmetric positive definite, then the Gauss-Seidel method converges (Jacobi maybe not).

- (Prop 5.4) Let A be a tridiagonal non-singular matrix whose diagonal elements are all non-null. Then the Jacobi and the Gauss-Seidel methods are either both divergent or both convergent. In the latter case, $\rho(B_{GS}) = \rho(B_J)^2$.

Non zero element only on the 3 biggest "diagonals"

(≤)

! For Jacobi and Gauss-Seidel $\alpha_k = 1$ (⇒ we are NOT accelerating)

Richardson method

(Sec. 5.10)

Let consider the following iterative method:

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)}, \qquad k \geq 0. \tag{5}$$

If $\alpha_k = \alpha$ (a constant) this method is called stationary preconditioned Richardson method; otherwise dynamic preconditioned Richardson method when α_k varies during the iterations.

The matrix P is called preconditioner of A.

If A and P are symmetric positive definite, then there are two optimal criteria to choose α_k:

1. *Stationary case*:

$$\alpha_k = \alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}}, \qquad k \geq 0,$$

← from algebraic and geometrical properties

where λ_{min} and λ_{max} represent the smaller and the larger eigenvalue of the matrix $P^{-1}A$.

2. *Dynamic case*:

$$\alpha_k = \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A \mathbf{z}^{(k)}}, \qquad k \geq 0,$$

This is a lin syst

where $\mathbf{z}^{(k)} = P^{-1}\mathbf{r}^{(k)}$ is the preconditioned residual. ← it is a TRF of the residual vector

This method is also called preconditioned gradient method.

If $P = I$ and A is symmetric definite positive, we get the following methods:

- the Stationary Richardson if we choose:

$$\alpha_k = \alpha_{opt} = \frac{2}{\lambda_{min}(A) + \lambda_{max}(A)}. \tag{6}$$

$P^{-1}A = A$

- the Gradient method if :

$$\alpha_k = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T A \mathbf{r}^{(k)}}, \qquad k \geq 0. \tag{7}$$

$P^{-1}r = r$

The gradient method can be written as:
Let $\mathbf{x}^{(0)}$ be given, set $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$, then for $k \geq 0$,

$$
\begin{aligned}
P\mathbf{z}^{(k)} &= \mathbf{r}^{(k)} \\
\alpha_k &= \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A\mathbf{z}^{(k)}} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k A\mathbf{z}^{(k)}.
\end{aligned}
$$

We have to apply once A and inverse P at each iteration. P should then be such that the resolution of the associated system results easy (i.e. it requires a reasonable amount of computing cost). For example, we can choose a diagonal P (Like in the gradient or the stationary Richardson cases) or triangular.

Convergence of Richardson method

When A and P are s.p.d. and with the two optimal choices for α, we can show that the preconditioned Richardson Method converges to x when $k \to \infty$, and that

$$\|\mathbf{x}^{(k)} - \mathbf{x}\|_A \leq \left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \right)^k \|\mathbf{x}^{(0)} - \mathbf{x}\|_A, \quad k \geq 0, \tag{8}$$

where $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A \mathbf{v}}$ and $K(P^{-1}A)$ is the condition number of $P^{-1}A$.

Remark If A et P are s.p.d., we have that

$$K(P^{-1}A) = \frac{\lambda_{max}}{\lambda_{min}}.$$

Demonstration The iteration matrix of the method is given by $R_\alpha = I - \alpha P^{-1} A$, where the eigenvalues of R_α are of the form $1 - \alpha\lambda_i$. The method is convergent if and only if $|1 - \alpha\lambda_i| < 1$ for $i = 1, \ldots, n$, therefore $-1 < 1 - \alpha\lambda_i < 1$ for $i = 1, \ldots, n$. As $\alpha > 0$, this is the equivalent to $-1 < 1 - \alpha\lambda_{max}$, from where the necessary and sufficient condition for convergence remains $\alpha < 2/\lambda_{max}$. Consequently, $\rho(R_\alpha)$ is minimal if $1 - \alpha\lambda_{min} = \alpha\lambda_{max} - 1$, i.e., for $\alpha_{opt} = 2/(\lambda_{min} + \lambda_{max})$. By substitution, we obtain

$$\rho_{opt} = \rho(R_{opt}) = 1 - \alpha_{opt}\lambda_{min} = 1 - \frac{2\lambda_{min}}{\lambda_{min} + \lambda_{max}} = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{min} + \lambda_{max}}$$

what allows us to complete the proof. □

In the dynamic case, we get a result that allows us to optimally choose the iteration parameters at each step, if the matrix A is symmetric definite positive:

Theorem 1 (Dynamic case)**.** *If A is symmetric definite positive, the optimal choice for α_k is given by*

$$\alpha_k = \frac{(\mathbf{r}^{(k)}, \mathbf{z}^{(k)})}{(A\mathbf{z}^{(k)}, \mathbf{z}^{(k)})}, \qquad k \geq 0 \tag{9}$$

where

$$\mathbf{z}^{(k)} = P^{-1}\mathbf{r}^{(k)}. \tag{10}$$

Demonstration On the one hand we have

sometimes not present in literature

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} = A(\mathbf{x} - \mathbf{x}^{(k)}) = A\mathbf{e}^{(k)}, \tag{11}$$

and thus, using (10),

$$P^{-1}A\mathbf{e}^{(k)} = -\mathbf{z}^{(k)}, \tag{12}$$

where $\mathbf{e}^{(k)}$ represents the error at the step k. On the other hand

$$\mathbf{e}^{(k+1)} = \mathbf{e}^{(k+1)}(\alpha) = \underbrace{(I - \alpha P^{-1}A)}_{R_\alpha}\mathbf{e}^{(k)}.$$

some kind of a (static or dynamic)

We notice that, in order to update the residual, we have the relation

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha A\mathbf{z}^{(k)} = \mathbf{r}^{(k)} - \alpha AP^{-1}\mathbf{r}^{(k)}.$$

Thus, expressing as $\|\cdot\|_A$ the vector norm associated to the scalar product $(\mathbf{x},\mathbf{y})_A = (A\mathbf{x},\mathbf{y})$, what means, $\|\mathbf{x}\|_A = (A\mathbf{x},\mathbf{x})^{1/2}$ we can write

Remember: A is symmetric

$$\begin{aligned}
\|\mathbf{e}^{(k+1)}\|_A^2 &= (A\mathbf{e}^{(k+1)},\mathbf{e}^{(k+1)}) = -(\mathbf{r}^{(k+1)},\mathbf{e}^{(k+1)}) \\
&= -(\mathbf{r}^{(k)} - \alpha AP^{-1}\mathbf{r}^{(k)}, \mathbf{e}^{(k)} - \alpha P^{-1}A\mathbf{e}^{(k)}) \\
&= -(\mathbf{r}^{(k)},\mathbf{e}^{(k)}) + \alpha[(\mathbf{r}^{(k)}, P^{-1}A\mathbf{e}^{(k)}) + (A\mathbf{z}^{(k)},\mathbf{e}^{(k)})] \\
&\quad -\alpha^2(A\mathbf{z}^{(k)}, P^{-1}A\mathbf{e}^{(k)})
\end{aligned}$$

Now we choose α as the α_k that minimises $\|\mathbf{e}^{(k+1)}(\alpha)\|_A$:

$$\frac{d}{d\alpha}\|\mathbf{e}^{(k+1)}(\alpha)\|_A\Big|_{\alpha=\alpha_k} = 0$$

We then obtain

$$\alpha_k = \frac{1}{2}\frac{(\mathbf{r}^{(k)}, P^{-1}A\mathbf{e}^{(k)}) + (A\mathbf{z}^{(k)}, \mathbf{e}^{(k)})}{(A\mathbf{z}^{(k)}, P^{-1}A\mathbf{e}^{(k)})} = \frac{1}{2}\frac{-(\mathbf{r}^{(k)}, \mathbf{z}^{(k)}) + (A\mathbf{z}^{(k)}, \mathbf{e}^{(k)})}{-(A\mathbf{z}^{(k)}, \mathbf{z}^{(k)})}$$

and using the equality $(A\mathbf{z}^{(k)}, \mathbf{e}^{(k)}) = (\mathbf{z}^{(k)}, A\mathbf{e}^{(k)})$ knowing that A is symmetric definite positive, and noting that $A\mathbf{e}^{(k)} = -\mathbf{r}^{(k)}$, we find

$$\alpha_k = \frac{(\mathbf{r}^{(k)}, \mathbf{z}^{(k)})}{(A\mathbf{z}^{(k)}, \mathbf{z}^{(k)})}$$

For the stationary case and for the dynamic one we can prove that, if A and P are symmetric definite positive, the series $\{\mathbf{x}^{(k)}\}$ given by the Richardson method (stationary and dynamic) converges towards $\mathbf{x}$ when $k \to \infty$, and

$$\|\mathbf{x}^{(k)} - \mathbf{x}\|_A \leq \left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \right)^k \|\mathbf{x}^{(0)} - \mathbf{x}\|_A, \quad k \geq 0, \tag{13}$$

Again, K should make sense $\leftrightarrow$ A should be stable

where $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A \mathbf{v}}$ and $K(P^{-1}A)$ is the conditioning of the matrix $P^{-1}A$.

Remark. In the case of the gradient method or the Richardson stationary method the error estimation becomes

$$\|\mathbf{x}^{(k)} - \mathbf{x}\|_A \leq \left(\frac{K(A)-1}{K(A)+1}\right)^k \|\mathbf{x}^{(0)} - \mathbf{x}\|_A, \quad k \geq 0. \tag{14}$$

Remark. If A and P are symmetric definite positive, we have

$$K(P^{-1}A) = \frac{\lambda_{max}(P^{-1}A)}{\lambda_{min}(P^{-1}A)}.$$

The conjugate gradient method

(Sec. 5.11)

When A and P are s.p.d, there exists a very efficient and effective method to iteratively solve the system: the conjugate gradient method

Let $\mathbf{x}^{(0)}$ be given; we compute $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$, $\mathbf{z}^{(0)} = P^{-1}\mathbf{r}^{(0)}$, $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$,then for $k \geq 0$,

$$
\begin{aligned}
\alpha_k &= \frac{{\mathbf{p}^{(k)}}^T \mathbf{r}^{(k)}}{{\mathbf{p}^{(k)}}^T A\mathbf{p}^{(k)}} \\
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)} \\
\mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k A\mathbf{p}^{(k)} \\
P\mathbf{z}^{(k+1)} &= \mathbf{r}^{(k+1)} \\
\beta_k &= \frac{\left(A\mathbf{p}^{(k)}\right)^T \mathbf{z}^{(k+1)}}{\left(A\mathbf{p}^{(k)}\right)^T \mathbf{p}^{(k)}} \\
\mathbf{p}^{(k+1)} &= \mathbf{z}^{(k+1)} - \beta_k \mathbf{p}^{(k)} .
\end{aligned}
$$

Now we correct also the DIRECTION. correction given by $p^{(k)}$

given as A conjugate (A orthogonal) direction wrt the direction $p^{(k-1)}$

special directions are NOT so, but only in number = to order of A matrix

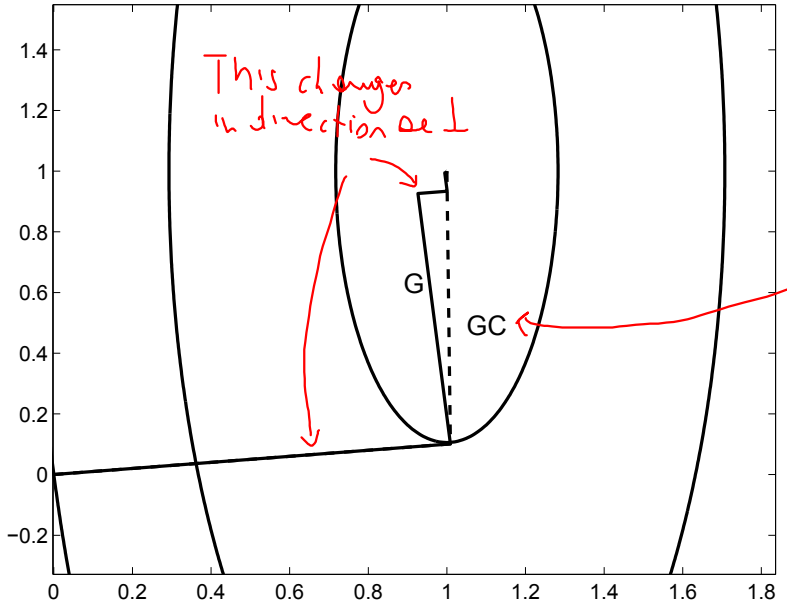
The error estimate is given by

$$\|\mathbf{x}^{(k)} - \mathbf{x}\|_A \leq \frac{2c^k}{1+c^{2k}} \|\mathbf{x}^{(0)} - \mathbf{x}\|_A , \quad k \geq 0$$

où

$$c = \frac{\sqrt{K_2(P^{-1}A)} - 1}{\sqrt{K_2(P^{-1}A)} + 1} .$$

(15)



we reach solution without changing direction because we already do it in the procedure

Example 2. Let consider the following linear system:

$$\begin{cases} 2x_1 + x_2 &= 1 \\ x_1 + 3x_2 &= 0 \end{cases} \tag{16}$$

whose matrix is $A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$ is s.p.d. The solution to this system is $x_1 = 3/5 = 0.6$ et $x_2 = -1/5 = -0.2$.

Preliminary convergence studies

- A is strictly diagonal dominant by row. Hence Jacobi and Gauss-Seidel methods converge.
- A is regular, tridiagonal with non-zero diagonal elements. Then $\rho(B_{GS}) = \rho(B_J)^2$. Therefore we expect a quicker convergence of Gauss-Seidel w.r.t. Jacobi.
- A is s.p.d., hence the gradient and the conjugate gradient methods converge. Moreover (see error estimates), the CG shall converge faster.

We want to approximate the solution with an iterative method starting with

$$\mathbf{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}.$$

We can see that

$$\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{pmatrix} -\frac{3}{2} \\ -\frac{5}{2} \end{pmatrix}$$

and

$$\|\mathbf{r}^{(0)}\|_2 = \sqrt{(\mathbf{r}^{(0)})^T \mathbf{r}^{(0)}} = \frac{\sqrt{34}}{2} \approx 2.9155.$$

Jacobi method

$$\mathbf{x}^{(k+1)} = B_J \mathbf{x}^{(k)} + \mathbf{g}_J, \quad k \geq 0, \qquad \text{where } B_J = I - D^{-1}A \text{ and } \mathbf{g}_J = D^{-1}\mathbf{b}.$$

We have

$$B_J = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{3} & 0 \end{pmatrix}$$

$$\mathbf{g}_J = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix}$$

and $\rho(B_J) = max|\lambda_i(B_J)| = \texttt{max(abs(eig(}B_J\texttt{)))} = 0.4082.$

For $k = 0$ (first iteration) we find:

$$\mathbf{x}^{(1)} = B_J \mathbf{x}^{(0)} + \mathbf{g}_J = \begin{pmatrix} 0 & -\frac{1}{2} \\ -\frac{1}{3} & 0 \end{pmatrix} \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ -\frac{1}{3} \end{pmatrix} \approx \begin{pmatrix} 0.25 \\ -0.3333 \end{pmatrix}.$$

Notice that

$$\mathbf{r}^{(1)} = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{pmatrix} 0.8333 \\ 0.75 \end{pmatrix} \qquad \text{and } \|\mathbf{r}^{(1)}\|_2 = 1.1211.$$

Gauss-Seidel method

$$\mathbf{x}^{(k+1)} = B_{GS}\mathbf{x}^{(k)} + \mathbf{g}_{GS}, \quad k \geq 0, \quad \text{where } B_{GS} = (D-E)^{-1}(D-E-A)$$

$$\text{and } \mathbf{g}_{GS} = (D-E)^{-1}\mathbf{b}.$$

We have

$$B_{GS} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{1}{2} \\ 0 & \frac{1}{6} \end{pmatrix}$$

$$\mathbf{g}_{GS} = \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{6} \end{pmatrix}$$

In this case $\rho(B_{GS}) = max|\lambda_i(B_{GS})| =$ `max(abs(eig(`B_{GS}`)))` $= 0.1667$.
We can verify that $\rho(B_{GS}) = \rho(B_J)^2$.

For $k = 0$ (first iteration) we find:

$$\mathbf{x}^{(1)} = B_{GS}\mathbf{x}^{(0)} + \mathbf{g}_{GS} = \begin{pmatrix} 0 & -\frac{1}{2} \\ 0 & \frac{1}{6} \end{pmatrix} \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{6} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ -\frac{1}{12} \end{pmatrix} \approx \begin{pmatrix} 0.25 \\ -0.0833 \end{pmatrix}.$$

We have

$$\mathbf{r}^{(1)} = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{pmatrix} 0.5833 \\ 0 \end{pmatrix} \qquad \text{and } \|\mathbf{r}^{(1)}\|_2 = 0.5833.$$

Preconditioned gradient method with $P = D$

We set $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} -\frac{3}{2} \\ -\frac{5}{2} \end{pmatrix}$.

For $k = 0$, we have:

$$P\mathbf{z}^{(0)} = \mathbf{r}^{(0)} \quad \Leftrightarrow \quad \mathbf{z}^{(0)} = P^{-1}\mathbf{r}^{(0)} = \begin{pmatrix} -\frac{3}{4} \\ -\frac{5}{6} \end{pmatrix}$$

$$\alpha_0 = \frac{(\mathbf{z}^{(0)})^T \mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T A\mathbf{z}^{(0)}} = \frac{77}{107}$$

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{z}^{(0)} = \begin{pmatrix} 0.4603 \\ -0.0997 \end{pmatrix}$$

$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha_0 A\mathbf{z}^{(0)} = \begin{pmatrix} 0.1791 \\ -0.1612 \end{pmatrix} \quad \text{and } \|\mathbf{r}^{(1)}\|_2 = 0.2410.$$

Conjugated preconditioned gradient method with $P = D$

We set $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$, $\mathbf{z}^{(0)} = P^{-1}\mathbf{r}^{(0)}$ and $\mathbf{p}^{(0)} = \mathbf{z}^{(0)}$. For $k = 0$, we have:

$$\alpha_0 = \frac{(\mathbf{p}^{(0)})^T\mathbf{r}^{(0)}}{(\mathbf{p}^{(0)})^T A\mathbf{p}^{(0)}} = \frac{(\mathbf{z}^{(0)})^T\mathbf{r}^{(0)}}{(\mathbf{z}^{(0)})^T A\mathbf{z}^{(0)}}$$

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0\mathbf{p}^{(0)} = \mathbf{x}^{(0)} + \alpha_0\mathbf{z}^{(0)}$$

$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha_0 A\mathbf{p}^{(0)} = \mathbf{r}^{(0)} - \alpha_0 A\mathbf{z}^{(0)}.$$

We see that the first iteration $\mathbf{x}^{(1)}$ matches with the one obtained by the preconditioned gradient method.

We then complete the first iteration of the preconditioned conjugate gradient method:

$$P\mathbf{z}^{(1)} = \mathbf{r}^{(1)} \quad \Leftrightarrow \quad \mathbf{z}^{(1)} = P^{-1}\mathbf{r}^{(1)} = \begin{pmatrix} 0.0896 \\ -0.0537 \end{pmatrix}$$

$$\beta_0 = \frac{(A\mathbf{p}^{(0)})^T\mathbf{z}^{(1)}}{(A\mathbf{p}^{(0)})^T A\mathbf{p}^{(0)}} = \frac{(A\mathbf{z}^{(0)})^T\mathbf{z}^{(1)}}{(A\mathbf{z}^{(0)})^T\mathbf{z}^{(0)}} = -0.0077$$

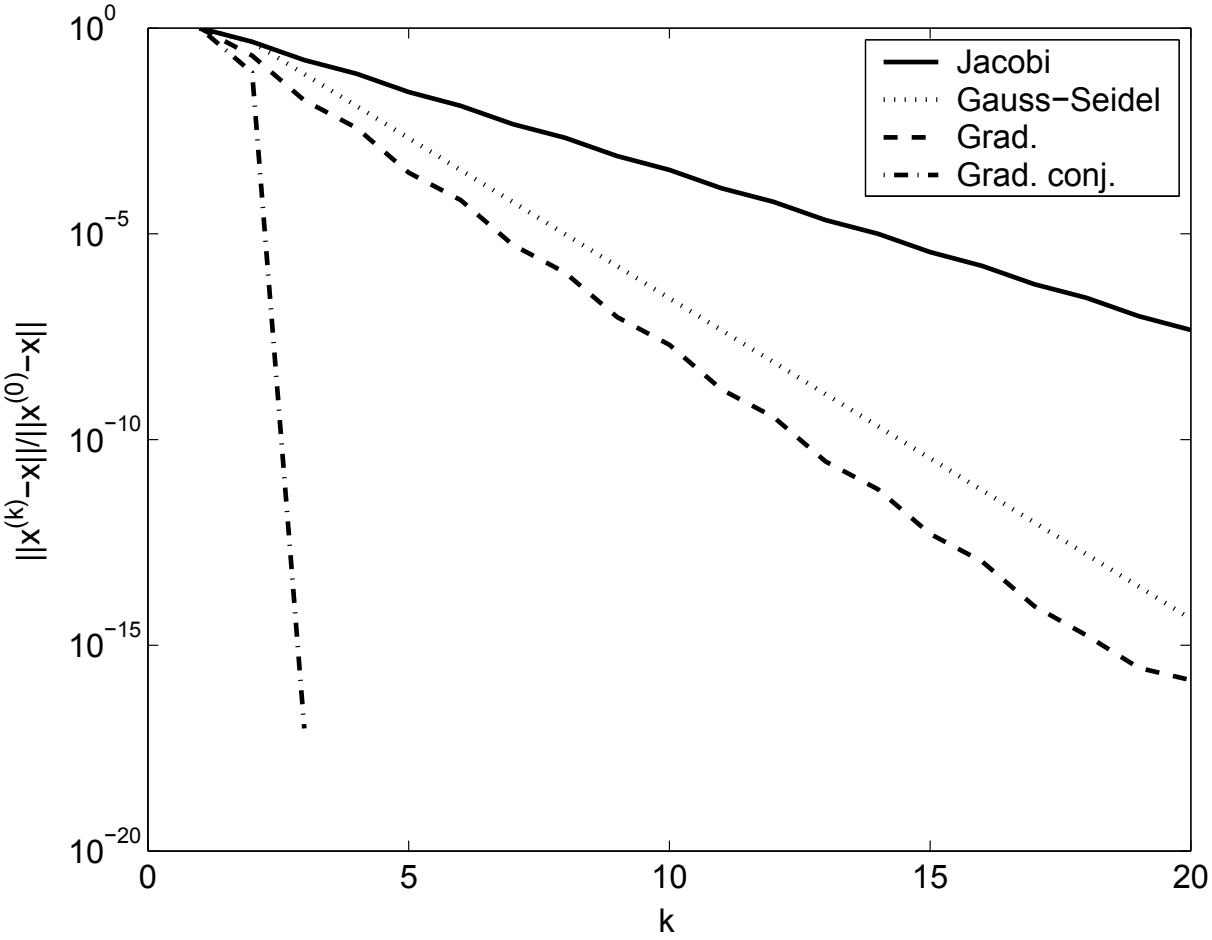
$$\mathbf{p}^{(1)} = \mathbf{z}^{(1)} - \beta_0\mathbf{p}^{(0)} = \mathbf{z}^{(1)} - \beta_0\mathbf{z}^{(0)} = \begin{pmatrix} 0.0838 \\ -0.0602 \end{pmatrix}.$$

At the second iteration, with the four different methods, we have:

Method	$\mathbf{x}^{(2)}$	$\mathbf{r}^{(2)}$	$\|\mathbf{r}^{(2)}\|_2$
Jacobi	$\begin{pmatrix} 0.6667 \\ -0.0833 \end{pmatrix}$	$\begin{pmatrix} -0.2500 \\ -0.4167 \end{pmatrix}$	0.4859
Gauss-Seidel	$\begin{pmatrix} 0.5417 \\ -0.1806 \end{pmatrix}$	$\begin{pmatrix} 0.0972 \\ 0 \end{pmatrix}$	0.0972
PG	$\begin{pmatrix} 0.6070 \\ -0.1877 \end{pmatrix}$	$\begin{pmatrix} -0.0263 \\ -0.0438 \end{pmatrix}$	0.0511
PCG	$\begin{pmatrix} 0.60000 \\ -0.2000 \end{pmatrix}$	$\begin{pmatrix} -0.2220 \\ -0.3886 \end{pmatrix} \cdot 10^{-15}$	$4.4755 \cdot 10^{-16}$

Behavior of the relative error applied to the system (16) :

Graph important for the exam



Example 3. Let now consider another example:

$$\begin{cases} 2x_1 + x_2 & = 1 \\ -x_1 + 3x_2 & = 0 \end{cases} \tag{17}$$

whose solution is $x_1 = 3/7$, $x_2 = 1/7$.

Preliminary convergence studies

The associatd matrix is $A = \begin{pmatrix} 2 & 1 \\ -1 & 3 \end{pmatrix}$. ← not symmetric => not positive definite

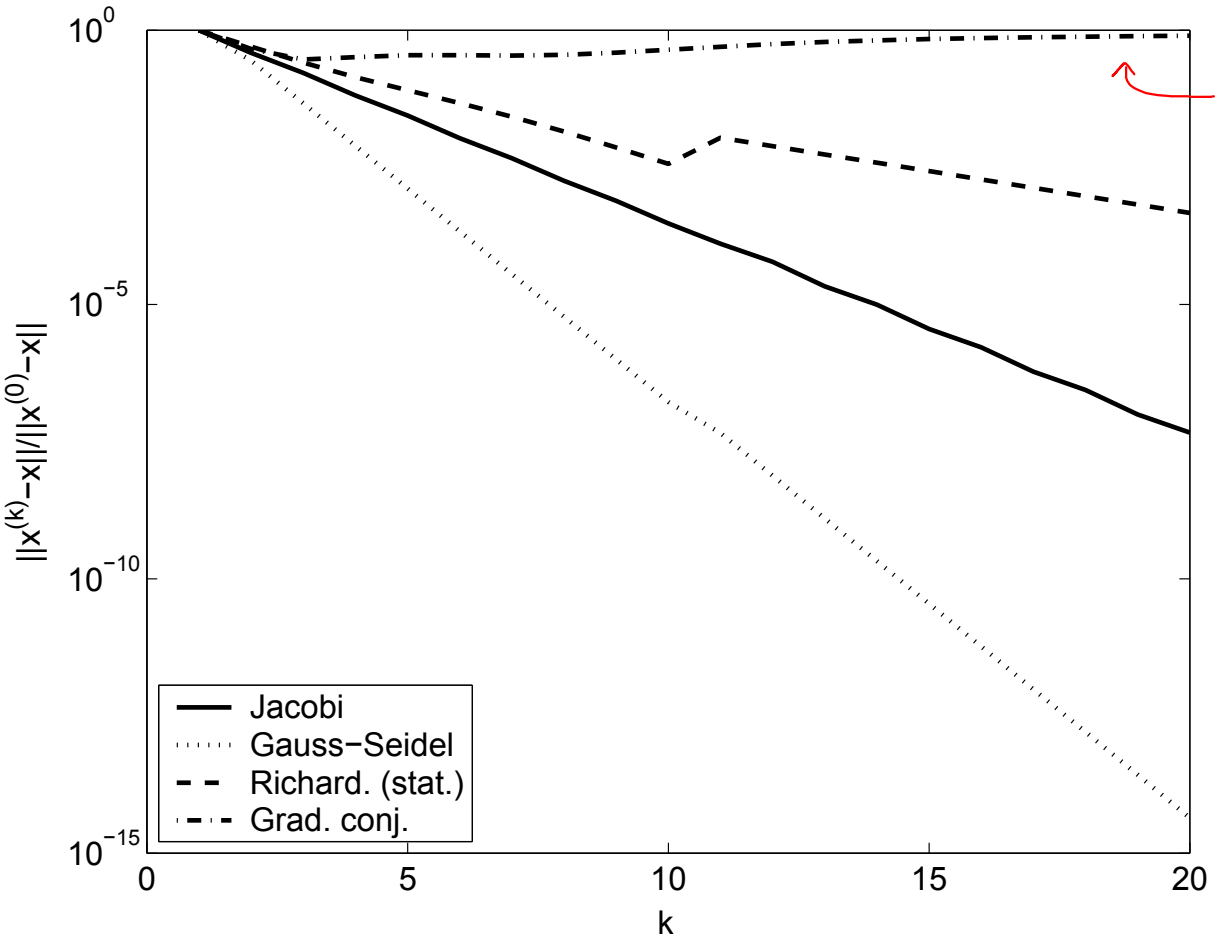
- A is strictly diagonal dominant by row. Hence Jacobi and Gauss-Seidel methods converge.
- A is regular, tridiagonal with non-zero diagonal elements. Then $\rho(B_{GS}) = \rho(B_J)^2$. Therefore we aspect a quicker convergence of Gauss-Seidel w.r.t. Jacobi.
- A **is not s.p.d.**, therefore we have no idea if the gradient or the conjugate gradient converge.

We approximate the solution with an iterative method starting from

$$\mathbf{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix}.$$

The following figure shows the value of $\frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|}{\|\mathbf{x}^{(0)} - \mathbf{x}\|}$ for the Jacobi, Gauss-Seidel, Richardson stationary (preconditioned with $\alpha = 0.5$ and $P = D = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$), and the preconditioned (with $P = D$) conjugate gradient methods.
Remark that this time the preconditioned conjugate gradient method doesn't converge.

Behavior of the relative error applied to the system (17) :



why? we are applying wrong methodology, since A IS NOT SYMMETRIC

when this happens use other methods, not conjugate gradient

Convergence Criteria

(Sec. 5.12)

We have the following error bound:

If A is s.p.d, then

$$\frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(A) \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|}. \tag{18}$$

The relative error at the iteration k is bounded by the condition number of A times the residual scaled with the right hand side.

We can also use another relation in case of a preconditioned system:

$$\frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq K(P^{-1}A) \frac{\|P^{-1}\mathbf{r}^{(k)}\|}{\|P^{-1}\mathbf{b}\|}.$$

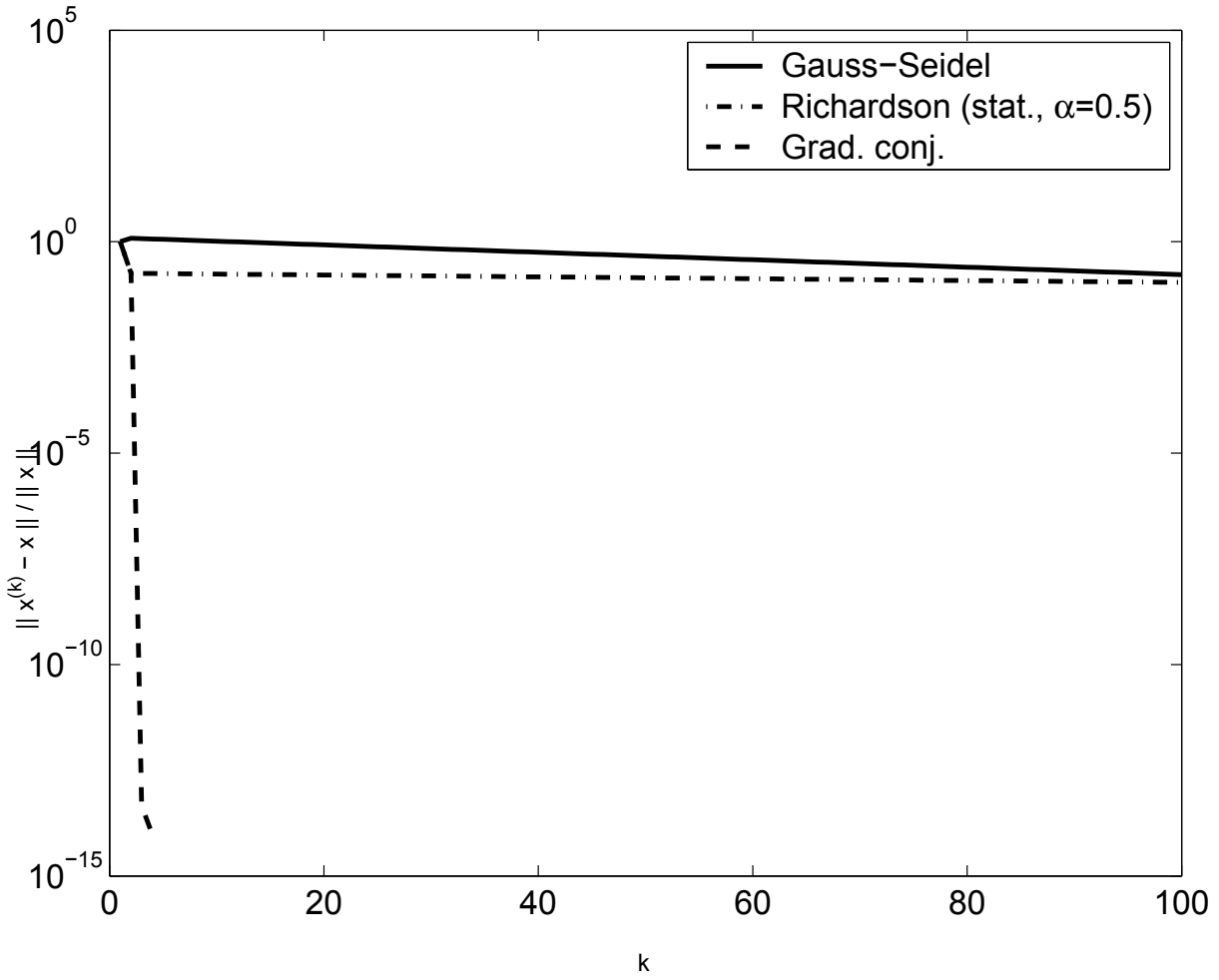
Some examples of convergence of iterative methods applied to linear systems of the kind $A\mathbf{x} = \mathbf{b}$.

Example 4. Lets start with the matrix

$$A = \begin{pmatrix} 5 & 7 \\ 7 & 10 \end{pmatrix} \tag{19}$$

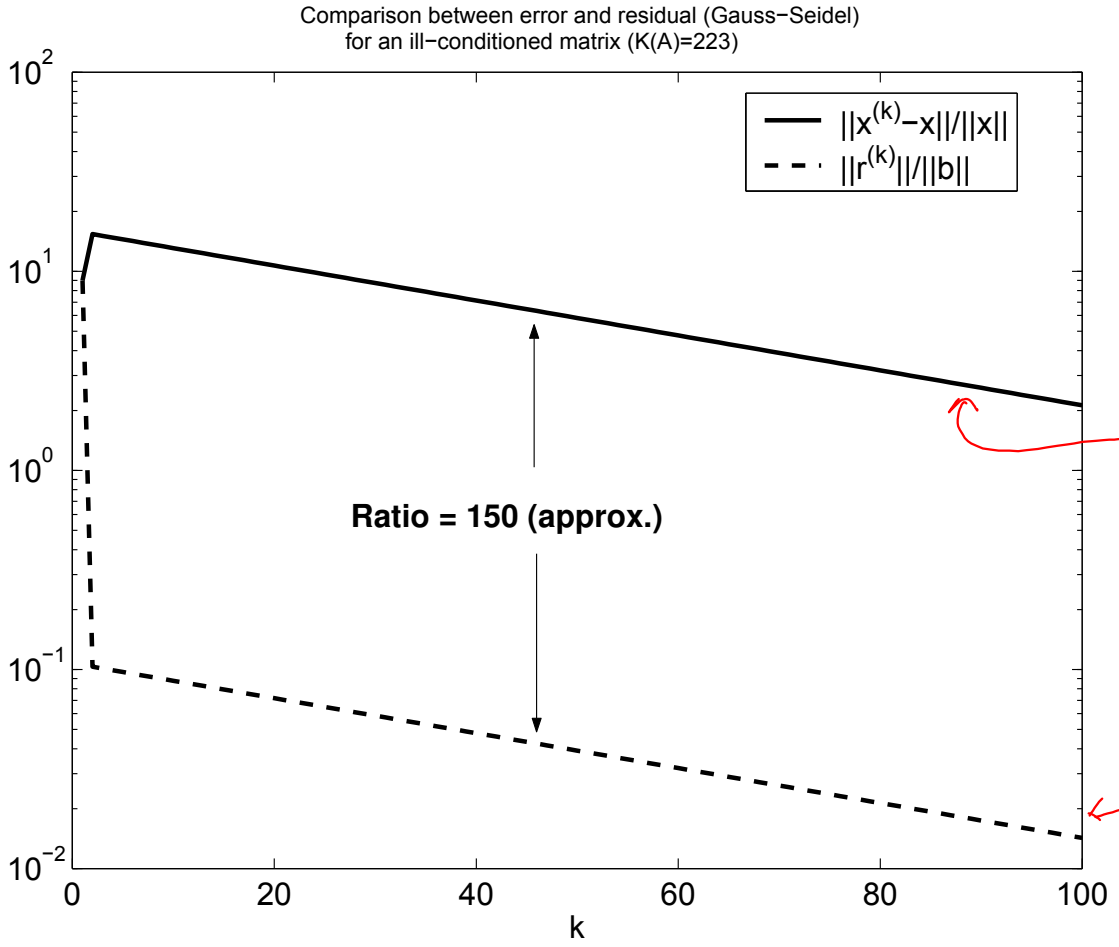
The condition number of this matrix is $K(A) \approx 223$. We consider the Gauss-Seidel method and stationary preconditioned Richardson method with $\alpha = 0.5$ and $P = \text{diag}(A)$. We have that $\rho(B_{GS}) = 0.98$ and $\rho(B_{Rich}) = 0.98$, where $B_{Rich} = I - \alpha P^{-1} A$ is the iterative matrix for the stationary Richardson method.

This figure shows the relative error behavior for Gauss-Seidel method and stationary preconditioned Richardson method and the cojugate gradient preconditioned with the same matrix



100 its for 2 x 2 matrix is UNACCEPTABLE

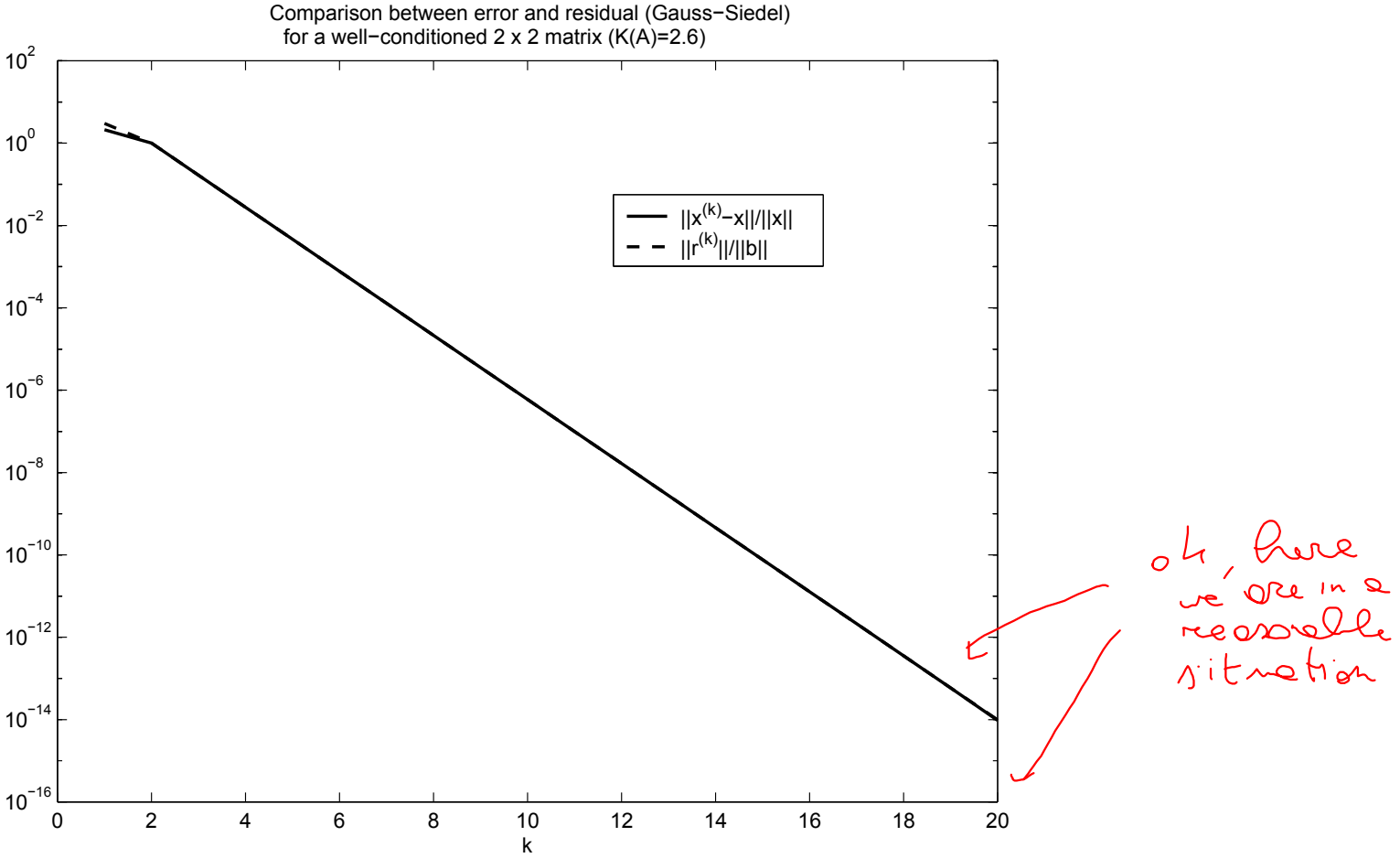
Comparison between error and residual (Gauss-Seidel) (19). Recall that $K(A) \approx 223$



LOOK AT THE 00 ORDER OF MAGNITUDE! But often this line is not available

If a metrix 2×2 requires 100 its should tell me sth is wrong

We can compare the previous figure with the same curves for well conditioned matrix 2×2 ($K(A) \approx 2.6$):



Comparison between the relative error and the residual, for a well conditioned matrix.

Example 5. We take the Hilbert matrix and we solve a linear system for different size n. We set P =diag(A) and use the preconditioned gradient method. The stoping tolerance is set to 10^-6 on the relative residual. We take $\mathbf{x}^{(0)} = \mathbf{0}$.
We evaluate the relative error

```
for j=2:7;
     n=2*j; i=j-1; nn(i)=n;
     A=hilb(n);
     x_ex=ones(n,1); b=A*x_ex;
     Acond(i)=cond(A);
     tol=1.e-6; maxit=10000;
     R=diag(diag(A));
     x0=zeros(n,1);
     [x,iter_gr(i)]=gradient(A,b,x0,maxit,tol,R);
     error_gr(i)=norm(x-x_ex)/norm(x_ex);
   end
```

For the iterative method, we set a tolerance of 10^{-6} on the relative residual. Because the matrix is ill conditioned, it is to be expected to get a relative error in the solution greater than 10^{-6} (see inequality (18)).

		Gradient method		
n	$K(A)$	Error	Iterations	Residual
4	1.55e+04	8.72e-03	995	1.00e-06
6	1.50e+07	3.60e-03	1813	9.99e-07
8	1.53e+10	6.30e-03	1089	9.96e-07
10	1.60e+13	7.99e-03	875	9.99e-07
12	1.67e+16	5.09e-03	1355	9.99e-07
14	2.04e+17	3.91e-03	1379	9.98e-07

Already a WARNING

4 OD of magnitude greater than residual → could lead to very wrong assumption on the problem you are trying to solve

Some observations

Memory and computational costs

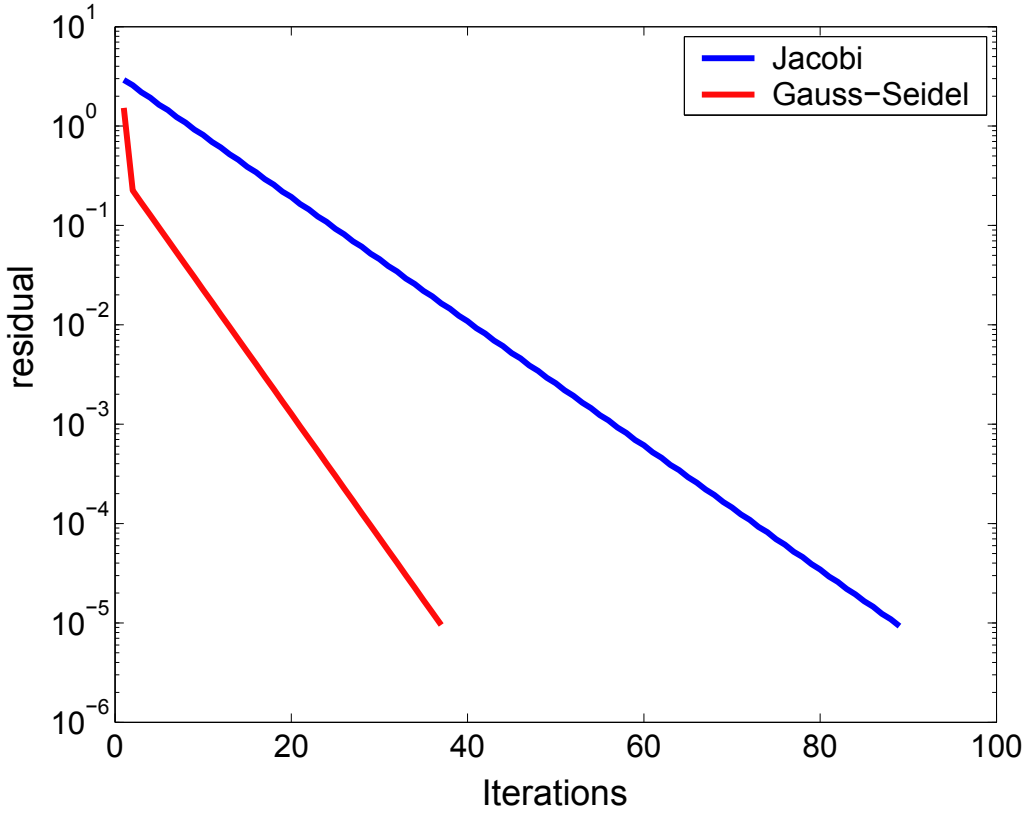
Computational cost (*flops*) and used memory (*bytes*): we consider the Cholesky an the conjugate gradient methods for sparse matrices of size n (originated in the approximation of solutions to the Poisson equation in the finite elements method) with m non zero elements.

		Cholesky		Conjugate gradient		flops(Chol.)/	Mem(Chol.)/
n	m/n^2	flops	Memory	flops	Memory	flops(GC)	Mem(GC)
47	0.12	8.05e+03	464	1.26e+04	228	0.64	2.04
83	0.07	3.96e+04	1406	3.03e+04	533	1.31	2.64
150	0.04	2.01e+05	4235	8.86e+04	1245	2.26	3.4
225	0.03	6.39e+05	9260	1.95e+05	2073	3.27	4.47
329	0.02	1.74e+06	17974	3.39e+05	3330	5.15	5.39
424	0.02	3.78e+06	30815	5.49e+05	4513	6.88	6.83
530	0.01	8.31e+06	50785	8.61e+05	5981	9.65	8.49
661	0.01	1.19e+07	68468	1.11e+06	7421	10.66	9.23

conjugate gradient saves 10 times the bytes and flops than the cholesky method

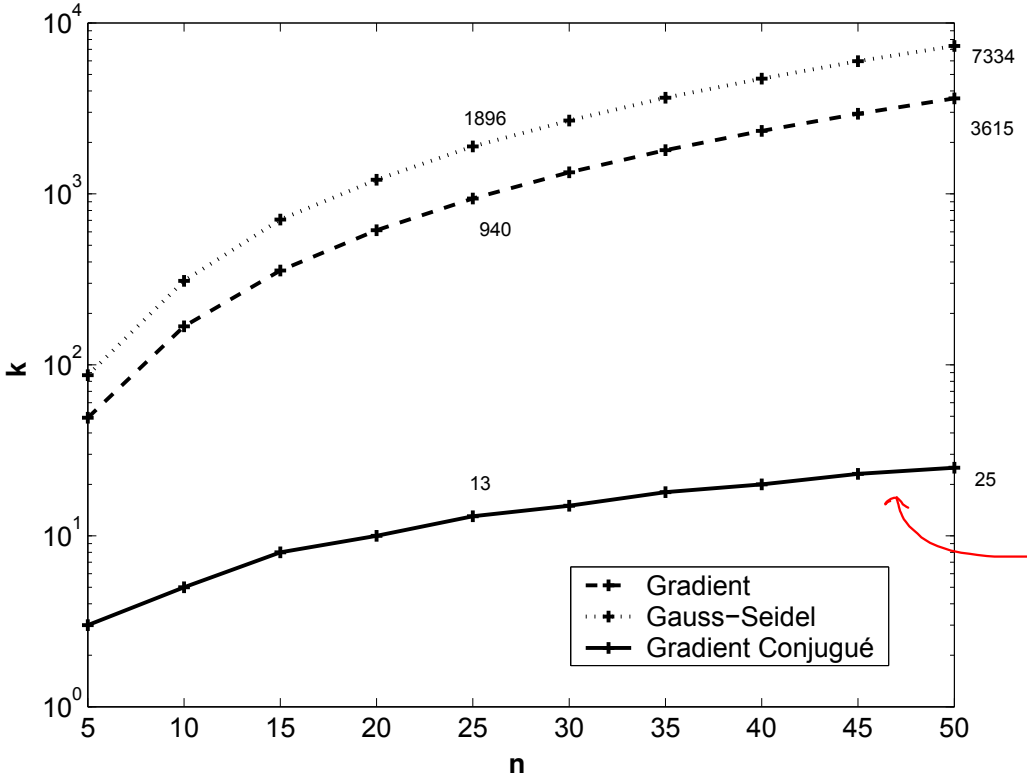
Iterative methods: Jacobi, Gauss-Seidel

Behaviour of the error for a well conditioned matrix ($K = 20$)



Iterative methods: G-S, Gradient, Gradient Conjugate

Number of iterations as function of the size n of a stiffness matrix, tolerance 10^{-6}

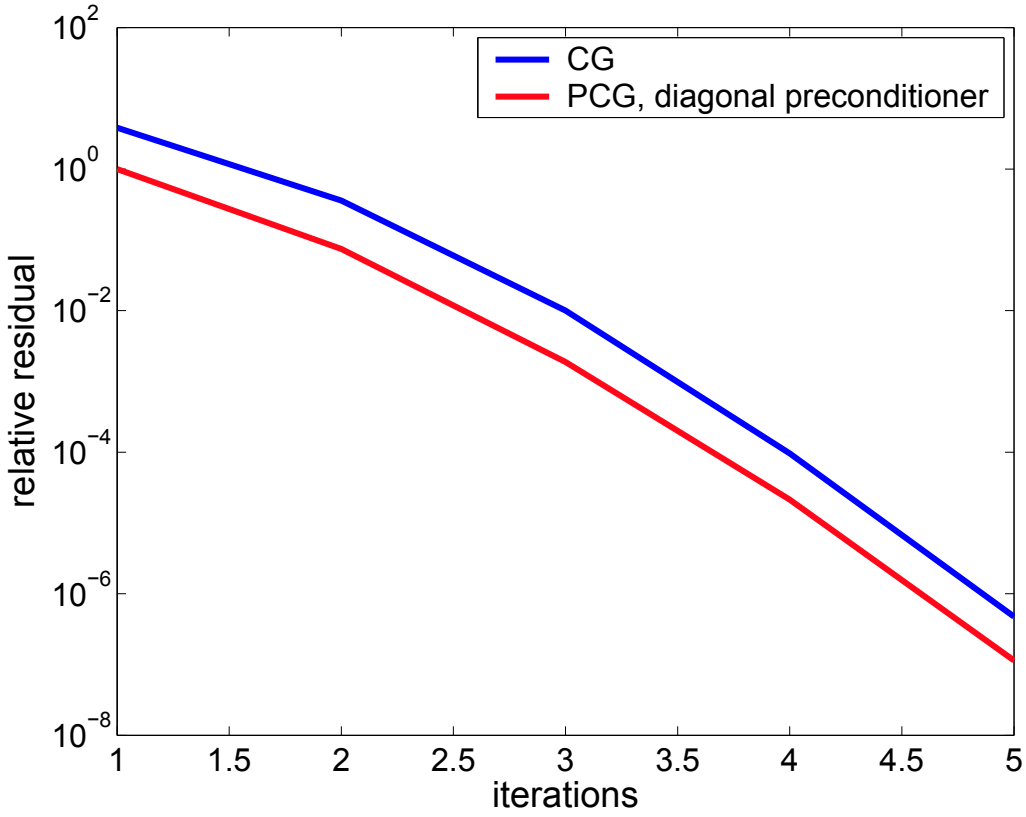


# of iteration

conjugate gradient requires less since the # of A orthogonal direction is equal to the order of the matrix

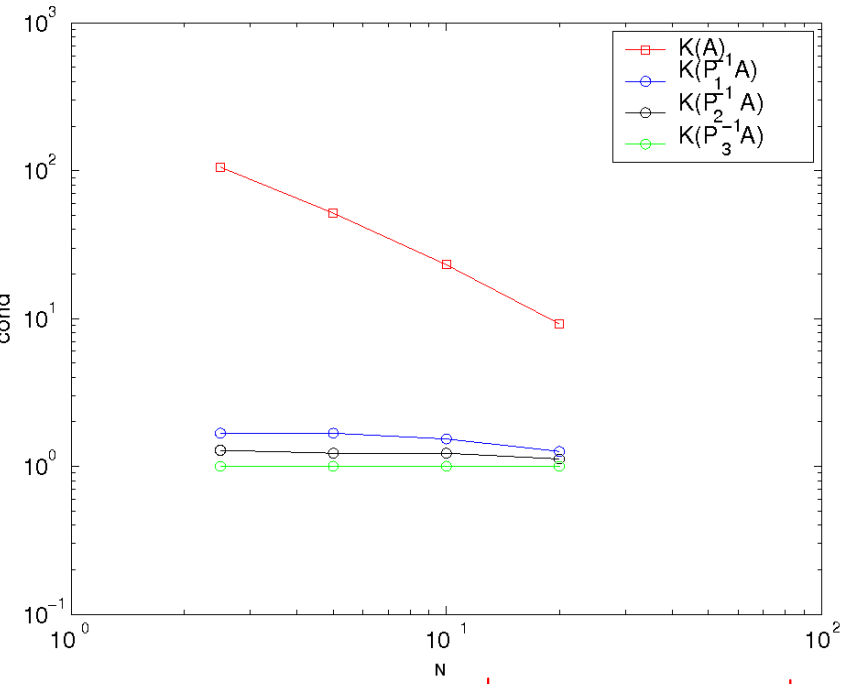
Preconditioning

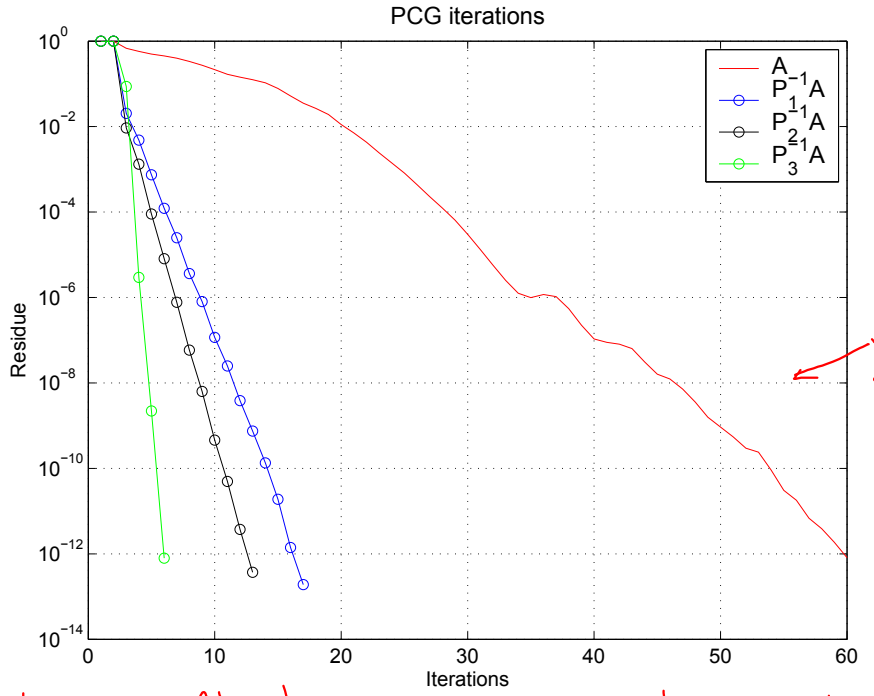
The convergence conjugate gradient and the preconditioned conjugate gradient methods with a diagonal preconditioner ($K = 4 \cdot 10^8$)



Preconditioning

Role of the preconditioning over the condition number of a matrix (approximation by the finite element method of the Laplacian operator)



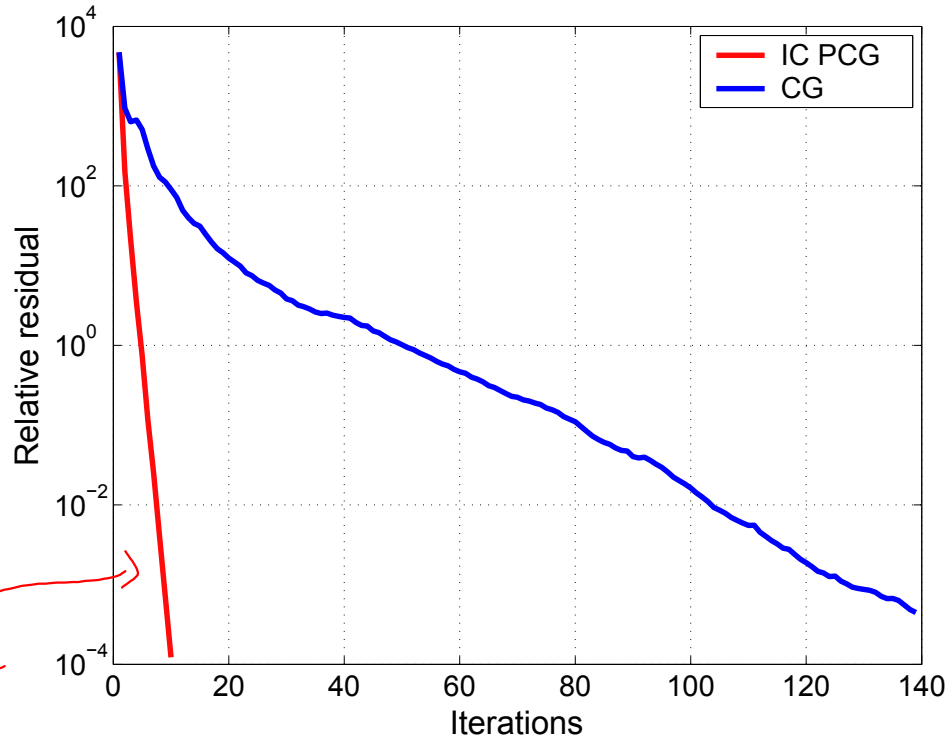


preconditioning SPEEDS UP convergence

Question: do you know which matrix could be preconditioned and which isn't? If K ~ 1 could be preconditioned. If K is high, is likely not preconditioned

Preconditioning

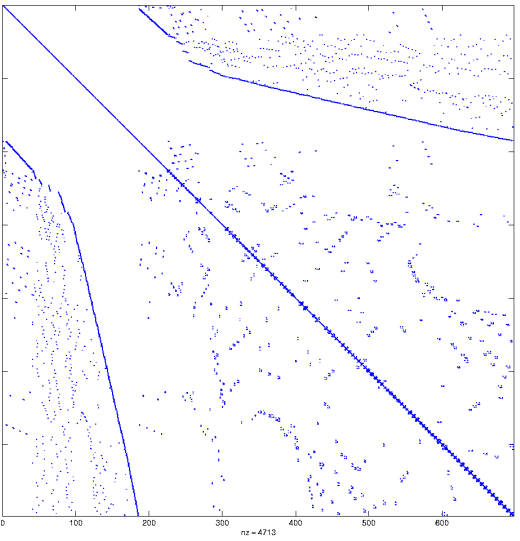
The convergence of the conjugate gradient method and the preconditioned conjugate gradient method with a preconditioner based on an Cholesky incomplete decomposition for a sparse matrix originated from the finite element method ($K = 1.5 \cdot 10^3$)



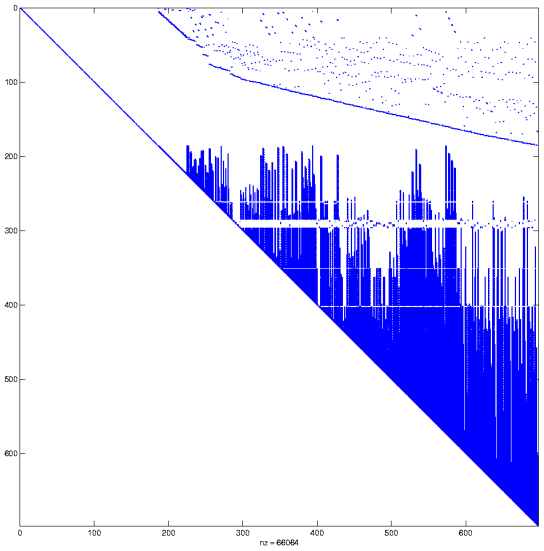
Cholesky as preconditioner

Pre-conditioning

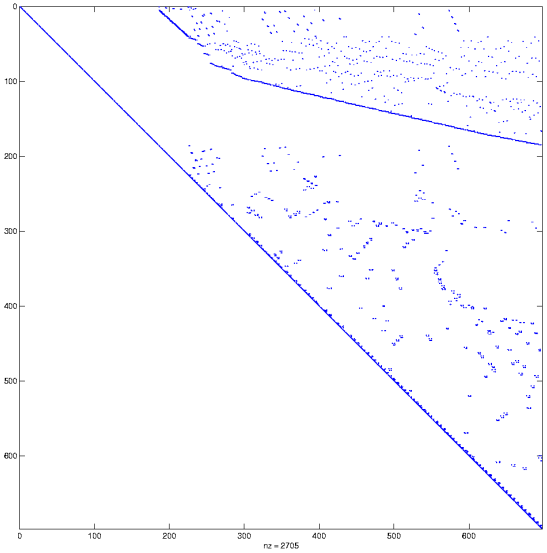
Comparison between the non zero elements of the sparse matrix A from the previous example, its Cholesky factor R and the matrix $\tilde{R}$ obtained by the Cholesky incomplete decomposition:



A



R



$\tilde{R}$

Non-linear systems

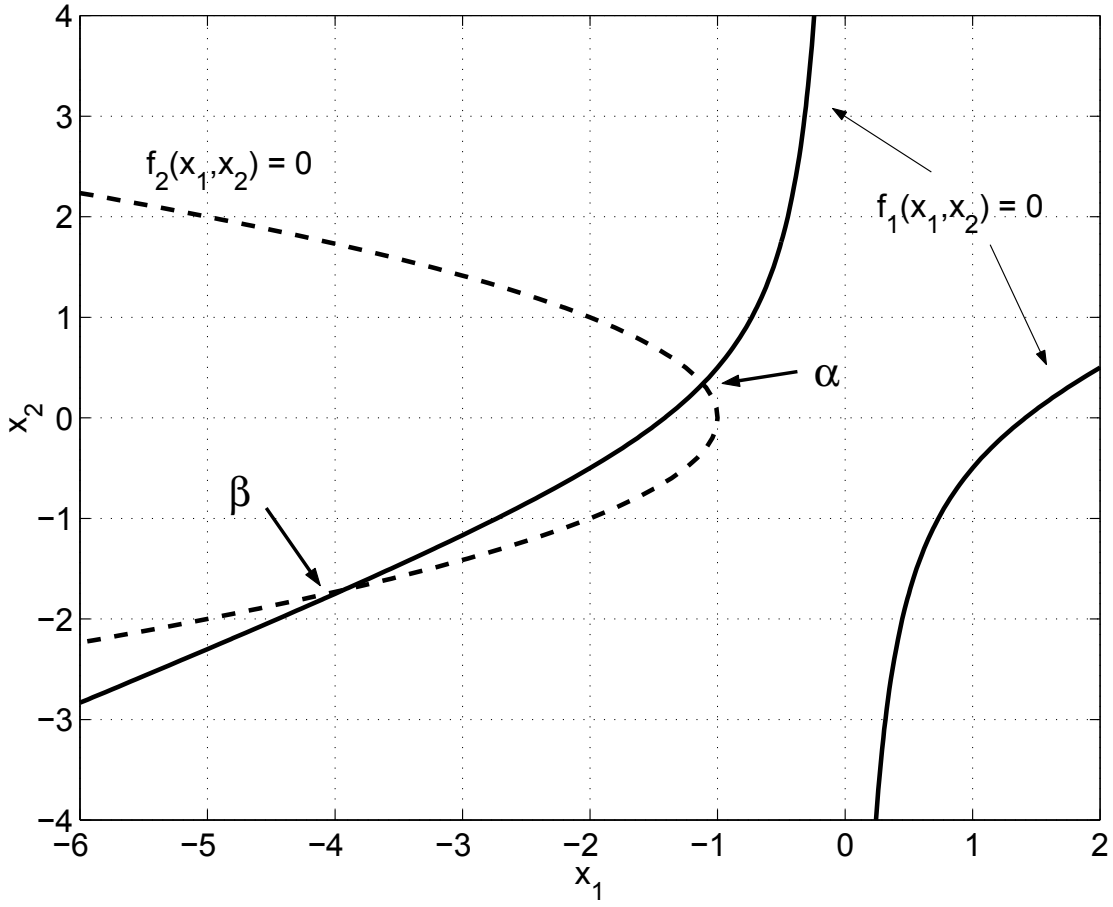
Example 6. Lets consider the following system of non-linear equations:

$$\begin{cases} x_1^2 - 2x_1x_2 = 2 \\ x_1 + x_2^2 = -1. \end{cases} \tag{20}$$

This system can be written in the form:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \qquad \text{i.e.} \qquad \begin{cases} f_1(x_1, x_2) = 0 \\ f_2(x_1, x_2) = 0 \end{cases}$$

where $\mathbf{f} = (f_1, f_2)$, $f_1(x_1, x_2) = x_1^2 - 2x_1x_2 - 2$ and $f_2(x_1, x_2) = x_1 + x_2^2 + 1$.



Curves $f_1 = 0$ *and* $f_2 = 0$ *in the square* $-6 \leq x_1 \leq 2$, $-4 \leq x_2 \leq 4$

We want to generalise the Newton method for the case of non-linear systems. To do this, we define the *Jacobian* matrix of the vector $\mathbf{f}$:

$$J_{\mathbf{f}}(\mathbf{x} = (x_1, x_2)) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 - 2x_2 & -2x_1 \\ 1 & 2x_2 \end{bmatrix}.$$

If $J_{\mathbf{f}}(\mathbf{x}^{(k)})$ is invertible, the Newton method for non linear systems is written : *Let* $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)})$, *we compute for* $k = 0, 1, 2, \ldots$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [J_{\mathbf{f}}(\mathbf{x}^{(k)})]^{-1}\mathbf{f}(\mathbf{x}^{(k)}), \quad k = 0, 1, 2 \ldots . \tag{21}$$

We can write (21) as

{ in place of 1/f' of Newton method, we put the Jacobian at -1 (inverse of Jacobian)

$$[J_{\mathbf{f}}(\mathbf{x}^{(k)})](\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = -\mathbf{f}(\mathbf{x}^{(k)}), \quad k = 0, 1, 2 \ldots . \tag{22}$$

↑ quasi-Newton methods update Jacobian every m iteration with m > 1

Description of the first step of the algorithm: given the vector $\mathbf{x}^{(0)} = [1, 1]^T$.
We calculate:

$$J_{\mathbf{f}}(\mathbf{x}^{(0)}) = \begin{bmatrix} 0 & -2 \\ 1 & 2 \end{bmatrix}.$$

We determine $\mathbf{x}^{(1)}$ as solution of the equation :

$$[J_{\mathbf{f}}(\mathbf{x}^{(0)})](\mathbf{x}^{(1)} - \mathbf{x}^{(0)}) = -\mathbf{f}(\mathbf{x}^{(0)}).$$

And we continue with (22).

Lecture 21 — Solution of prototypical PDE in dimension 1. — 10/12/2020

Prototypical problem

Given f (regular enough) find u s.t.

$$\begin{cases} -u'' = f & \text{in } [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

(unknown of the problem is a ft; $u(0) = u(1) = 0$: boundary CNs)

Problems like this can be found:
- Heat conduction
- Elasticity
- Maxwell equations

Requirement for WELL POSEDNESS: $f \in C^0([0,1])$, and $u \in C^2([0,1])$

Strong formulation

First approach

FINITE DIFFERENCES.

This prob will be well posed if the final linear system is invertible (det of matrix is $\neq 0$)

Approximate the differential operators using the incremental definition of the derivative:

$$u'(x) := \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}$$

Forward difference — FD

$$u'(x) := \lim_{h \to 0} \frac{u(x) - u(x-h)}{h}$$

backward — BD

(can do also for u'')

$$u'(x) := \lim_{h \to 0} \frac{u(x+h) - u(x-h)}{2h}$$

centred — CD

for C^1 fts these are $(\Rightarrow$ for $h \to 0$

Then the FINITE DIFFERENCE scheme is:

Fix h, and define Du as

$$D^F u = \frac{u(x+h) - u(x)}{h} \qquad \text{FD}$$

$$D^B u = \frac{u(x) - u(x-h)}{h} \qquad \text{BD}$$

$$D^C u = \frac{u(x+h) - u(x-h)}{2h} \qquad \text{CD}$$

Assume $u \in C^\infty(I(x))$ in an open set of x $I(x)$

then a taylor expansion

$$u(x+h) = u(x) + u'(x)h + \frac{u''}{2}(x)h^2 + \frac{u'''}{6}(x)h^3 + \frac{u''''}{24}(x)h^4 + \dots + \frac{u^{(n)}h^n}{n!}$$

$$u(x-h) = u(x) - u'(x)h + \frac{u''}{2}(x)h^2 - \frac{u'''}{6}(x)h^3 + \frac{u''''}{24}(x)h^4 + \dots + \frac{u^{(n)}(-h)^n}{n!}$$

Theorem Given $u \in C^\infty([a,b])$, $x \in (a,b)$, then $\forall k > 0$

$\exists \xi \in (x, x+h)$

equality EXACT

$$u(x+h) = \left(\sum_{i=0}^{k-1} \frac{u^{(i)}(x)}{i!} h^i \right) + \frac{u^{(k)}(\xi)}{k!} h^k$$

You can truncate expansion at any para and what you find is the value of the ft at one point in $(x, x+h)$

Truncation Error of Finite Difference schemes:

! At point 1 for truncating, I have to use the ξ point, by previous theorem

because I divide by h

FD $$\frac{u(x+h) - u(x)}{h} = u'(x) + \frac{u''(\xi)}{2}h$$ 1st order scheme

BD $$\frac{u(x) - u(x-h)}{h} = u'(x) - \frac{u''(\xi)}{2}h$$ 1st order scheme

CD $$\frac{u(x+h) - u(x-h)}{2h} = u'(x) + \frac{u'''(\xi)}{6}h^2$$ 2nd order scheme

Use low order (I ord) if ft is not regular (maybe u'' doesn't exist) and low h, while high order if u is regular, and high h

CFD^{II} Central finite diff of II ord

$$\frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = u''(x) + \frac{u''''(\xi)}{12}h^2$$

I want II derivative

2nd order scheme for 2nd derivative

Apply CFD II to $-u'' = f$, $u(0) = u(1) = 0$ in $[0,1]$

Split the interval $[0,1]$ into $N-1$ subintervals (N points)

$$\{x_i\}_{i=0}^{N-1} = \left\{0, \frac{1}{N-1}, \frac{2}{N-1}, \dots, \frac{N-1}{N-1}=1\right\} = \left\{\frac{i}{(N-1)}\right\}_{i=0}^{N-1} = \{i \cdot h\}_{i=0}^{N-1}$$

$$h = \frac{1}{(N-1)} \qquad \text{set } u^i := u(x_i)$$

$$\{u_i\}_{i=0}^{N-1} \in \mathbb{R}^N$$

For $i = 1$ to $N-2$ write: (For all i)

$$\frac{1}{h^2}\underbrace{\left[-u^{i-1} + 2u^i - u^{i+1}\right]}_{\simeq -u''} = f_i \equiv \underline{\underline{A}}\,\underline{u} = \underline{f}$$

system of equations

$$A \in \mathbb{R}^{N\times N}, \quad u, f \in \mathbb{R}^N$$

$$\sum_{J=0}^{N-1} A_{iJ}\, u^J = f_i$$

$$A_{iJ} = \frac{1}{h^2}\begin{cases} -1 & \text{when } J = i-1 \text{ or } i+1 \\ 2 & \text{when } J = i \\ 0 & \text{otherwise} \end{cases} \qquad \text{and } i \in [1, N-2]$$

$$\begin{matrix} 0 \\ 1 \\ \\ \\ \\ N-2 \\ N-1 \end{matrix} \begin{pmatrix} \cdots & & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & \cdots \end{pmatrix} \cdot \frac{1}{h^2}$$

$$f_i = f(x_i) \qquad i = 1, \dots, N-2$$

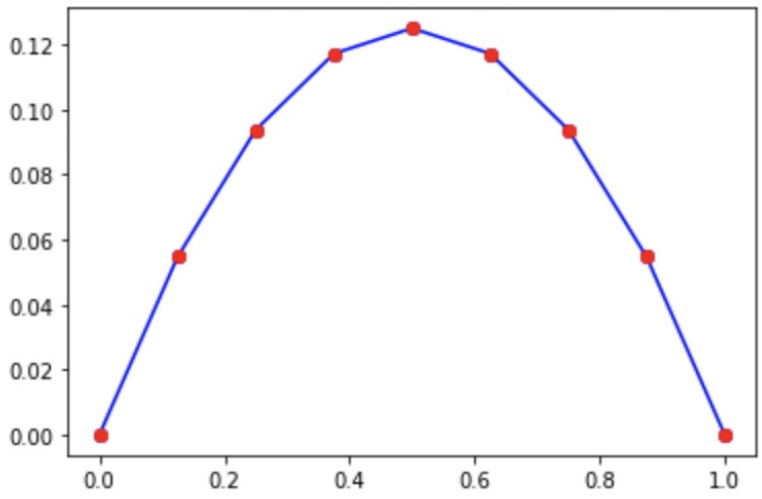
What about I and last columns?

For $i = 0$, $i = N-1$ we impose ← we set the BOUNDARY CONDITION

$$u^0 = 0 \quad \rightarrow \quad A_{00} = 1 \qquad f_0 = 0$$

$$u^{N-1} = 0 \quad \rightarrow \quad A_{N-1,N-1} = 1 \qquad f_{N-1} = 0$$

! The more the points the higher the CN NUMBER => more ill-conditioned



Example of solution to

$$-u'' = 1$$

$$u(0) = u(1) = 0$$

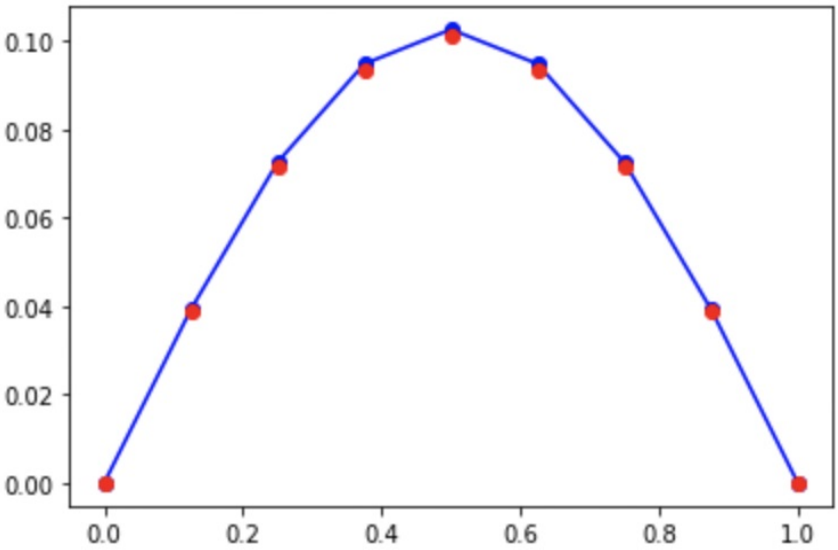
with $N = 9$

the red dots are the exact solution.

$$u_{exact} = \frac{x(1-x)}{2}$$ ← exact function

We have error zero (blue overlap red) because error is up to u'''', but u'''' of a parabola is zero!

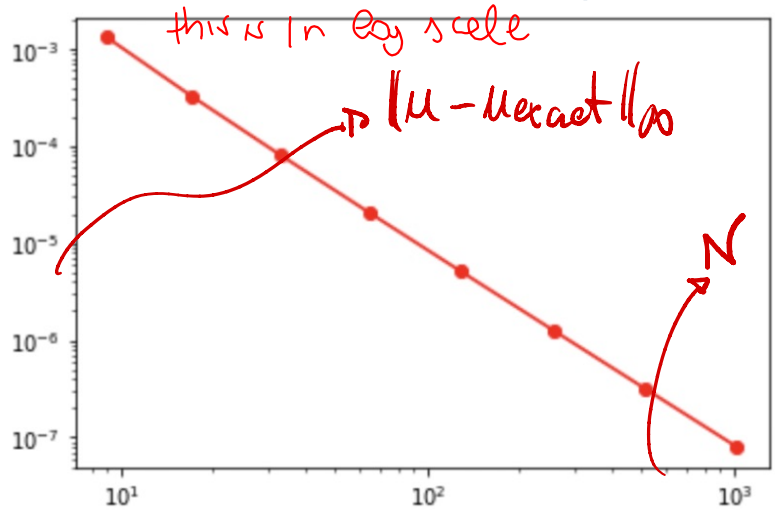
Same thing with $f(x) = \sin(\pi x)$



Here the error is not zero, because

$$u'''' = -f'' \neq 0$$

Plot the error $\|u - u_{exact}\|_\infty = \max_{i \in [0, N-1]} |u^i - u(x_i)|$



Straight line:

$$E = c h^k$$

$$= c \left(\frac{1}{N-1}\right)^k$$

what is k?

inclination of line (with minus sign) in LOG SCALE (→ found $\log E$)

How to measure the rate k ?

Two options: 1) we know the exact solution

2) we do <u>not</u> know the exact solution

Case 1

Hypothesis: C is indep. of h_i and h → target

$$E_1 = \|u_{h_1} - u_{exact}\|_\infty \simeq C h_1^k$$

$$E_2 = \|u_{h_2} - u_{exact}\|_\infty \simeq C h_2^k$$

same k for the 2 cases

$$\frac{E_1}{E_2} \simeq \left(\frac{h_1}{h_2}\right)^k$$

we need at least 2 measurements

$$k = \frac{\log(E_1/E_2)}{\log(h_1/h_2)} = \frac{\log(E_1) - \log(E_2)}{\log(h_1) - \log(h_2)}$$

Case 2

Make sure we reduce h by a constant factor θ, and use at least 3 approx solutions.

$h_{i+1} = \theta h_i$

$$\|u_{h_2} - u_{h_1}\|_\infty \leq \|u_{h_2} - u_{exact}\|_\infty + \|u_{h_1} - u_{exact}\|_\infty$$

$$\simeq C h_1^k + C h_2^k$$

$$\simeq C h_1^k + C \theta^k h_1^k$$

$$\simeq \underbrace{C(1+\theta^k)}_{C_2} h_1^k$$

$$\frac{\| u_{h_1} - u_{h_2} \|_\infty}{\| u_{h_2} - u_{h_3} \|_\infty} \sim \frac{C_2 h_1^k}{C_2 h_2^k} = \theta^{-k}$$

$h_2 = \theta h_1$

$h_3 = \theta h_2$

I do a third measurement and I do the same, by the fact that 2 consecutive stv go the same in terms of error

$$k = \frac{\log\left(\| u_{i+2} - u_{i+1} \|_\infty\right) - \log\left(\| u_{i+1} - u_i \|_\infty\right)}{\log(\theta)}$$

I remember that we had $u_i, u_{i\pm1}$, that are 3 consecutive approximate solutions

Lecture 22 LH

PDE - 1D - FEM ← Finite Elements Method

15.12.2020

$$\text{(1)} \begin{cases} -u'' = f & \text{in } [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

We tried to:

⇝ approximate u''

- split $[0,1]$ in $N-1$ intervals with size $h = \frac{1}{(N-1)}$
- define approx of u'' on each $x_i = ih \qquad i = 0, \dots, N-1$

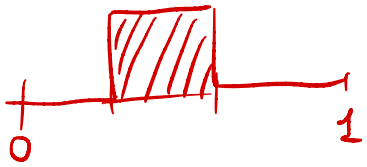
$$-u^{i-1} + 2u^i - u^{i+1} = f_i$$

- issues: i) u'' is defined if $u \in C^2([0,1])$

  ii) ($f \in C^0([0,1])$)

excluded situations:

f:



iii) truncation error: ← defines the order by which I approximate u''

$$u'' = CD^{II}(u) + O(\|u''''\|_\infty h^2)$$

SOLUTION: rewrite the problem (1) in weak form

Let's see a special case

- Let v be a smooth function: $v : [0,1] \to \mathbb{R}$

Multiply (1) by v and integrate on $[0,1]$

$$\int_0^1 -u'' \cdot v = \int_0^1 f \cdot v \qquad \forall\, v \text{ "smooth enough"}$$

Let's further assume that $v(0) = v(1) = 0$ ← v satisfies boundary CNs of problem

→ Integrate by parts: $\int_0^1 u'v' - u'v\Big|_0^1$ → 0 because $v(0) = v(1) = 0$

$$(wv)' = w'v + wv' \qquad w = u'$$

so the problem is now:

Find u in V s.t.

$$\text{(2)} \quad \int_0^1 u'v' = \int_0^1 fv \qquad \forall\, v \in V$$

Weak formulation of (1) $(-u'' = f)$

← This is weaker: we are only asking u to have only one degree of continuity (FIRST DERIVATIVE not even needed)

What's the "right" space V for (2) to make sense?

i) V must incorporate that $v(0) = v(1) = 0 \quad \forall v \in V$

ii) (2) must make sense $\forall u, v \in V$

in particular

$\rightarrow v = u \Rightarrow \int_0^1 (u')^2 = \int_0^1 fu$

$\Rightarrow$ It only makes sense if $\int_0^1 (u')^2 < +\infty$

So it must be

$\Rightarrow V = \{ v \text{ s.t. } \int_0^1 (v')^2 < +\infty \;/\; v(0) = v(1) = 0 \}$

(V is the Sobolev space $H_0^1([0,1])$ ← means: "first derivatives")

space of ft that are in L^2 and whose first derivatives are in L^2

V is the space of functions whose first derivative is in L^2, and that are zero on $\{0, 1\}$

A solution to (1) is a solution to (2) (always true!)

↓ sum

$(wv)' = w'v + wv'$

$\Rightarrow \int_0^1 (wv)' = \int_0^1 w'v + \int_0^1 wv'$

$wv\Big|_0^1 = \int_0^1 w'v + \int_0^1 wv'$

$\Rightarrow \int_0^1 w'v = -\int_0^1 wv' + wv\Big|_0^1$

stronger in weak sense, but not the contrary in general

$$\boxed{\int_0^1 -u''v = \int_0^1 u'v' + u'v\Big|_0^1}$$

? 0 because $v(0) = v(1) = 0$

"weak" solution

$\not\Leftarrow$ false: $-u'' = f \Rightarrow \int_0^1 u'v' = \int_0^1 fv \quad \forall v \in V$

You def the WEAK vf of pb as the solution of this problem

$V \equiv H_0^1([0,1]) := \{ v \in L^2, \ v' \in L^2, \ v(0) = v(1) = 0 \}$

f must be s.t. $\int_0^1 fv < +\infty \quad \forall v \in H_0^1([0,1])$

true if $f \in L^2$

but also if f is less reg. than L^2, but such that $\int_0^1 f \cdot v < +\infty \quad \forall v \in H_0^1([0,1])$

$f \in V^*$ ← dual space of V ← what is that!

↳ for example Dirac δ is not in L^2, but this works anyway

Weak derivative:

$Du(x) = u'(x) \quad \forall u \in C^1([0,1]), \forall x \in [0,1]$

$Du(x)$ is ← weak derivative s.t. $-\int_0^1 u \varphi' = \int_0^1 Du \, \varphi \quad \forall \varphi \in C_0^\infty((0,1))$

By construction, if $u \in C^1([0,1]) \Rightarrow$

$-\int_0^1 u \varphi' \overset{\text{integration by part}}{=} \int u' \varphi = \int_0^1 Du \, \varphi \Rightarrow Du = u'$

↑ True only if $u \in C^1$, but then can be def whenever $\int u \varphi'$ exist

$D^\alpha u = u^{(\alpha)}$ if $u \in C^\alpha([0,1])$

D^α is a weak derivative of u if ← this is satisfied

$$\int_0^1 D^\alpha u, \varphi = (-1)^\alpha \int_0^1 \varphi^{(\alpha)} Du \quad \forall \varphi \in C_0^\infty((0,1))$$

where $C_0^\infty((0,1)) := \{ \varphi \in C^\infty([0,1]) \text{ s.t. } \varphi^{(\alpha)}(0) = \varphi^{(\alpha)}(1) = 0 \ \forall \alpha \in \mathbb{N}_0 \}$

The final weak form of our problem is

Given $f \in L^2([0,1])$ (could be $(H_0^1([0,1]))^*$ dual of $H_0^1([0,1])$)

Find $u \in H_0^1([0,1])$ s.t.

$$\int_0^1 u' v' = \int_0^1 f v \qquad \forall\, v \in H_0^1([0,1])$$

$$H_0^1([0,1]) := \{ v \in L^2([0,1]),\ v' \in L^2([0,1]),\ v(0) = v(1) = 0 \}$$

Assume that we have $V_h \subset V \equiv H_0^1(\Omega)$

$V_h = \text{span}\{ v_i \}_{i=0}^{N-1}$ $\quad \dim(V_h) = N$ ← finite dim

Write a approximate problem:

find $u_h \in V_h$ s.t. $\quad \left(u_h(x) = \sum_{J=0}^{N-1} u^J v_J(x) \right)$

$$\int_0^1 u_h' v_i' = \int_0^1 v_i f \qquad i = 0, \dots, N-1$$

← that is for ALL basis functions

$$\rightarrow \sum_{J=0}^{N-1} \int_0^1 u^J v_J' v_i' = \int_0^1 v_i f$$

$$A u = F$$

$$A_{iJ} := \int_0^1 v_J' v_i' \qquad F_i = \int_0^1 v_i f$$

I DO NOT CHANGE WAY IN WHICH I COMPUTE DERIVATIVES: I will choose basis so I can compute derivative exactly

Define V_h piecewise polynomial and $C^0([0,1])$

Select N points that define $N-1$ intervals

$$\{x_i\}_{i=0}^{N-1} = ih = \frac{i}{(N-1)}$$

the intervals are $[x_i, x_{i+1}]$ $i = 0, \dots, N-2$

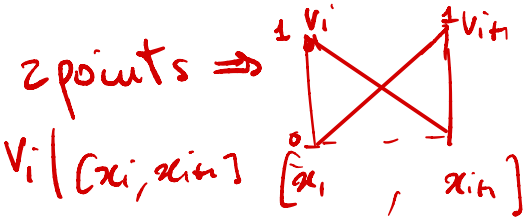
poly of order k

$$V_h^k := \{ v \in \mathbb{P}^k([x_i, x_{i+1}]) \mid v(0) = v(1) = 0 \quad v \in C^0([0,1]) \}$$

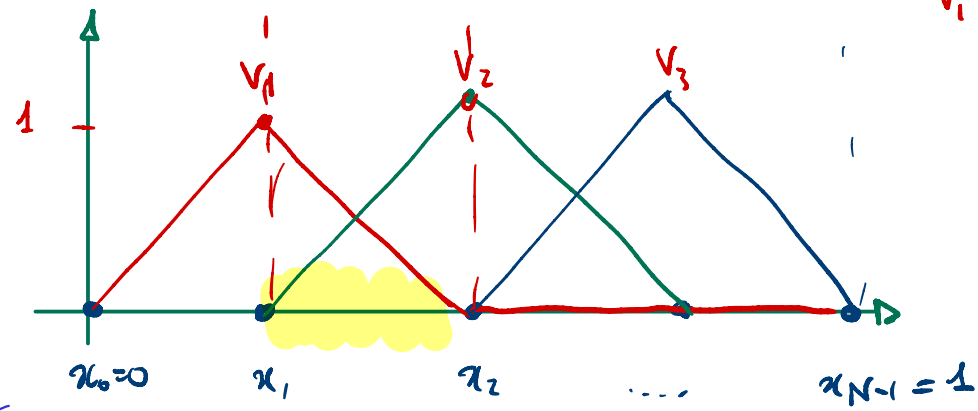
Continuous piecewise polynomial of order k.

Example $k = 1 \Rightarrow \mathbb{P}^1([x_i, x_{i+1}])$

we choose Lagrange basis of order 1 in $[x_i, x_{i+1}]$ with support points $\{x_i, x_{i+1}\}$ 2 points $\Rightarrow$



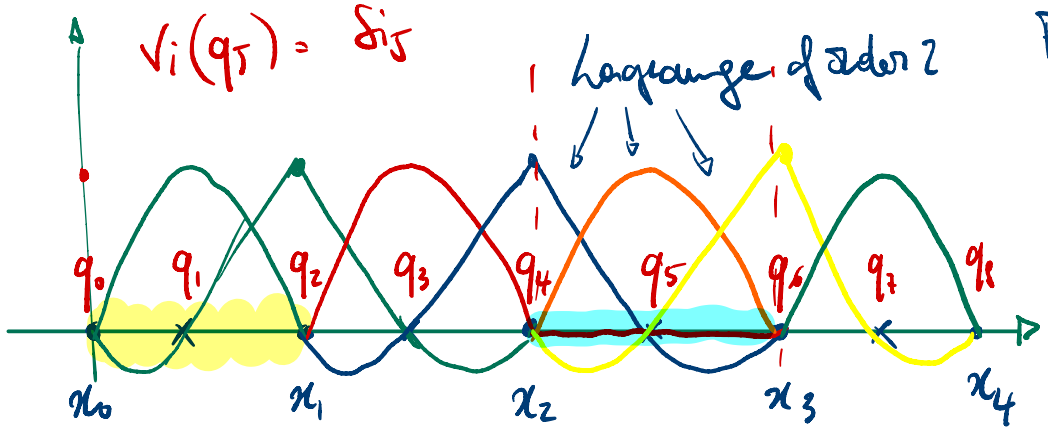
$v_i | [x_i, x_{i+1}]$ $[x_i, x_{i+1}]$



$q_i = x_i$ for linear polynomials.

For example, choose

$$v_i(x_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \quad v_i \in V_h^1$$



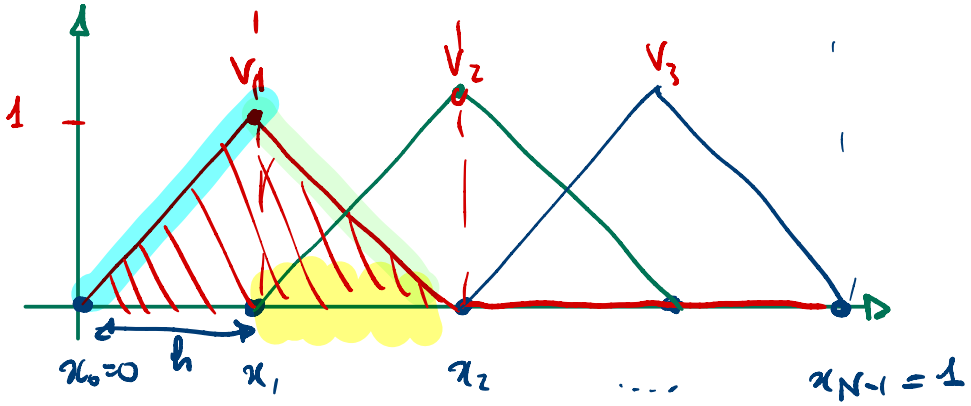
For polynomials of order 2, choose

$$\{q\}_{i=0}^{(N-1)+(N-2)} = \{x_i\}_{i=0}^{N-1} \cup \left\{\frac{x_i + x_{i+1}}{2}\right\}_{i=0}^{N-2}$$

union

Consider

1D case linear

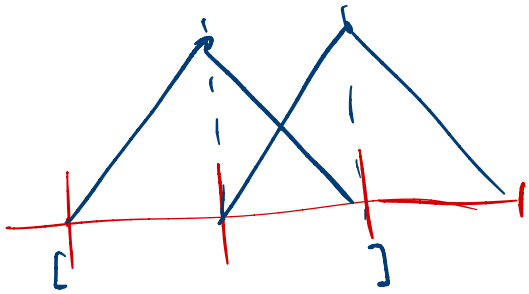


$$\dim(V_h) = N-2$$

$$V_i = \begin{cases} \dfrac{(x - x_{i-1})}{h} & x \in [x_{i-1}, x_i] \\ \dfrac{(x_{i+1} - x)}{h} & x \in [x_i, x_{i+1}] \\ 0 & \text{elsewhere} \end{cases}$$

$i = 1, \dots, N-2$

$$V_i' = \begin{cases} \dfrac{1}{h} & \text{in } [x_{i-1}, x_i] \\ -\dfrac{1}{h} & \text{in } [x_i, x_{i+1}] \\ 0 & \text{elsewhere} \end{cases}$$



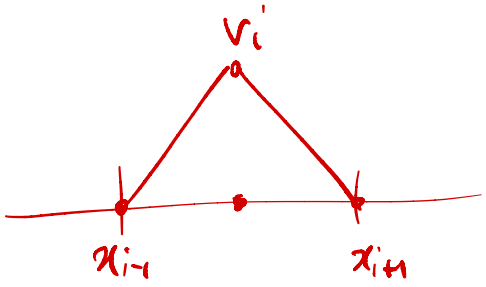
$$A_{ij} := \int_0^1 V_i' V_j' \begin{cases} \neq 0 & \text{if } i = j, j-1, j+1 \\ 0 & \text{elsewhere} \end{cases}$$

$$A_{ij} = \begin{cases} \displaystyle\int_{x_{i-1}}^{x_{i+1}} \left(\frac{1}{h}\right)^2 = 2\frac{1}{h} & \text{if } i = j \\ \displaystyle\int_{x_i}^{x_{i+1}} -\left(\frac{1}{h}\right)^2 = -\frac{1}{h} & \text{if } i = j-1 \text{ or } i = j+1 \end{cases}$$

$$A_{ij} := \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \end{pmatrix} \frac{1}{h}$$

Same matrix as FD ($\frac{1}{h}$!)

$$F_i = \int_{x_{i-1}}^{x_{i+1}} f(x)\, v_i(x)\, dx$$



$$\approx f(x_i) \cdot h$$ using trapez rule (same as FD with h)

since integral of v_i between x_{i-1}, x_{i+1} is 1/2 of the whole rectangle of height $1/h$

proof:

$$\int_{x_{i-1}}^{x_i} v_i(x) f(x) + \int_{x_i}^{x_{i+1}} v_i(x) f(x)$$

$$h\left[\frac{1}{2} v_i(x_{i-1}) f(x_{i-1}) + \frac{1}{2} v_i(x_i) f(x_i)\right] + h\left[\frac{1}{2} v_i(x_i) f(x_i) + \frac{1}{2} v_i(x_{i+1}) f(x_{i+1})\right]$$

($v_i(x_{i-1}) = 0$, $v_i(x_{i+1}) = 0$)

$$= f(x_i)\, h$$

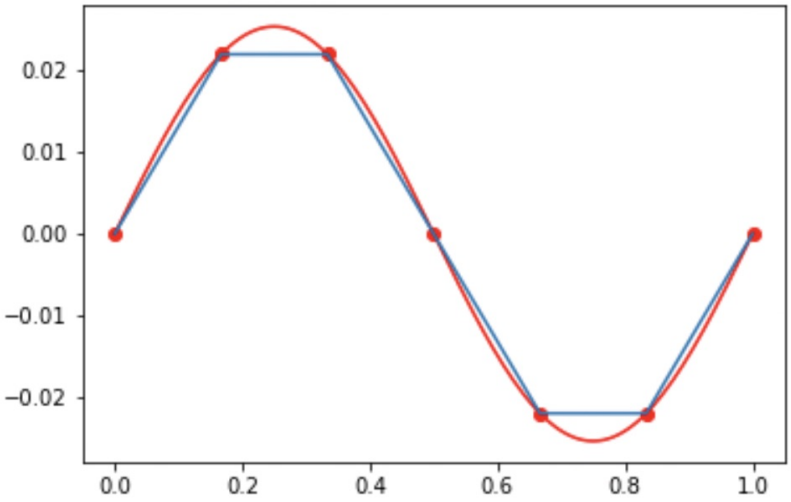
Approximation property of FEM:

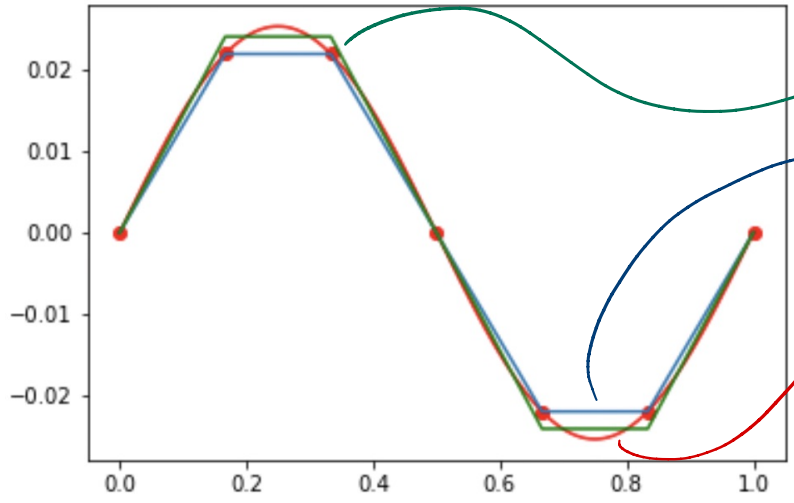
$$\int_0^1 u' v_h' = \int_0^1 f v_h \qquad \forall v_h \in V_h$$

$$\int_0^1 u_h' v_h' = \int_0^1 f v_h \qquad \forall v_h \in V_h \subset V$$

$$\int_0^1 (u' - u_h') v_h' = 0 \qquad \forall v_h \in V_h \subset V$$

orthogonality of the error





FD

FE

Exact

In 1D, the FEM solution coincides with the exact solution at the nodes (while FD does not)

but it depends on the partial diff equation and IT IS EXACT FOR ANY SYMMETRIC POSITIVE DEFINITE SYSTEM (when you can construct a norm with differential operators)

FD is special case of FEM, but it is the "prototipo figurato" of FEM

Lecture 23 LH — Abstract setting of FEM — 17.12.2020

Small recap of necessary concepts of functional analysis

Given V, W normed vector spaces (on the real field $\mathbb{R}$)

Define $\mathcal{L}(V,W)$ the space of linear functions from V to W:

$$f \in \mathcal{L}(V,W) \qquad f : V \longrightarrow W$$

i) $\forall u, v \in V$, $\forall \alpha, \beta \in \mathbb{R}$ $\quad W \ni f(\alpha u + \beta v) = \alpha f(u) + \beta f(v)$

ii) $\forall u \in V \quad \|f(u)\|_W \leq \|f\|_* \, \|u\|_V$

$$\|f\|_* := \sup_{0 \neq v \in V} \frac{\|f(v)\|_W}{\|v\|_V}$$

Special cases:

$$\mathcal{L}(V,\mathbb{R}) \equiv V^* \quad \text{the dual space of } V$$

$$f \in V^* \rightarrow f : V \longrightarrow \mathbb{R}, \quad v \longrightarrow f(v) \equiv \underbrace{{}_{V^*}\langle f, v \rangle_V}_{\text{just notation for now}} \in \mathbb{R}$$

you define

$$\|f\|_{V^*} = \|f\|_* = \sup_{0 \neq v \in V} \frac{|f(v)|}{\|v\|_V} = \sup_{0 \neq v \in V} \frac{|{}_{V^*}\langle f, v \rangle_V|}{\|v\|_V}$$

Bilinear forms: $a : V \times V \longrightarrow \mathbb{R}$

$$a(u,v) \equiv \langle Au, v \rangle \qquad \forall\, u, v \in V$$

where

$$A \in \mathcal{L}(V, V^*) \qquad A : V \longrightarrow V^*$$

$$(Au)(v) \in \mathbb{R}$$

(Au: this is an object in V^*)

$a(\cdot,\cdot)$ is a bilinear form if it is a function from $V \times V$ to $\mathbb{R}$ which is linear in both arguments separately

$$\forall\, u, v, w, z \in V \qquad \alpha, \beta, \gamma, \theta \in \mathbb{R}$$

Linearity in both arguments:

$$a(\alpha u + \beta v, \gamma w + \theta z) = \gamma\alpha\, a(u,w) + \alpha\theta\, a(u,z) \ldots.$$

$$|a(u,v)| \leq \|A\|_* \, \|u\|_V \, \|v\|_V \qquad \forall\, u, v \in V$$

where

$$\|A\|_* := \sup_{0 \neq u \in V} \frac{\|Au\|_{V^*}}{\|u\|_V} = \sup_{0 \neq u,v \in V} \frac{|\langle Au, v \rangle|}{\|u\|_V \, \|v\|_V} = \sup_{0 \neq u,v \in V} \frac{|a(u,v)|}{\|u\| \, \|v\|}$$

if $\|A\|_*$ is $< +\infty$ $\longrightarrow$ then $a(u,v)$ is bounded

Finite element method:

- Take the strong form of the PDE (1)
- Look for a solution in a vector space V
- multiply (1) by test functions in V and integrate on the domain Ω of the PDE
- integrate by parts until you cannot reduce any more the order of the differential op.

$\Rightarrow$ you end up with:

i) a bilinear form in $V \times V$

ii) a linear operator in V^*

Example

new notation for II derivatives

① $\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$

② Take u in V (V contains the condition $u=0|_{\partial\Omega}$)

③ $\int_\Omega -\Delta u \cdot v = \int_\Omega f v$

$-\Delta u \to -\operatorname{div}(\nabla u)$

④ Integration by part

$\int_\Omega \operatorname{div}(w) = \int_{\partial\Omega} w \cdot n$

$v = 0|_{\partial\Omega}$

0

$\int_\Omega \nabla u \cdot \nabla v - \int_{\partial\Omega} \nabla u \cdot n \, v = \int_\Omega f v$

⑤ $a(u,v) = \int_\Omega \nabla u \nabla v$

$f(v) := \int_\Omega f v$

$f \in V^*$

Consider the case where V is Hilbert,

i) $(u,v)_V$ is its scalar product

ii) $\|u\|_V^2 := (u,u)_V$

Example for FEM of Poisson Problem

$V \equiv H_0^1(\Omega) := \{ v \in L^2(\Omega),\ \nabla v \in L^2(\Omega),\ v|_{\partial\Omega} = 0 \}$

$$(u,v)_V \overset{\text{definition}}{=} \int_\Omega uv + \int_\Omega \nabla u \cdot \nabla v$$

$$\Rightarrow \|u\|_V = \|u\|_1 := \left(\int_\Omega u^2 + \int |\nabla u|^2 \right)^{\frac{1}{2}}$$

A scalar product is a BILINEAR FORM

Abstract setting. V Hilbert (not only H_0^1, but any Hilbert)

Lax Milgram Lemma:

Let

1) a: bilinear, bounded operator

2) a: coercive

1) $a(u,v) \leq \|A\|_* \|u\|_V \|v\|_V \quad \forall u,v \in V$

2) $a(u,u) \geqslant \alpha \|u\|_V^2 \quad \forall u \in V$

Then $\forall f \in V^*$, $\exists!$ solution $u \in V$ to

① $a(u,v) = f(v) = \langle f, v \rangle \quad \forall v \in V$

this it has the property:

And $\|u\| \leq \frac{1}{\alpha} \|f\|_*$ ← solution is BOUNDED with respect to "data"

Since ① is true for every $v \in V$, and $u \in V$ then:

$$\alpha \|u\|^2 \leq a(u,u) = \langle f, u \rangle = f(u) \leq \|f\|_* \|u\|$$

$$\rightarrow \quad \|u\| \leq \frac{1}{\alpha} \|f\|_*$$

Proof of Lax Milgram

2 ingredients
1) Riesz Theorem
2) Contraction theorem (fixed points)

Riesz theorem (1): (Riesz operator $\tau : V^* \rightarrow V$)

$$\forall \; f \in V^* \;, \quad \exists! \; \tilde{f} \in V \quad \text{s.t.}$$

i) $(\tilde{f}, v)_V = \langle f, v \rangle = f(v)$

ii) $\|\tilde{f}\|_V = \|f\|_*$ $\qquad \tilde{f} = \tau f$

Contraction Theorem

Let $T : V \longrightarrow V$ be a contraction, i.e:

$\exists L < 1$ s.t.

$$\|T(u) - T(v)\|_V \leq L \|u - v\|$$

Then $\exists!$ φ fixed point of T: $T(\varphi) = \varphi$

↓ proof of contraction theorem

Let $u^0 \in V$, and define $u^{k+1} = T(u^k)$

$$\rightarrow \|u^{k+1} - u^k\|_V = \|T(u^k) - T(u^{k-1})\| \leq L \|u^k - u^{k-1}\|$$

$$\rightarrow \|u^{k+1} - u^k\| \leq L^k \|T(u^0) - u^0\|$$

$$|L| < 1 \Rightarrow \|u^{k+1} - u^k\| \rightarrow 0$$

$$\Rightarrow \exists!\ u \text{ s.t. } u^k \rightarrow u$$

Proof of Lax Milgram

$$\langle Au, v \rangle = \langle f, v \rangle \qquad \forall v \in V$$

$$a(u, v) = f(v)$$

$$\langle \underbrace{Au - f}_{V^*}, v \rangle \qquad \forall v \in V$$

Riesz repr. theorem

$$(z(Au - f), v)_V \qquad \forall v \in V$$

Construct a fixed point iteration (for $\rho \in \mathbb{R}$)

$$T_\rho(v) := v - \rho z(Av - f) \qquad T_\rho : V \rightarrow V$$

↑ How we see this is a contraction!

$$\|T_\rho(v) - T_\rho(w)\|_V^2 = \|v - w - \rho z A(v - w)\|_V^2$$

$$\|T_\rho(v)-T_\rho(w)\|_V^2 = \|v-w\|_V^2 + \rho^2\|A(v-w)\|_V^2 - 2\rho(A(v-w), v-w)_V$$

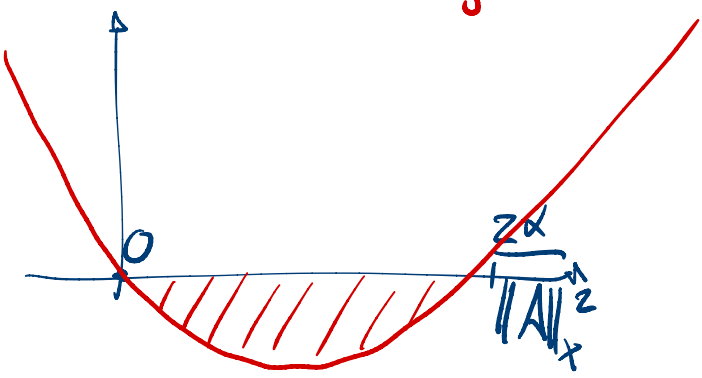
$$= \|v-w\|_V^2 + \rho^2\|A(v-w)\|_{V^*}^2 - 2\rho\langle A(v-w), v-w\rangle$$

$$\leq \|v-w\|^2 + \rho^2\|A\|_*^2\|v-w\|_V^2 - 2\rho\alpha\|v-w\|_V^2$$

$$\|T_\rho(v)-T_\rho(w)\|^2 \leq \underbrace{\left(\rho^2\|A\|_*^2 - 2\rho\alpha + 1\right)}_{<1}\|v-w\|_V^2$$

Can we choose ρ s.t. $\rho^2\|A\|_*^2 - 2\rho\alpha + 1 < 1$?

$$\rho\left(\rho - \frac{2\alpha}{\|A\|_*^2}\right) < 0$$



If $\rho \in \left(0, \frac{2\alpha}{\|A\|_*^2}\right)$ then T is a contraction

$\Rightarrow \exists! u \mid T(u) = u \Rightarrow$ $u = u - \rho z(Au - f)$

$\Rightarrow Au = f$

Galerkin methods:

Choose $V_h = \text{span}\{v_i\}_{i=0}^{N-1}$ s.t. $V_h \subset V$

→ Find $u_h \in V_h$ ⇒ $u_h = \sum_{J=0}^{N-1} u^J v_J$ s.t.

$$\langle A u_h, v_i \rangle = \langle f, v_i \rangle = f(v_i) \qquad i = 0, \dots, N-1$$

$$\sum_J \langle A u^J v_J, v_i \rangle = \langle f, v_i \rangle = f(v_i)$$

$$A_{iJ} := \langle A v_J, v_i \rangle$$

$$\underline{\underline{A}}\,\underline{u} = \underline{f} \qquad \underline{f}_i := f(v_i)$$

1): $\langle A u_h, v_h \rangle = \langle f, v_h \rangle \quad \forall v_h \in V_h$

$\langle A u, v_h \rangle = \langle f, v_h \rangle \quad \forall v_h \in V_h$

I'm saying that the error is A CONJUGATE wrt V_h

$\langle A(u - u_h), v_h \rangle = 0 \quad \forall v_h \in V_h$

The pic above etc

Cea's Lemma (orthogonality of error)

coercivity:

$$\alpha \|u - u_h\|^2 \leq \langle A(u - u_h), u - u_h \rangle = \quad \text{Cea's lemma} \quad \forall v_h \in V_h$$

$$= \langle A(u - u_h), u - v_h \rangle \leq \|A\|^* \|u - u_h\| \, \|u - v_h\|$$

$$\|u - u_h\|_V \leq \frac{\|A\|_*}{\alpha} \inf_{v_h \in V_h} \|u - v_h\|_V$$ ← Infimum

A priori error estimate

Trick: Make $\mathrm{dist}(V_h, V)$ small!

$$\mathrm{dist}(V_h, V) := \sup_{u \in V} \inf_{v_h \in V_h} \|u - v_h\|$$

if you can $\mathrm{dist}(V_h, V) < \varepsilon$ ⇒ V is ε approximable by V_h

For Lagrangian Finite elements of order k we bound $u - \mathcal{I}(u)$ in terms of Sobolev norms of u:

If $\mathcal{I}(u)$ is Lagrange interpolation of u, then

$$\|u - \mathcal{I}(u)\|_{m,T} < C\, h_T^{k+1-m} |u|_{k+1,T}$$

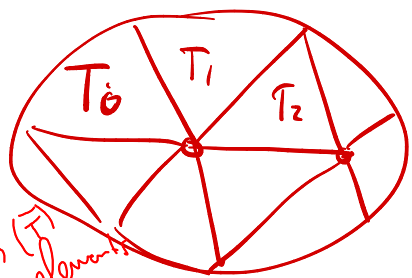
T ← "intervals" or "elements"; h_T ← measure of the element

on each "element" T of Ω_h, discretization of Ω. 1D: $\Omega = [0,1]$ $\quad T = [x_i, x_{i+1}]$

$$h_T = (x_{i+1} - x_i)$$

$$\Omega_h = \overset{\circ}{\overline{\bigcup_{i=1}^{M} \overline{T_i}}}$$

↑ internal part ($\overset{\circ}{\cdot}$) of the union of all the closures ($\overline{T}$) of the elements



$$|u|_{\alpha,T} := \left(\sum_{|m|=\alpha} \int_T \left(D^m u \right)^2 \right)^{\frac{1}{2}}$$

$D^m u$: weak derivative of order m

$$\|u\|_{\alpha,T} := \left(\sum_{|m|\le\alpha} |u|^2_{\alpha,T} \right)^{\frac{1}{2}}$$

m is multi index $m_1, m_2, \dots, m_d$ in dimension d

and $|m| = \sum_{i=1}^{d} |m_i|$

in 1D: $|u|_{\alpha,T} := \left(\int_T \left(u^{(\alpha)} \right)^2 \right)^{\frac{1}{2}}$

Exercise

we try to approx a finite dim space V, with a smaller finite dim space V_h

$V = \mathbb{R}^n \qquad V_h = \text{span}\{v_i\}_{i=0}^{m-1} \qquad m \le n \qquad v_i \in \mathbb{R}^n \ (\equiv V)$

$$(u, v) := \sum_{i=0}^{n-1} u^i v^i$$

$$\|u\| := \left(\sum_{i=0}^{n-1} (u^i)^2 \right)^{\frac{1}{2}}$$

Apply Galerkin to this problem:

$$a : V \times V \longrightarrow \mathbb{R}$$

$$\underline{\underline{A}} : V = \mathbb{R}^n \longrightarrow V^* \ (\equiv V) = \mathbb{R}^n$$

$\underline{\underline{A}}$ is a $\mathbb{R}^{n\times n}$ matrix, SPD, v_i are $\mathbb{R}^n$ vectors

Galerkin approx:

$$\langle \underline{\underline{A}} \, u^j \underline{v}_j, \underline{v}_i \rangle = \langle \underline{f}, \underline{v}_i \rangle \qquad i = 0, \dots, m-1$$

Should give me a $\mathbb{R}^{m\times m}$ matrix A

$$A_{ij} = \langle \underline{\underline{A}} \underline{v}_j, \underline{v}_i \rangle = \underline{v}_i^T \underline{\underline{A}} \underline{v}_j$$

$$F_i = \langle \underline{f}, \underline{v}_i \rangle = \underline{v}_i^T \underline{f}$$

$V_{id} := (\underline{v}_d)_i \qquad V \in \mathbb{R}^{n \times m}$

Matrix containing the basis $\{\underline{v}_d\}_{d=0}^{m-1}$

$$\boxed{V^T \underline{\underline{A}} V u = V^T \underline{f}}$$

$$A u = F$$

$V \in \mathbb{R}^{n \times m} \qquad \underline{\underline{A}} \in \mathbb{R}^{n \times n} \qquad \underline{f} \in \mathbb{R}^n$

$u \in \mathbb{R}^m$

If we manage to construct $V^T A V = \underline{Id}$ (what I wish to get is)

then $u = V^T f$ (nothing has to be inverted here) $\qquad u \simeq V V^T f$

$V u \simeq \underline{u} \qquad u_0 = 0$

Start with $\varphi_0 = r_0 = A u_0 - f = -f$

$\varphi_0^T \underline{\underline{A}} \varphi_0 := \beta_0^2 \longrightarrow v_0 = \frac{\varphi_0}{\beta_0}$

$\longrightarrow v_0^T \underline{\underline{A}} v_0 = 1$

construct v_{k+1} s.t. $\boxed{v_{k+1}^T A v_j = 0 \quad \forall j \leq k}$

and $\boxed{v_{k+1}^T A v_{k+1} = 1}$

$\Rightarrow$ at most after n iteration $u = V^T V f$

And you get exact for the CG method

Lecture 29 - LA

FEM / FDM 2D

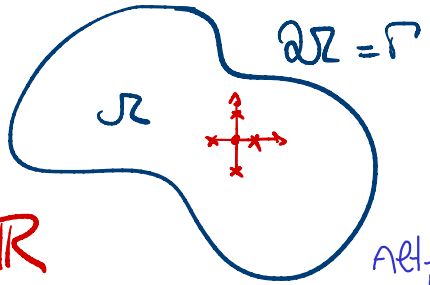
14.01.2021

One problem formulation is

1) $-\Delta u = f$ in Ω

$u = 0$ on $\partial\Omega$

$N=2$



LAPLACIAN

$$-\Delta u := -\frac{\partial^2 u}{\partial x_0^2} - \frac{\partial^2 u}{\partial x_1^2} = \sum_{d=0}^{N-1} -\frac{\partial^2 u}{\partial x_d^2}$$

$u : \Omega \to \mathbb{R}$

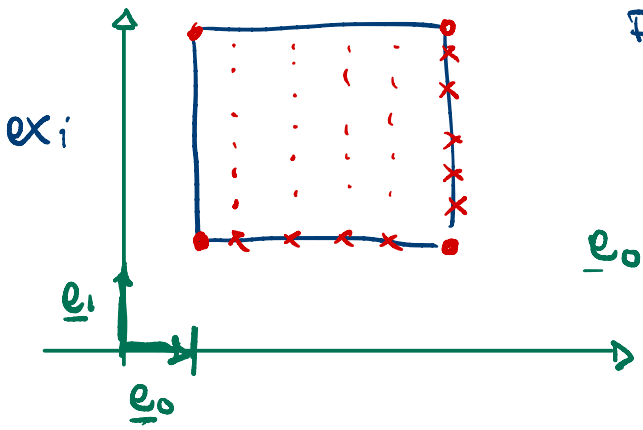
sometimes:

Alternatively, another problem that can appear is

$-\Delta u + u = f$

2) $\frac{\partial u}{\partial n} = 0$ on $\partial\Omega$

For FD Ω has to be "simple" and "regular"



Fix a grid of points $X_{ij} \in \Omega \subset \mathbb{R}^2$

$\Omega = [a,b] \times [c,d]$

$e_0 := (1,0)$ $e_1 = (0,1)$

$$\underline{x} = \underline{e}_0 x^0 + \underline{e}_1 x^1 = \underline{e}_i x^i$$

FD: along e_0: choose n points as $\{\varphi_i\}_{i=0}^{n-1} := [a, a+\dots, \dots, b]$

along e_1: choose m points as $\{\psi_\alpha\}_{\alpha=0}^{m-1} := [c, \dots, \dots, d]$

construct $X_{\alpha i} \in \mathbb{R}^2$ ($n \times m$ points) $\alpha \in [0,\dots,m-1]$, $i \in [0,\dots,n-1]$

$(X_{\alpha i})_d \to d$ comp. of $X_{\alpha i}$

$$X_{\alpha i} := (\varphi_i, \psi_\alpha) \qquad \{X_{\alpha i d}\}_{\substack{\alpha=0,\dots,m-1 \\ i=0,\dots,n-1 \\ d=0,1}}$$

You want same N of pts on both x and y direction, for 2 reasons: 1) simplicity 2) if I flatten the things I get $m \cdot n \cdot 2$ or $n \times m$ points in dimension 2

$$\text{flat}(X) \in \mathbb{R}$$

The X object has elements

$$X_{\alpha i d} = \begin{cases} \varphi_i & \text{if } d = 0 \leftarrow \text{first direction} \\ \psi_\alpha & \text{if } d = 1 \leftarrow \text{second direction} \end{cases}$$

$X \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^N$ $\quad shape(X) = (m,n,N)$

$X_{ij} \in \mathbb{R}^N$ $\quad \underline{x}$

$X_{ijd} \in \mathbb{R}$ $\quad x_i$

$X_d \in \mathbb{R}^{n \times N}$

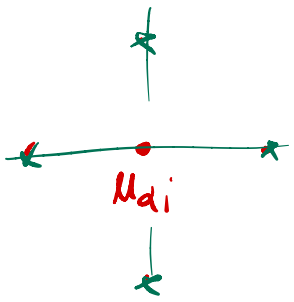
Now we assume we know u on X (that is only on the grid) only (on all $n \cdot m$ points $X_{di} \in \Omega$ such that)

$u_{di} := u(X_{di})$ ← u is a function that if you pass it a "matrix" (matrix element) you get a matrix (matrix element)

We apply CFD to u_{di} on both directions:

CFD^2-1D: $-v_i'' \sim \frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2}$ (along $i \rightsquigarrow e_0$)

CFD^2-1D along $d \rightsquigarrow e_1$ $\quad -w_d'' \sim \frac{-w_{d-1} + 2w_d - w_{d+1}}{h^2}$



you assume that the approximation at u_{di} depends only on the value at these 5 points

$v_i := u_{di}$
$w_d := u_{di}$

$-\Delta u_{di} \simeq \frac{1}{h^2}\left[4u_{di} - u_{d-1\,i} - u_{d+1\,i} - u_{d\,i-1} - u_{d\,i+1}\right]$

$-\tilde{\Delta} u_{di} = f_{di} = f(X_{di})$

$Au = f$

$A \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$

because I have 2 directions $d = 0$ and 1

Construct

$$A_{\alpha i\ \beta J} \quad \text{s.t.} \quad \sum_{\beta,J} A_{\alpha i\ \beta J}\, u_{\beta J} = -\tilde{\Delta}\, u_{\alpha i}$$

$$A_{\alpha i\ \beta J} = \begin{cases} 4 & \beta=\alpha \quad J=i \\ -1 & \beta=\alpha-1 \quad J=i \\ -1 & \beta=\alpha+1 \quad J=i \\ -1 & \beta=\alpha \quad J=i-1 \\ -1 & \beta=\alpha \quad J=i+1 \\ 0 & \text{all other cases} \end{cases}$$

$$A_{\alpha i\ \beta J}\, u_{\beta J} = f_{\alpha i}$$

$$\Downarrow \text{ reshape}$$

$$\sum_J \tilde{A}_{IJ}\, \tilde{u}_J = \tilde{f}_J$$

$$I, J \in [0, \dots, m \cdot n - 1]$$

$$\tilde{u}_J = \sum_I \left(\tilde{A}^{-1}\right)_{JI} \tilde{f}_I$$