Alessio Hu (12345678)

Numerical Analysis Summary

Numerical Analysis
Prof. Perotto Simona
- Politecnico di Milano

Something:

Something else

A.Y. 2022-2023

Contents

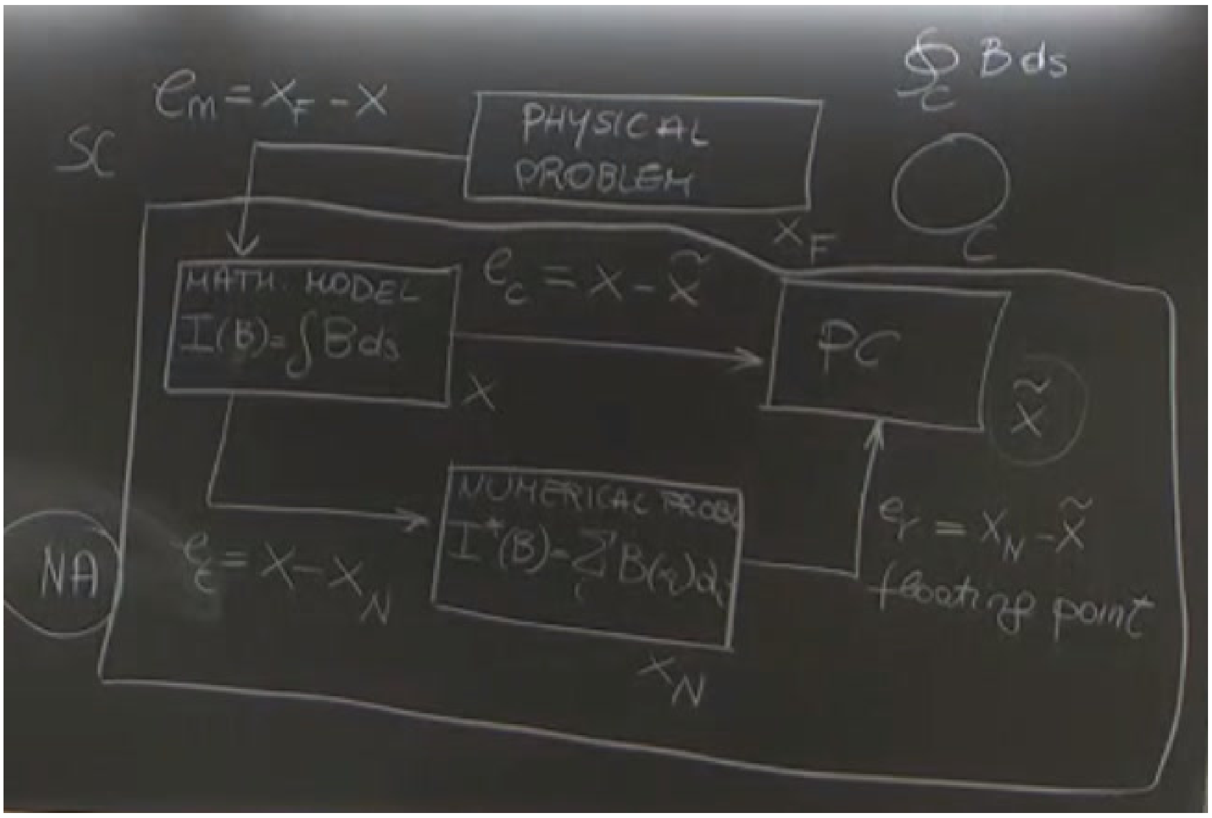
1 Intro

1.1 Numerical Analysis and Errors

M = Math
CS = Computer Science
E = Engineering

Numerical Analysis = M∩CS
Basic methods to approach math problems

Scientific Computing = M∩CS∩E Take a problem and replicate it on a digital device to understand better the situation



Where:

$$\begin{cases} x_F \text{ solution of physical model} \\ x \text{ solution of mathematical model} \\ \tilde{x} \text{ does not substitute reality simulation} \end{cases}$$

We replaced the integral with a summation, ia PC there is no concept of infinity.
What to do after we observe what's going on? Use a better model or a better $x \leftrightarrow \tilde{x}$ mapping.

Errors:

$$\begin{cases} e_m = x_F - x \text{ modelling error between physical problem and mathematical model} \\ e_c = x - \tilde{x} \text{ computational error} = e_t + e_r \begin{cases} e_t = x - x_N \text{ truncation error} \\ e_r = x_N - \tilde{x} \text{ rounding error, floating point approximation} \end{cases} \end{cases}$$

Kinds of errors:

- $|x - \tilde{x}|$
  Absolute error, $|e_c|$. Consider absolute error based on model

- $\frac{|x-\tilde{x}|}{|x|}$
  Relative error, more meaningful, like percentage

1.2 Floating point representation

Not really important, skipping...

2 Nonlinear equations

f, we want to find $\alpha \in \mathbb{R}$ zero of f such that $f(\alpha) = 0$

It is not easy to find the zero of a function when $\mathbb{P}_n$ $n > 4$. From here we need to approximate the zeros of a nonlinear function.

An example of physical phenomenol: the ideal gas equation:

$$pV = nRT$$

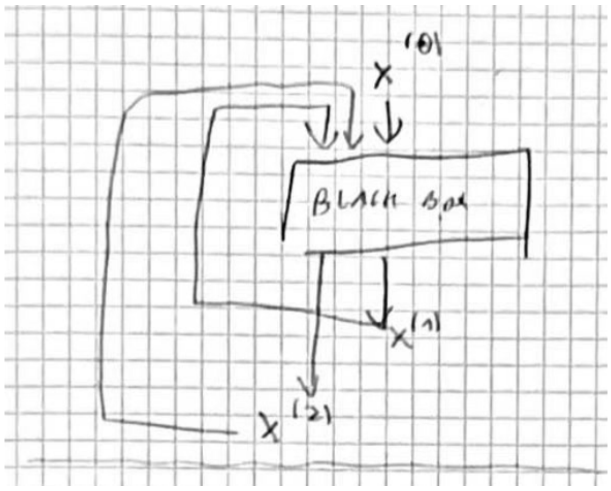
To find V

$$\left[p + a\left(\frac{N}{V}\right)^2\right](V - Nb) = kNT$$

A nonlinear equation, although we know the pressure, the temperature and the constants a and k, it is not easy to solve.

We use **iterative methods**

$$\underbrace{x^{(0)}}_{\text{Initial guess}}$$



$$x^{(k)} \simeq \alpha$$

Ideally I want

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

Convergence, or equivalently

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

$$\lim_{k\to\infty} e^k = 0$$

But we have to decide when to stop this approximation, **stopping criteria**: set a maximum number of iterations and a tolerance error.

We are looking at the approximation error

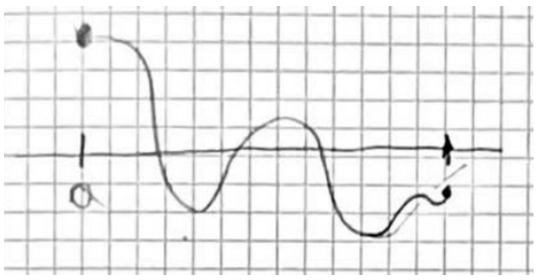
$$?\ \alpha \in \mathbb{R}\ s.t.\ \underbrace{f(\alpha) = 0}_{\text{non linear}}$$

Two methods: bisection and Newton method

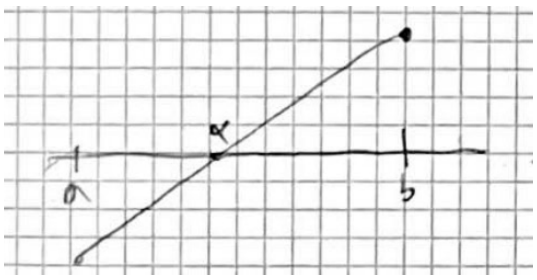
2.1 Bisection method

In mathematics, the bisection method is a root-finding method that applies to any continuous function for which one knows two values with opposite signs.
We first have to fix the hypothesis, which coincides on the hypothesis of the zero of nonlinear continuous functions:

1. $f \in C^0([a,b])$
   Set of all continuous functions in $[a,b] \subset \mathbb{R}$
2. $f(a)f(b) < 0$



(a) The example has more zeros even, this is too much for us, even one is enough



(b) The function is taking values at opposite sign at endpoints, which means that the function f has at least one zero in the interval

Starting from an interval, we shrink it:

$$\alpha \in I^{(0)} = [a,b]$$
$$\alpha \in I^{(1)} \subset I^{(0)}$$
$$|I^{(1)}| = \frac{|I^{(0)}|}{2}$$
$$\alpha \in I^{(2)} \subset I^{(1)}$$
$$|I^{(2)}| = \frac{|I^{(1)}|}{2}$$
$$\vdots$$

Collection of intervals that are getting smaller and smaller, that all contain α (the zero), so at the end we will get to α

Strength point of bisection: **always convergent, but has a lot of drawbacks too**.

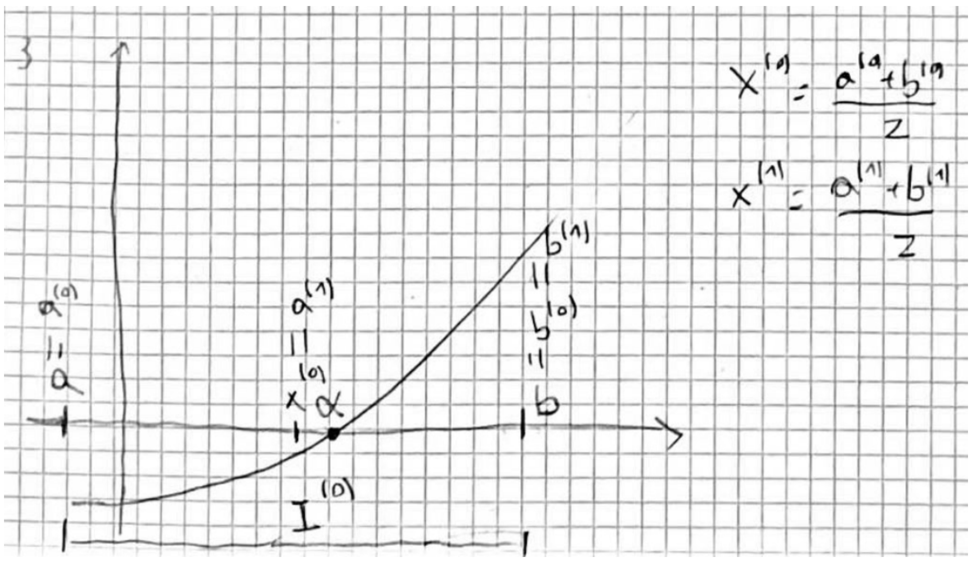
But we are looking for $x^{(k)}$ that approximates α

$$?\, x^{(k)} \simeq \alpha$$

To do this we consider the midpoint of the interval

We see that in this example the first midpoint is already close to α. Now we have to choose a new subinterval $I^{(1)}$, we will choose the right subinterval as we need to satisfy the hypothesis that the extremes have opposite signs. We then continue iteratively.

The algorithm is similar to binary search. Formally:
Inputs: $a = a^{(0)}$, $b^{(0)} = v$, f; TOL, $Nmax$
Where TOL is the tolerance and $Nmax$ the maximum number of iterations.



$x^{(0)} = \frac{a^{(0)}+b^{(0)}}{2}$
$x^{(1)} = \frac{a^{(1)}+b^{(1)}}{2}$
$a^{(0)} = a$
$a^{(1)} = x^{(0)}$
α
$b^{(1)} = b^{(0)} = b$
$I^{(0)}$

```
        while(true)
            $x^{(k)} = (a^{(k)} + b^{(k)})/2$
            if($f(x^{(k-1)}) = 0$) break;
            if $f(a^{(k-1)})f(x^{(k-1)}) < 0$
                $a^{(k)} = a^{(k-1)}, b^{(k)} = x^{(k-1)}$;
            else
                $a^{(k)} = x^{(k-1)}, b^{(k)} = b^{(k-1)}$;
            end
```

Regarding the intervals, we see that:

$$|I^{(k)}| = \frac{b-a}{2^k} = \frac{|I^{(0)}|}{2^k}$$

$$|e^{(k)}| = |\alpha - x^{(k)}| < \frac{|I^{(k)}|}{2} = \frac{b-a}{2^{k+1}}$$

$$\lim_{k\to\infty} |e^{(k)}| = (b-a) \lim_{k\to\infty} \left(\frac{1}{2}\right)^{(k+1)} = 0$$

Regarding the tolerance

$$?\, k \text{ s.t. } |e^{(k)}| \leq TOL = 10^{-9}$$

$$\frac{b-a}{2^{(k+1)}} \leq TOL$$

$$\frac{b-a}{TOL} \leq 2^{(k+1)}$$

$$\log_2\left(\frac{b-a}{TOL}\right) \leq k+1$$

$$k \geq \log_2\left(\frac{b-a}{TOL}\right) - 1$$

The right member will represent the *Nmax*

$$Nmax = \underbrace{\left\lceil \log_2\left(\frac{b-a}{TOL}\right) - 1 \right\rceil}_{\log_2(\cdots)-1} = \left\lceil \log_{10}\left(\frac{b-a}{TOL}\right) / \log_{10} 2 - 1 \right\rceil$$

2.1.1 Pros and Cons

Pros:

- Converges
- Can compute maximum number of iterations

Cons:

- We are losing the monotonicity of the method, a subsequent guess is not guaranteed to find a x closer to α w.r.t. the previous x, **not monotonic w.r.t. to error, the error does not necessarily decreases at each step, we have no convergence order**
- Cannot find α in a single step
- We are only exploiting the fact that the function is changing sign, function values of f are ignored

2.2 Newton method

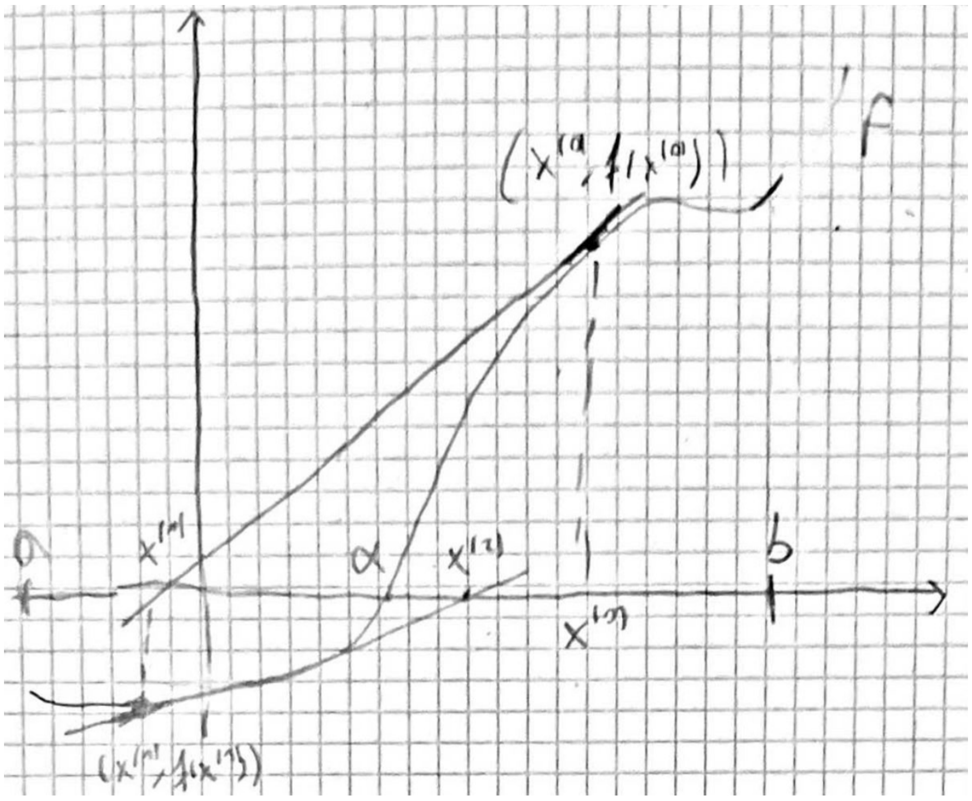
Nmax has to hold for all continuous functions in that interval $[a,b]$, so it has to be large.

In Newton we are not only exploiting the sign of f, but also the values. It is the most powerful method, and demands that:

$$f \in C^1([a,b])$$

Set of functions continuous in their first derivative

Replace f with the tangent line at point of our initial guess $x^{(0)}$, then take the intersection of the tangent with the x axis, that will be our $x^{(1)}$. Then repeat:



tg to f at $(x^{(k)}, f(x^{(k)}))$
$y(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)})$
$x^{(k+1)}$ s.t. $y(x^{(k+1)}) = 0$

$f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0$
From this equality we find $x^{(k+1)}$:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k \geq 0$$

Assuming $f'(x^{(k)}) \neq 0$
It can be proved that this algorithm is just a truncation of the Taylor expansion

2.2.1 Taylor expansion

We must first decide the center and where to evaluate the expansion. In our case the center is $x^{(k)}$, we evaluate at $x^{(k+1)}$

$$f(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + O\left((x - x^{(k)})^2\right) x^{(k+1)}$$

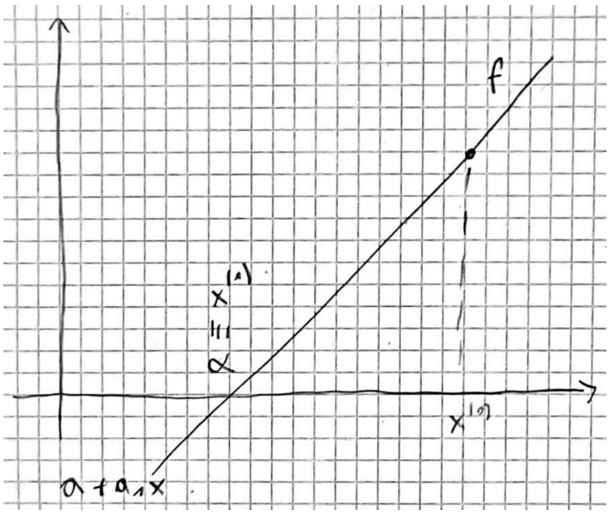
So if we evaluate at $x^{(k+1)}$, we simply replace x. We neglect the big O term and if k is sufficiently large, we can approximate to α.

$$0 = f(\alpha) \simeq f(x^{(k+1)}) \cong f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)})$$

Which is exactly the Newton method

2.2.2 Comparison with bisection

Newton can identify α in a single step.
Geometrical proof:



Analytical proof:

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} = x^{(0)} - \frac{a_0 + a_1 x^{(0)}}{a_1} = -\frac{a_0}{a_1}$$

Bisection converges without initial guess, but Newton is better even though it needs an initial guess. **If initial guess not close to the zero, we do not converge**

2.2.3 Convergence

Two hypothesis

H1) $x^{(0)}$ sufficiently close to α. But we do not know α, we could:

- Graphically plot it
- Do some steps of the bisection and use some outputs of it for Newton: **bisection-Newton: predictor-corrector**, use a weaker method then a stronger one. With bisection we know that we will converge, even if slowly, then after a desired steps (sufficiently close) we use Newton which is faster

H2) α is a simple zero of f

$$\begin{cases} f(\alpha) = 0 \\ f'(\alpha) \neq 0 \end{cases}$$

Reminder, an α is a zero of order m if

$$\begin{cases} f(\alpha) = f'(\alpha) = f''(\alpha) = \cdots = f^{(m-1)}(\alpha) = 0 \\ f^{(m)}(\alpha) \neq 0 \end{cases}$$

$\Rightarrow$ **Newton is convergent**

Adding a third hypothesis

H3) $f \in C^2([a,b])$, we can say that the following limit holds:

$$\lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{[x^{(k)} - \alpha]^2} = \underbrace{\frac{f''(\alpha)}{2f'(\alpha)}}_{\text{C}}$$

In general **convergence order** equal to P if $\exists\, C$ independent from k, such that

$$\frac{|x^{(k+1)} - \alpha|}{|x^{(k)} - \alpha|^P} \leq C,\ \forall\, k \geq k_0$$

If $P = 1$, linear convergence. In our case $P = 2$, quadratic convergence. In general, with a higher convergence order the error reduces:

$x^{(k)} - \alpha = 10^{-2}$
$x^{(k)} - \alpha \simeq 10^{-4}$ $\quad P = 2$
$x^{(k)} - \alpha \simeq 10^{-6}$ $\quad P = 3$

The constant C does not have any requirement, but in some sense it is slowing the convergence (for $C = 10, P = 3,$ the error $\simeq 10^{-6} * 10 = 10^{-5}$), but for $P = 1$ reduction of the error not guaranteed if $C = 1$, so:

$$P = 1 \to C < 1$$

In the Newton case, if H1, H2 and H3 hold, the convergence of Newton is **quadratic**:

$$C = \frac{f''(\alpha)}{2f'(\alpha)}$$

2.2.4 Modified Newton scheme

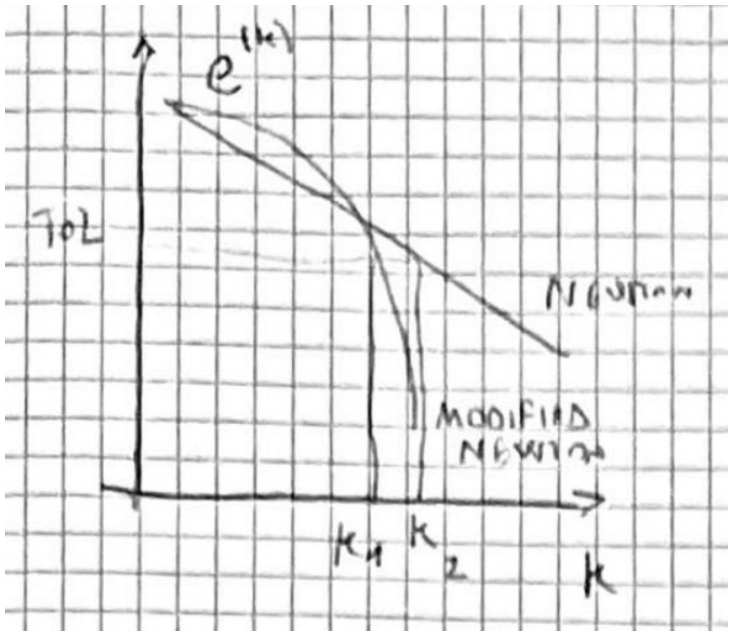
What if H2 does not hold, can we still use Newton? Yes, but we lose the quadratic order of the convergence.
If α is a multiple zero of f (multiplicity m) and if $x^{(0)}$ is sufficiently close to $\alpha \Rightarrow$ Newton converges linearly ($P = 1$, we lost an order of convergence)

We can use the modified Newton scheme:

$$x^{(k+1)} = x^{(k)} - m\frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k \geq 0$$

Example:

$$f(x) = (x-1)\log(x)$$
$$\alpha = 1$$
$$m = 2$$



2.2.5 System of nonlinear equations, vector

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

Example

$$\begin{cases} f_1(x_1, x_2) = x_1^3 + \sin x_2 = 0 \\ f_2(x_1, x_2) = -x_1\sqrt{x_2} + \operatorname{tg}\left(\frac{x_2}{3x_1}\right) = 0 \end{cases}$$

We can rewrite this complex model to a more manageable form, a **vectorial way**

$$\overrightarrow{x} = [x_1, x_2, \cdots, x_n]^T$$

$$\overrightarrow{f} = [f_1(\overrightarrow{x}), f_2(\overrightarrow{x}), \cdots, f_n(\overrightarrow{x})]^T$$
$$\overrightarrow{0} = [0, \cdots, 0]^T \in \mathbb{R}^n$$

So we can rewrite the system of nonlinear equations in:

$$\overrightarrow{f}(\overrightarrow{x}) = \overrightarrow{0}$$

Now applying the Newton method, we jsut introduce the vectors that contain the k-approximation error:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

Now with

$$\overrightarrow{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}\right]^T \in \mathbb{R}^n$$

$$\overrightarrow{x}^{(k+1)} = \left[x_1^{(k+1)}, x_2^{(k+1)}, \cdots, x_n^{(k+1)}\right]^T \in \mathbb{R}^n$$

$$\overrightarrow{f}(\overrightarrow{x}^{(k)}) = \left[f_1(\overrightarrow{x}^{(k)}), f_2(\overrightarrow{x}^{(k)}), \cdots, f_n(\overrightarrow{x}^{(k)})\right]^T \in \mathbb{R}^n$$

What about the derivative? Use Jacobian

$$x^{(k+1)} = x^{(k)} - \underbrace{\frac{f(x^{(k)})}{f'(x^{(k)})}}_{\delta x^{(k)}}$$

$$f'(x^{(k)})\delta x^{(k)} = -f(x^{(k)})$$

$$\overrightarrow{x}^{(k+1)} = \overrightarrow{x}^{(k)} + \delta \overrightarrow{x}^{(k)}$$

$$?\, \delta \overrightarrow{x}^{(k)} = -\overrightarrow{f}(\overrightarrow{x}^{(k)})$$

The Jacobian

$$(J_F)_y = \frac{\partial f_i}{\partial x_j} \qquad i, j = 1, \cdots, n$$

The question mark becomes

$$\underbrace{J_F\left(\overrightarrow{x}^{(k)}\right)}_{\mathbb{R}^{n\times n}} \underbrace{\delta x^{(k)}}_{\mathbb{R}^n} = \underbrace{-\overrightarrow{f}(\overrightarrow{x})^{(k)}}_{\mathbb{R}^n}$$

2.2.6 Bisection - Newton method

They are predictor-corrector mtehods. Newton is conditional convergent (the initial guess bust be close to the root), while bisection is unconditional convergent: **by using bisection we are sure that the intial guess is close to the root (predictor) then use Newton as corrector**.

2.3 Convergence order

$$\{x^{(k)}\} \simeq \alpha$$

Convergence order equal to p if $\exists\, c$ independent from k, such that $\frac{|x^{k+1}-\alpha|}{|x^k-\alpha|^p} \leq c$, $\forall\, k \geq k_0$

- **Bisection**, error no monotone, no convergence order
- **Newton**

  $$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k \geq 0$$

  In Newton, under assumptions that:

  - $f(\alpha) = 0$
  - $f'(\alpha) \neq 0$

  α simple zero, we have

  - Convergence order of 1 if $f \in C^1$
  - Convergence order of 2 if $f \in C^2$

  If α not simple, $f \in C^2$, α is a zero of order m $(f(\alpha) = 0, \cdots, f^{(m-1)}(\alpha) = 0, f^{(m)}(\alpha) \neq 0)$, more simply when there is a power the zero is of the order of that power.

- **Modified Newton method**, unlike bisection, with newton we can compute the convergence order. Can find non-simple zeros, we have a convergence order of 2 for example. We use a different update rule:

$$x^{(k+1)} = x^{(k)} - m\frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k \geq 0$$

$$e = \{|x^k - \alpha|\}_k$$
$$e^k = |x^k - \alpha|$$
$$\text{Convergence order} = p = \frac{\log\left[\frac{e^{k+2}}{e^{k+1}}\right]}{\log\left[\frac{e^{k+1}}{e^k}\right]}$$

Do not mix up exponential and error, the e^k there stands for the error vector considering index from k to end

2.4 Stopping criteria/point

$$\underbrace{|x^{(k)} - \alpha|}_{e^{(k)}} \leq \underbrace{CS}_{\text{Error estimator}} < TOL$$

$$\left|e^{(kmin)}\right| = \left|x^{(kmin)} - \alpha\right| \leq CS = C\left|x^{(kmin+1)} - x^{(kmin)}\right| \leq TOL\ (e.g. = 10^{-9})$$

A constant that multiplies our error estimator S. If large like 10^4 we lose 4 orders. It is called the **reliability**, if

- $C = O(1)$, estimator reliable
- $C = O(10^s)$, not reliable

Two kinds of estimators for the error, when one not reliable, can rely on the other one

$$S = \begin{cases} x^{(k+1)} - x^{(k)} \text{ increment, if convergence this difference becomes smaller and smaller} \\ \qquad \textbf{Newton} \\ r^{(k)} = f(x^{(k)}) \text{ residual, huge if } x^{(k)} \text{ far from } \alpha \text{, less it is, smaller is the error} \\ \qquad \textbf{Bisection} \end{cases}$$

The increment one, cycle till:

$$|x^{(k+1)} - x^{(k)}| > TOL \,\&\&\, i < TOL$$

2.4.1 Reliability of the residual

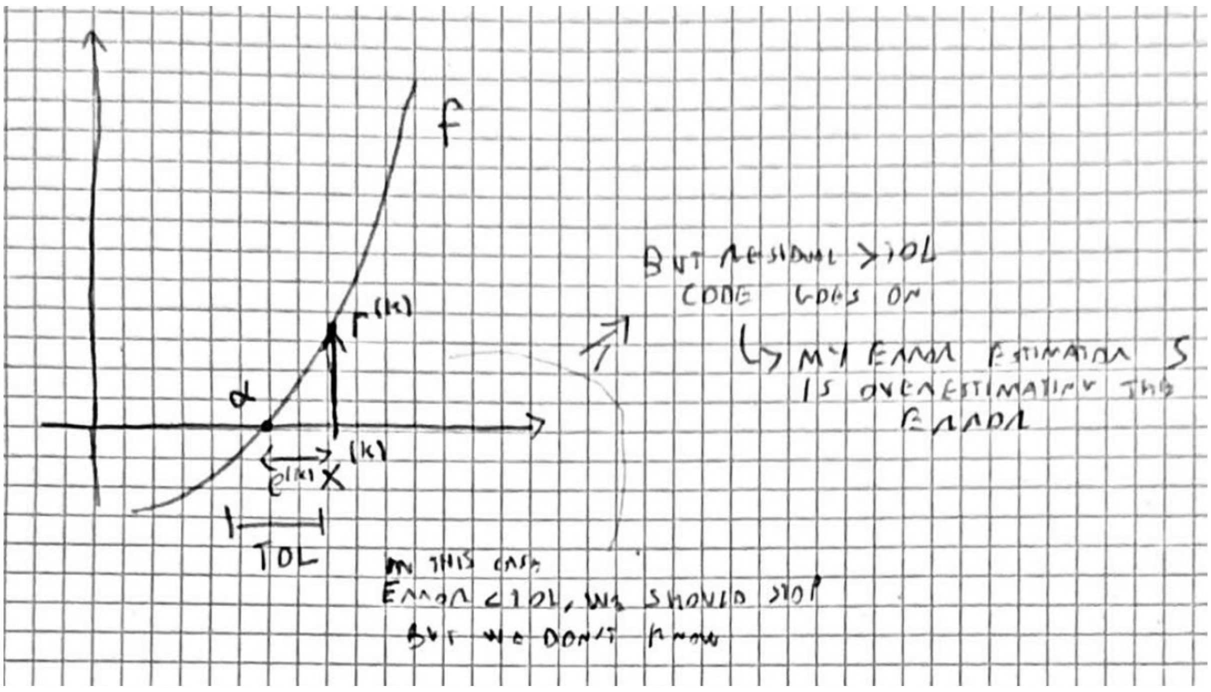

f
$r^{(k)}$
α
$e^{(k)}$
$x^{(k)}$
TOL
BUT RESIDUAL > TOL
CODE GOES ON
MY ERROR ESTIMATOR S IS OVERESTIMATING THE ERROR
IN THIS CASE ERROR < TOL, WE SHOULD STOP
BUT WE DON'T KNOW

Figure 1: Overestimating

My error estimator S is overestimating the error $e^{(k)}$

$$|f'(\alpha)| >> 1$$

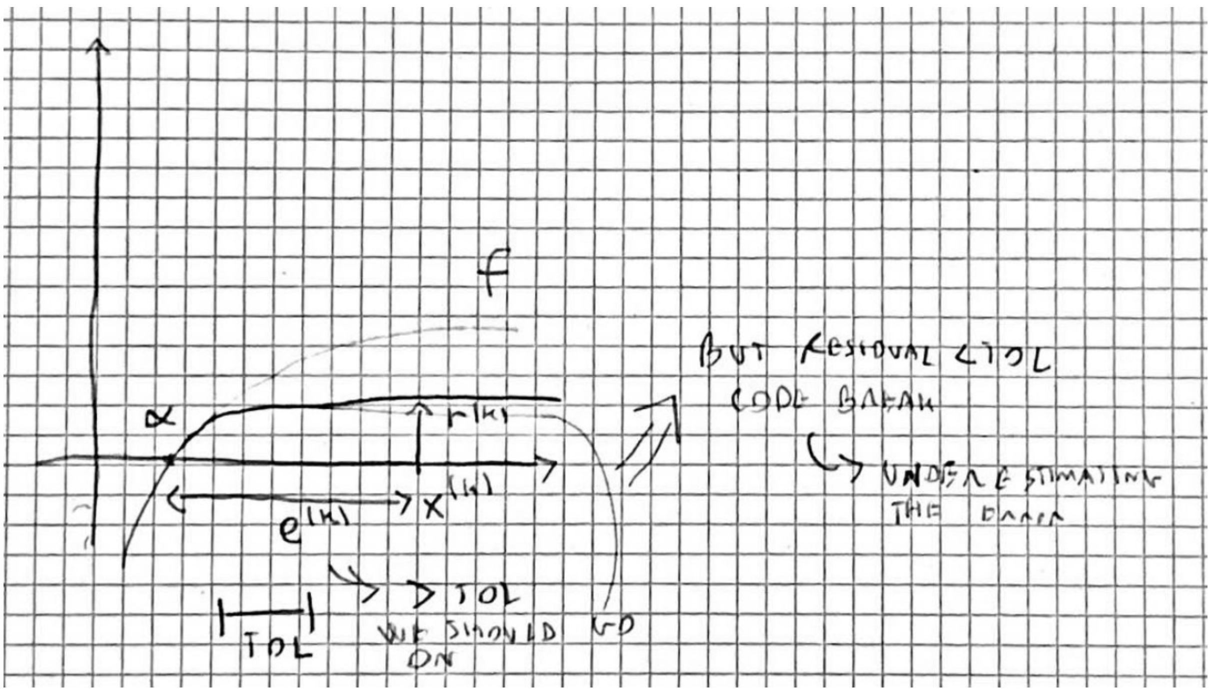


Figure 2: Underestimating

My error estimator S is underestimating the error $e^{(k)}$

$$|f'(\alpha)| << 1$$

What's better? Better when overestimating, we do a little more work, but at the end we get the better result

So this estimator is reliable when:

$$|f'(\alpha)| \simeq 1$$

2.5 Fixed Point Method

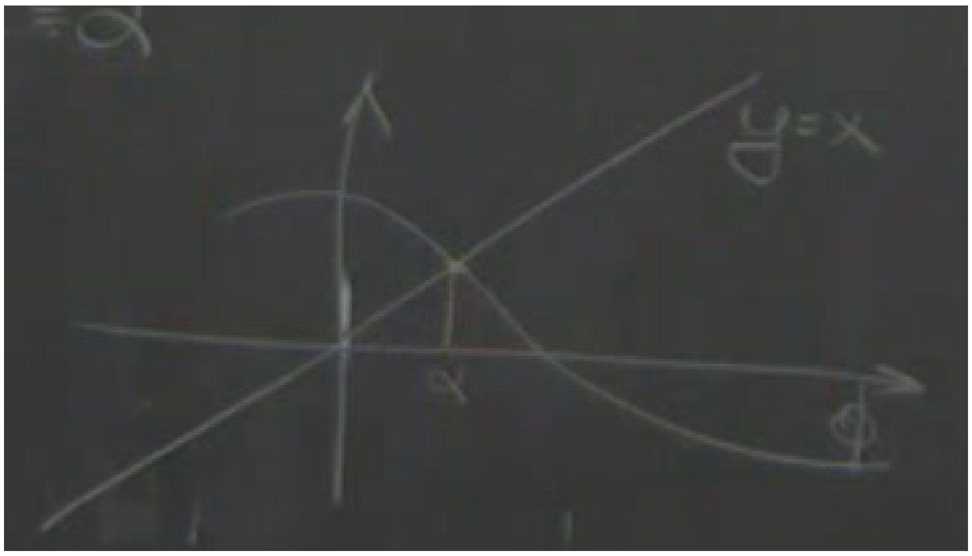
A fixed point α for a function ϕ is:

$$\phi : [a,b] \subset \mathbb{R} \to \mathbb{R}$$

$$\alpha \in \mathbb{R} \text{ fixed point } \phi(\alpha) = \alpha$$

Geometrically we are looking for the intersection of:

$$\begin{cases} y = \phi(x) \\ y = x \qquad \text{Bisector line} \end{cases}$$



- Does any function has a fixed point? No, for example parallel line or exponential function, it grows fast and does not encounter the bisector line
- Does a function admit more fixed points? Yes, function can encounter the bisector line multiple times

How can a fixed point interest us? We can talk about a sort of duality problem:

$$\begin{cases} ?\ \alpha \in \mathbb{R}\ st\ f(\alpha) = 0 \qquad \text{Our original problem of zeros} \\ ?\ \alpha \in \mathbb{R}\ st\ \phi(\alpha) = \alpha \qquad \text{Fixed point problem} \end{cases}$$

2.5.1 Problems correlation

The transition:

$$\left[\begin{array}{ccc}
?\,\alpha\in\mathbb{R}\;st\;f(\alpha)=0 & \Leftrightarrow & ?\,\alpha\in\mathbb{R}\;st\;\phi(\alpha)=\alpha \\
f(x)=0 & \Leftrightarrow & \underbrace{f(x)+x}_{\phi(x)}=x \\
\\
\Rightarrow \text{Hp: } f(\alpha)=0 & & \Leftarrow \text{Hp: } \phi(\alpha)=\alpha \\
\text{Th: } \phi(\alpha)=\alpha & & \text{Th: } f(\alpha)=0 \\
\phi(\alpha)=\underbrace{f(\alpha)}_{=0}+\alpha=\alpha & & \underbrace{\phi(\alpha)}_{=\alpha}=f(\alpha)+\alpha=\alpha
\end{array}\right]$$

But the ϕ is not unique, we can build more and different fixed point functions (e.g. add constant multiplier), this is a **advantage**, we can pick a function that is for sure to have a fixed point

2.5.2 The method with Newton and Bisection

With an **iterative process**, now we try to solve

$$?\,\alpha\in\mathbb{R}\;st\;\phi(\alpha)=\alpha$$

With iterative method

$$x^{(k+1)}=\phi(x^{(k)}) \qquad k\geq 0$$

With initial guess $x^{(0)}$. Considering the previous methods bisection and Newton, can we rewrite them as a fixed point method?

- **Newton**: we can just pick a fixed point function as:

$$\phi_N(x)=x-\frac{f(x)}{f'(x)}$$

- **Bisection**: no, the midpoint depends on two variables, so bisection is not an example of fixed point method

2.5.3 Convergence

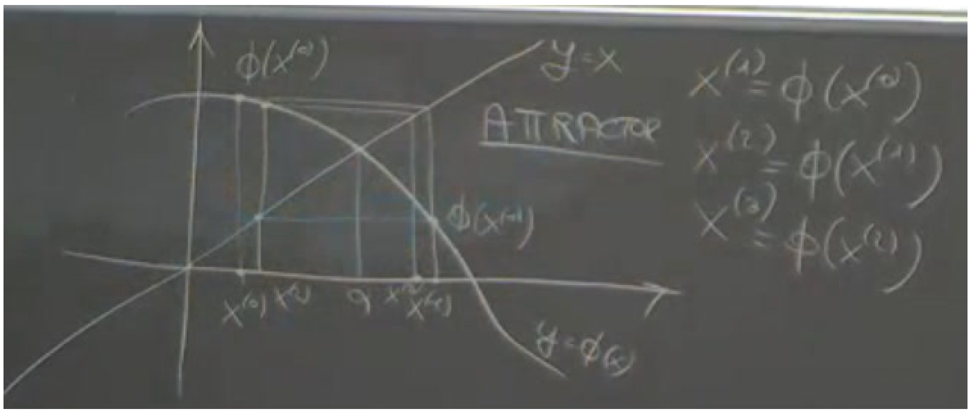


Figure 3: Attractor

All even at left side, all odd at right side.
Are we converging to α? Yes, the iterations are moving closer and closer to α, which is an **attractor**.

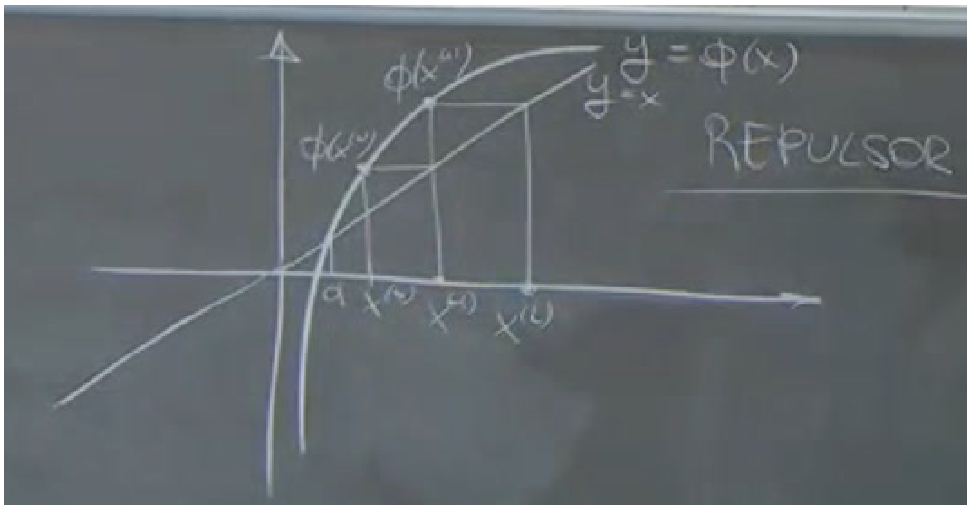


Figure 4: Repulsor

In this case we are unlucky, we diverge, **repulsor**.

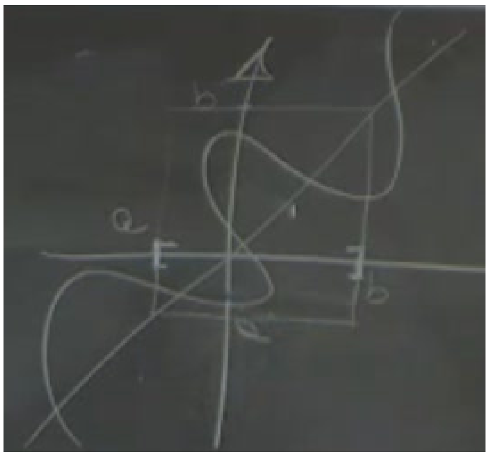
These are two particular cases, in total there are four: left-right convergent, left-right divergent, one direction convergent, one direction divergent.

Can we identify some features that are responsible for the convergent and divergent trend? The **value of the derivative $\phi'(\alpha)$ if less or greater than 1**

2.5.4 Global and Local Convergence Results

$$x^{(k+1)} = \phi(x^{(k)})$$

- **Global result**
  1. Let $\phi \in C^0([a,b])$ and s.t. $\phi(x) \in [a,b] \; \forall \, x \in [a,b]$

     We are demading that our function ϕ has to take values inside that rectangle, must be limited in a certain portion of the plane.
     Under this hypothesis, we can prove that exists at least a fixed point $\alpha \in [a,b]$ for function ϕ (the example has two), so **not uniqueness of fixed point**
  2. If in addition we assume that there exists an integer $L < 1$ s.t.

     $$|\phi(x_1) - \phi(x_2)| \leq L|x_1 - x_2| \; \forall \, x_1, x_2 \in [a,b]$$

     (Lipschitz continuity, weaker demand w.r.t. derivability, weaker than C^1) then $\exists ! \; \alpha \in [a,b]$ for ϕ and

     $$\{x^{(k)}\} \to \alpha \; \forall \, x^{(0)} \in \mathbb{R}$$

     Collection of approximations, with this assumption we get **uniqueness of fixed point and convergence of method independently from initial guess**.
     About the **rate of convergence of fixed point scheme with Lipschitz**, consider the error associated with $k+1$:

     $$|x^{(k+1)} - \alpha| = |\phi(x^{(k)}) - \phi(\alpha)| \leq L|x^{(k)} - \alpha|$$

     $$\frac{|x^{(k+1)} - \alpha|}{|x^{(k)} - \alpha|} \leq L < 1$$

     Which means that the fixed method is convergent with order 1 (power below is 1, but actually $p \geq 1$, so **order at least 1**)

  But this is not practical

- **Local result or Ostrowski's theorem**: let α be a fixed point for ϕ in $[a,b]$ (so we are already assuming the existence and uniqueness of α), with $\phi \in C^1(I_\alpha)$ and I_α neighborhood of α (different from before, we here stronger assumptions than global which only required C^0 and Lipschitz continuity, but C^1 locally).
  Under these hypotheses, if $|\phi'(\alpha)| < 1$ then $\exists \, \delta > 0$ s.t. $\forall \, x^{(0)}$ with $|x^{(0)} - \alpha| < \delta$:

  $$\{x^{(k)}\} \to \alpha \text{ and } \lim_{k \to \infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'(\alpha)$$

To summarize if:

$$\begin{cases} |\phi'(\alpha)| < 1 \text{ convergence} \\ |\phi'(\alpha)| > 1 \text{ divergence} \\ |\phi'(\alpha)| = 1 \text{ we cannot say anything} \end{cases}$$

Also for for smaller values, corresponding ϕs will be more efficient

2.5.5 Examples for the Local Result

1. $\phi(x) = \cos(x)$ and $\phi'(x) = -\sin(x)$

$$|\phi'(\alpha)| = |\sin(\alpha)| < 1 \qquad \alpha \neq 0$$

Converges

2. $\phi(x) = x^2 - 1$, we want to know if fixed point method is convergent or not: first we derive the fixed point of ϕ:

$$x = \phi(x) \to x^2 - x - 1 = 0$$

$$\alpha_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

So two fixed point, now compute derivative

$$\phi'(x) = 2x \to |\phi'(\alpha)| = \left|1 \pm \sqrt{5}\right|$$

In neither cases the module is less than 1, so divergent fixed point method

3. $\log(x) = \gamma$ with $\gamma \in \mathbb{R}$, we are demanded to approximate this function with fixed point:

$$f(x) = 0 \to \underbrace{\log(x) - \gamma}_{f(x)} = 0$$

We can use whatever method/fixed point function we like:

(a) With Newton

$$\phi_1(x) = \phi_N(x) = x - \frac{\log(x) - \gamma}{\frac{1}{x}} = x(1 - \log(x) + \gamma)$$

(b) Simply use $\phi(x) = f(x) + x$

$$\phi_2(x) = \log(x) - \gamma + x$$

(c)

$$x\log(x) - \gamma x = 0 \to \phi_3(x) = \frac{x\log(x)}{\gamma}$$

For $\gamma = -2$, we can verify that ϕ_1 and ϕ_3 are ok, while for the ϕ_2 it does not work

2.5.6 Fixed point method with any order of convergence

Proposition: let us assume that the Ostrowski's theorem hypothesis are verified (**so we are in local setting**). If $\phi \in C^P(I_\alpha)$ and $\phi^{(i)}(\alpha) = 0$ $i = 1, \cdots, p-1$ (derivatives till $p-1$) and $\phi^{(p)}(\alpha) \neq 0$ then $\exists\, \delta > 0$ s.t. $\forall\, x^{(0)}$ with $\left|x^{(0)} - \alpha\right| < \delta$

$$\{x^{(k)}\} \to \alpha \text{ and } \lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{\left[x^{(k)} - \alpha\right]^p} = \underbrace{\frac{\phi^{(p)}(\alpha)}{p!}}_{C}$$

We can guarantee the convergence of our fixed point method and compute the convergence order.

2.5.7 Stopping criteria

Reminder

$$\left|e^{(kmin)}\right| = \left|x^{(kmin)} - \alpha\right| \leq CS = C\left|x^{(kmin+1)} - x^{(kmin)}\right| \leq TOL\ (e.g. = 10^{-9})$$

Reminder: $\exists\, \beta_k$ between α and $x^{(k)}$ (min value theorem, appliable since ϕ is C^1)

$$\alpha - x^{(k+1)} = \phi(\alpha) - \phi(x^{(k)}) = \phi'(\beta_k)(\alpha - x^{(k)})$$

So, adding and subtracting $x^{(k+1)}$:

$$\alpha - x^{(k)} = \alpha - x^{(k+1)} + \underbrace{x^{(k+1)} - x^{(k)}}_{\delta^{(k)}} = \phi'(\beta_k)(\alpha - x^{(k)}) + \delta^{(k)}$$

$$\alpha - x^{(k)} = \frac{1}{1-\phi'(\beta_k)}(x^{(k+1)} - x^{(k)})$$

For k sufficiently large we can identify $x^{(k)}$ with α and β_k with α. So the error associated with the k iteration is:

$$\alpha - x^{(k)} \cong \underbrace{\frac{1}{1-\phi'(\alpha)}}_{C}(x^{(k+1)} - x^{(k)})$$

For C as much as possible close to 1, it means that $\phi'(\alpha)$ is very small, almost zero.

$$\phi'(\alpha) = 0 \rightarrow \text{convergence order of at least 2! Quadratic convergence}$$

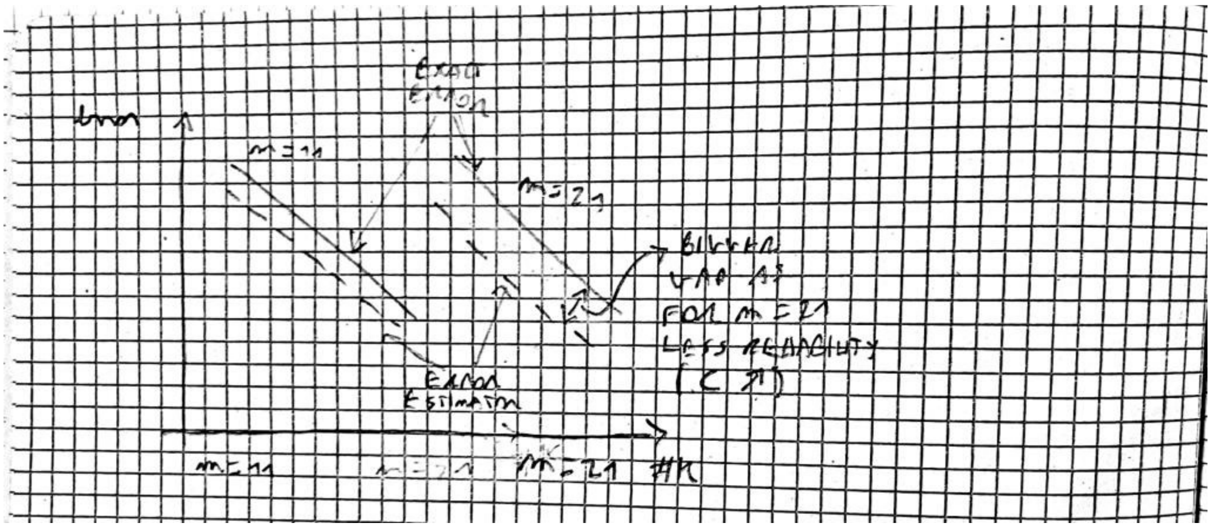
An **example**:

$$f(x) = (x-1)^{(n-1)}\log(x)$$

With one zero $\alpha = 1$ with multeplicity m. We can prove that:

$$\phi'(\alpha) = 1 - \frac{1}{m}$$

Larger is m, closer that quantity is to 1, which means we are losing in terms of reliability (C is growing to infinity).



A stopping criterion for fixed point: m not too high.

2.5.8 Consistency

$$x^{(k+1)} = \phi(x^{(k)})$$

If α is a fixed point of ϕ, the method is consistent

3 Systems of Linear Equations: Direct Methods

We are talking about

$$Ax = b \Leftrightarrow \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_n + \cdots + a_{nn}x_n = b_n \end{cases}$$

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

3.1 Cramer Method

Reminder:

$$x_i = \frac{\det(A_i)}{\det(A)} \qquad i = 1, \cdots, n$$

Problem: finding the determinants computationally intensive, reminder of the Laplace rule:

$$\det(A) = \begin{cases} a_n & \text{if } n = 1 \\ \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \Delta_{ij} & \forall\, i \qquad \Delta_{ij} = \det(A_{ij}) \end{cases}$$

Where A_{ij} is the submatrix obtained by removing the ith row and jth column.
On average we peform a number of operations equal to $3(n+1)!$, absolutely unfeasible, we must move to numerical counterparts/approximations.

3.2 Numerical Approximations: Direct Methods

We will talk about:

1) Direct methods (fixed number of steps to get the solution, no notion of convergence)
2) Iterative methods (non-ending number of steps, we will have to stop at a certain point, convergence)

There is no better alternative, both depend on the kind of the problem

At the basis of the direct methods we will use the factorization of the matrix: **LU factorization**.

3.2.1 LU factorization of a matrix

Let A a matrix, it can be expressed as the product of two matrices $A = LU$, where L is a lower triangular matrix (elements different from zero on the main diagonal and on elements below) and U is an upper triangular matrix.

Consider

$$Ax = b \qquad A \text{ nonsingular}$$

$$L\underbrace{Ux}_{y} = b$$

The solution of the system then changes to

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Solve the upper system first, then the lower one

As the matrices are triangular (a lot of zeros, they are sparse), the systems are easier to solve! A matrix is defined as **sparse** if the number of entries different from zeros is a $O(n)$ istead of $O(n^2)$

A remark, if A is nonsingular, then it follows that L and U are nonsingular as

$$\det(A) = \det(LU) = \det(L)\det(U)$$

Also since the **determinant of a triangular matrix is the product of the entries in the diagonal**, all elements on the diagonal are non-zero.

3.2.2 Forward substitution method

Assume L and U are given, we start from:

$$Ly = b$$

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$l_{11}y_1 = b_1 \rightarrow y_1 = \frac{b_1}{l_{11}} \qquad l_{11} \neq 0$

$l_{21}y_1 + l_{22}y_2 = b_2 \rightarrow y_2 = \frac{1}{l_{22}}\left[b_2 - l_{21}y_1\right]$

$l_{31}y_1 + l_{32}y_2 + l_{33}y_3 = b_3 \rightarrow y_3 = \frac{1}{l_{33}}\left(b_3 - l_{31}y_1 - l_{32}y_2\right)$

So for a generic lower triangular matrix L:

$$y_1 = \frac{b_1}{l_{11}}$$

$$y_i = \frac{1}{l_{ii}}\left[b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right] \qquad i = 2, \cdots, n$$

With a number of operations for each i:

- 1 division for l_{ii}
- In the squares $i-1$ subtractions
- Each subtractions is a product, we have $i-1$ multiplications

So the total number of operations is:

$$1 + \sum_{i=2}^{n}(1+2i-2) = \sum_{i=1}^{n}(1+2i-1) = \sum_{i=1}^{n} 1 + 2\sum_{i=1}^{n}(i-1) = n + 2\frac{n(n-1)}{2} = \mathbf{n^2}$$

3.2.3 Backward substitution method

Consider now the upper triangular system:

$$Ux = b$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$u_{33}x_3 = y_3 \rightarrow x_3 = \frac{y_3}{u_{33}}$

$\cdots \rightarrow x_2 = \frac{1}{u_{22}}\left[y_2 - u_{23}x_3\right]$

$\ldots$

So for a generic lower upper matrix U:

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

$$x_i = \frac{1}{u_{ii}}\left[y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right] \qquad i = n-1, \cdots, 1$$

So the total number of operations is n^2

3.2.4 Direct inspection

Suppose we know the matrix, we want to find the LU factorization:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \underbrace{\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & 0 \end{bmatrix}}_{U}$$

In this case, we can write:

$$\begin{cases} l_{11}u_{11} = a_{11} \\ l_{11}u_{12} = a_{12} \\ l_{21}u_{11} = a_{21} \\ l_{21}u_{12} + l_{22}u_{22} = a_{22} \end{cases}$$

We can see that we have 6 unknowns and 4 equations. By convetion we can assign "1" to the diagonal of the matrix L

$$\underbrace{\begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & 0 \end{bmatrix}}_{U}$$

So our system becomes:

$$\begin{cases} u_{11} = a_{11} \\ u_{12} = a_{12} \\ l_{21}u_{11} = a_{21} \\ l_{21}u_{12} + u_{22} = a_{22} \end{cases}$$

From which we easily find the unknowns.

If we consider a generic matrix N:

$$\underbrace{\begin{bmatrix} \ddots & & \ddots \\ & A & \\ \ddots & & \ddots \end{bmatrix}}_{n^2 \text{ equations}} = \underbrace{\begin{bmatrix} \ddots & & \\ & L & \\ & & \ddots \end{bmatrix}}_{\text{With } \frac{n(n+1)}{2} \text{ unknowns}} \underbrace{\begin{bmatrix} \ddots & & \\ & U & \\ & & \ddots \end{bmatrix}}_{\text{With } \frac{n(n+1)}{2} \text{ unknowns}}$$

Globally with n^2+n unknowns. Just like before, we assign "1" to the diagonal of L so #equations=#unknowns

3.2.5 Gaussian elimination method

GEM, less computationally intensive than direct inspection. Consider the matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

We assign for each entry a superindex to underline that it's the original matrix:

$$A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix}$$

1. First step is to move to the matrix $A^{(2)}$. Consider the coefficients:

   $l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$ pivot $a_{11}^{(1)} \neq 0$

   $l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}}$ pivot $a_{11}^{(1)} \neq 0$

   $$A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix}$$

   Where

   - $R2_{new} = R2_{old} - l_{21}R1_{old}$ and

     $a_{21}^{(2)} = a_{21}^{(1)} - l_{21}a_{11}^{(1)} = a_{21}^{(1)} - \frac{a_{21}^{(1)}}{a_{11}^{(1)}}a_{11}^{(1)} = 0$

     $a_{22}^{(2)} = a_{22}^{(1)} - l_{21}a_{12}^{(1)}$

     $a_{23}^{(2)} = a_{23}^{(1)} - l_{21}a_{13}^{(1)}$

   - $R3_{new} = R3_{old} - l_{31}R1_{old}$ and

     $a_{31}^{(2)} = a_{31}^{(1)} - l_{31}a_{11}^{(1)} = a_{31}^{(1)} - \frac{a_{31}^{(1)}}{a_{11}^{(1)}}a_{11}^{(1)} = 0$

     $a_{32}^{(2)} = a_{32}^{(1)} - l_{31}a_{12}^{(1)}$

     $a_{33}^{(2)} = a_{33}^{(1)} - l_{31}a_{13}^{(1)}$

2. Similarly find $A^{(3)}$. Consider the coefficient:

   $l_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}$ pivot $a_{22}^{(2)} \neq 0$

$$A^{(3)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} = U$$

   Where

   - $R3_{new} = R3_{old} - l_{32}R2_{old}$ and

     $a_{32}^{(3)} = a_{32}^{(2)} - l_{32}a_{22}^{(2)} = a_{32}^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}}a_{22}^{(2)} = 0$

     $a_{33}^{(3)} = a_{33}^{(2)} - l_{32}a_{23}^{(2)}$

Hence we found U. What about L? It is:

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

Let's see if $LU = A$, for example:

$$A = A^{(1)} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & -1 \\ -1 & 1 & 5 \end{bmatrix}$$

1. Step 1:

$$A^{(2)} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & -4 & -3 \\ 0 & 3 & 6 \end{bmatrix}$$

   With

   $l_{21} = 2$
   $l_{31} = -1$

2. Step 2:

$$A^{(3)} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & -4 & -3 \\ 0 & 0 & \frac{15}{4} \end{bmatrix} = U$$

   With

   $l_{32} = -\frac{3}{4}$

And L:

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

We check that:

$$L \cdot U = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -\frac{3}{4} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & -4 & -3 \\ 0 & 0 & \frac{15}{4} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & -1 \\ -1 & 1 & 5 \end{bmatrix} = A$$

What about the cost? It is CUBIC: $\frac{2}{3}n^3$.

When we want to solve $Ax = b$ we have multiple ways:

1. Using GEM and LU factorization
   - GEM $\frac{2}{3}n^3$.
   - Resolution of
     $$\begin{cases} Ly = b \\ Ux = y \end{cases}$$
     Paying $2n^2$

   Paying in total $\mathbf{\frac{2}{3}n^3 + 2n^2 \sim O(n^3)}$
2. GEM on $[A|b]$ paying $(> \mathbf{\frac{2}{3}n^3}) + \mathbf{2n^2 \sim O(n^3)}$
3. Using $x = A^{-1}b$ paying an higher cost than the previous two methods. The inverse can be found applying GEM till:
   $$[A|I_n] \Rightarrow \left[I_n|A^{-1}\right]$$

Consider methods 1 and 2, in which case one is the less costly than the other? When we have to solve multiple systems of the kind $Ax = c_{i\to q}$ with right side different every time: in this case the first method is better as we pay the cost of LU factorization only once (so we pay $\frac{2}{3}n^3 + q(2n^2)$) while with the second one we pay the whole cost every time (paying $q\left(\frac{2}{3}n^3 + 2n^2\right)$).
This is the case of method 3, we are solving $Ax = c_{i\to q}$ multiple times and in total we pay

$$\mathbf{\frac{2}{3}n^3 + n(2n^2) = \frac{8}{3}n^3}$$

3.2.6 Necessary and sufficient criteria for GEM

$$A = A^{(1)} \begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \to A^{(2)} = \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & -4 \\ 0 & 3 & -5 \end{bmatrix}$$

With $[l_{21}\ l_{31}\ l_{32}] = [2\ 3\ \frac{3}{0}]$. The last pivot error! Some conditions:

- **Necessary and sufficient condition**: let $A \in \mathbb{R}^{n\times n}$, then LU factorization $\exists!$ **if and only if** the **principal submatrices** A_i of A for $i = 1, \cdots, n-1$ are nonsingular. The principal submatrices are:

  $$A = \begin{bmatrix} A_1 & \vdots & \vdots & & \vdots & \\ \cdots & A_2 & \vdots & & \vdots & \\ \cdots & \cdots & A_3 & & \vdots & \\ \cdots & \cdots & \cdots & \ddots & \vdots & \\ \cdots & \cdots & \cdots & \cdots & A_{n-1} & \\ \cdots & & & & & \end{bmatrix}$$

  As i goes to $n-1$, the matrix $A_n == A$ can be singular, a LU exists anyway (but solution might not). This condition guarantess that a **LU factorization exists and it is unique**.

  Some examples

  – Singular matrix but LU factorization can be found

  $$A_1 = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

- We lose existence
$$A_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$
As $l_{21}u_{11} = 1$ is impossible to satisfy as $u_{11} = 0$
- We lose uniqueness
$$A_3 = \begin{bmatrix} 0 & 1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$
As $l_{21}u_{11} = 0$, always satisfyable as $u_{11} = 0$

- **Sufficient conditions for LU factorization existence and uniqueness**, 3 conditions that correspond to 3 families of lucky matrices

1) A belongs to the family of the strictly diagonally dominant by rows matrices, which means that the element of the diagonal is dominant w.r.t. the other elements in the same row: the absolute value of diagonal element is strictly greater than sum of absolute value of other elements:
$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad i = 1, \cdots, n$$
An example that is not strictly diagonal by rows:
$$\begin{bmatrix} -4 & 0 & 3 \\ 1 & 2 & -4 \\ 7 & -1 & 10 \end{bmatrix}$$
The second row does not satisfy the condition, something that satifies the condition would have second row as $[1 \ -4 \ 2]$

2) In some sense the dual family of the previous case: A belongs to the family of the strictly diagonally dominant by columns matrices:
$$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}| \qquad j = 1, \cdots, n$$
Strictly dominant by columns does not mean it is also strictly dominant by rows, and viceversa

3) A belongs to the family of symmetric positive definite (spd) matrices.
   - A matrix is symmetric if:
$$a_{ij} = a_{ji} \qquad i, j = 1, \cdots, n$$
$$A = A^T$$
   - A matrix is positive definite if:

   $\forall\, x \neq 0 \in \mathbb{R}$ for each vector, we compute the scalar and verify that:
$$x^T A x > 0$$

   But we perform an infinite number of checks, we consider the pair eigenvalues and eigenvectors:
$$A \in \mathbb{R}^{n\times n}$$
$$(\lambda, v)$$
$$Av = \lambda v$$
   - A matrix is symmetric if all eigenvalues are real numbers
   - A matrix is positive definite if all eigenvalues are positve $\lambda > 0$

3.2.7 LU strategies for particular matrices

- If matrix A is spd, we consider the $A = LU$, how does the two factors inherit the symmetric aspect of the original matrix? As
$$A = LU = A^T = (LU)^T = U^T L^T$$
One is the transpose of the other, so we compute only one factor. In matlab Cholesky, $R = chol(A)$, and the computational cost from $\frac{2n^3}{3}$ will be reduced to $\frac{n^3}{3}$.
$$A = R^T R$$
Also **the diagonal entries of matrix** R $r_{ii} > 0$ **will all be positve**.

The **Cholesky decomposition**:
$$H = \begin{cases} h_{11} = \sqrt{a_{11}} \\ h_{ij} = \frac{1}{h_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} h_{ik}h_{jk}\right) \qquad j = 1, \cdots, i-1 \\ h_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2} \end{cases}$$
Where
$$H^T = R$$
And our system becomes
$$\begin{cases} Ax = b \\ A = R^T R = HH^T \end{cases} \to R^T Rx = b \to \begin{cases} R^T y = b \\ Rx = y \end{cases} = \begin{cases} Hy = b \\ H^T x = y \end{cases}$$

- Consider a 3-diagonal matrix, which has diagonal, first upper and first lower diagonals the only non-null overall, while all the others zero.

$$A = \begin{bmatrix} \ddots & \ddots & 0 & 0 & 0 \\ \ddots & \ddots & \ddots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \ddots & \ddots \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \ddots & 1 & 0 & 0 & 0 \\ 0 & \ddots & 1 & 0 & 0 \\ 0 & 0 & \ddots & 1 & 0 \\ 0 & 0 & 0 & \ddots & 1 \end{bmatrix}}_{L \text{ lower bi-diagonal, with main diagonal all 1's}} \underbrace{\begin{bmatrix} \ddots & \ddots & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \ddots & \ddots \\ 0 & 0 & 0 & 0 & \ddots \end{bmatrix}}_{U \text{ upper bi-diagonal}}$$

To find the LU factorization, consider
$$A = \begin{bmatrix} a_1 & c_1 & 0 \\ e_2 & a_2 & c_2 \\ 0 & e_3 & a_3 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ \beta_2 & 1 & 0 \\ 0 & \beta_3 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \alpha_1 & \gamma_1 & 0 \\ 0 & \alpha_2 & \gamma_2 \\ 0 & 0 & \alpha_3 \end{bmatrix}}_{U}$$

It can be easily verified that

$\gamma_1 = c_1$ $\quad R1 * C2$ first row of L by second column of U
$\gamma_2 = c_2$ $\quad R2 * C3$

So we have 5 unknowns in β_i and α_i:

$R1 * C1 \qquad \alpha_1 = a_1$
$R2 * C1 \qquad \beta_2 \alpha_1 = e_2 \rightarrow \beta_2 = \frac{e_2}{\alpha_1}$
$R2 * C2 \qquad \beta_2 c_1 + \alpha_2 = a_2 \rightarrow \alpha_2 = a_2 - \beta_2 c_1$
$R3 * C2 \qquad \beta_3 \alpha_2 = e_3 \rightarrow \beta_3 = \frac{e_3}{\alpha_2}$
$R3 * C3 \qquad \beta_3 c_2 + \alpha_3 = a_3 \rightarrow \alpha_3 = a_3 - \beta_3 c_2$

So with a general $n \times n$ matrix:

$$\begin{cases} \alpha_1 = a_1 \\ \begin{cases} \beta_i = \frac{e_i}{\alpha_{i-1}} \\ \alpha_i = a_i - \beta_i c_{i-1} \\ i = 2, \cdots, n \end{cases} \end{cases}$$

For every β_i we pay 1 division, for every α_1 we pay 1 multiplication and 1 difference, so in total we pay:

$$3(n-1)$$

Our problem is

$$Ax = b \rightarrow LUx = b \rightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

– The first equation:

$$\begin{bmatrix} 1 & 0 & 0 \\ \beta_2 & 1 & 0 \\ .0 & \beta_3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

From which:

$y_1 = b_1$
$y_2 = b_2 - \beta_2 y_1$
$y_3 = b_3 - \beta_3 y_2$

Differently from forward substitution method:

* We don't have to subtract by a sum of elements
* There is no division term

And for a general $n \times n$ matrix:

$$\begin{aligned} & y_1 = b_1 \\ & y_i = b_i - \beta_i y_{i-1} \qquad i = 2, \cdots, n \end{aligned}$$

With cost of $2(n-1)$

– The second equation:

$$\begin{bmatrix} \alpha_1 & c_1 & 0 \\ 0 & \alpha_2 & c_2 \\ 0 & 0 & \alpha_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

From which:

$x_3 = \frac{y_3}{\alpha_3}$
$x_2 = \frac{1}{\alpha_2}[y_2 - c_2 x_3]$
$x_1 = \frac{1}{\alpha_1}[y_1 - c_1 x_2]$

Differently from backward substitution method we don't have to subtract a summation term. For a general matrix:

$$x_n = \frac{y_n}{\alpha_n}$$
$$x_i = \frac{1}{\alpha_i}[y_i - c_i x_{i+1}]$$

With cost of $3(n-1)+1$

So in total we pay:

$$\underbrace{3(n-1)}_{\text{factorization}} + \underbrace{2(n-1)}_{Ly=b} + \underbrace{3(n-1)+1}_{Ux=y} = 8n-7$$

All this process is knows as **Thomas algorithm**

3.2.8 Rows swapping: pivoting

Consider this matrix:

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 3 \\ 0 & 0 & -4 \\ 0 & 3 & -5 \end{bmatrix}$$

We have $a_{22}^{(2)} = 0$, a pivot is nullable. Also if we verify the necessary and sufficient condition we see that the second principal matrix is singular. A solution is to swap the rows:

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 3 & 6 & 4 \\ 2 & 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 3 \\ 0 & 3 & -5 \\ 0 & 0 & -4 \end{bmatrix}$$

The necessary and sufficient condition is satisfied now. Remember, **the matrix must be nonsingular**.

We perform the permutation when we meet the problem, otherwise we perform GEM all over again. We exchange rows i with i+1 if:

$$a_{ii}^{(i)} = 0 \ \&\& \ a_{i+1,i}^{(i)} \neq 0$$

To do such swap we need to define a **permutation matix** P, an orthogonal matrix ($PP^T = P^TP = I$), obtained from identity matrix by swapping some rows. Premultipying it by our original metrix we perform the exchange, for example:

$$PA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

To exchange two rows, we just need to premultiply the original matrix by a permutation matrix. If we postmultiply AP we swap columns instead.

This process is called **pivoting by rows**. So during GEM we keep track of swappings on the matrix P, so at the end we obtain LU factorization not for A but for PA

$$PA = LU$$

And the solution

$$Ax = b \rightarrow PAx = Pb \rightarrow LUx = Pb$$

$$\begin{cases} Ly = Pb \\ Ux = y \end{cases}$$

Only on the top equation we have effect of the permutation matrix.

But what if the pivot is not null but very small?

$$a_{ii}^{(i)} = 2.5 \cdot 10^{-4}$$

At a certain point we will divide by it, amplifying the floating point error. Consider:

$$A = \begin{bmatrix} 1 & 1+0.5\cdot 10^{-6} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{bmatrix}$$

The necessary and sufficient condition holds, the determinant of the second principal matrix is not zero but very colose to it, we can apply GEM. But the LU factorization might not be accurate. The idea is to keep the pivot as large as possible in the GEM.

In matlab we will use

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

3.3 Accuracy

This concerns iterative methods as well. Till now we saw that

$$LU + pivoting \rightarrow \text{accurate } LU \text{ factorization } A - LU$$

With A nonsingular. If this factorization is accurate, is the solution to the system $Ax = b$ accurate as well? **No, an accurate LU factorization does not necessarily ensure to have an accurate solution**.

Consider the Hilbert matrix:

$$a_{ij} = \frac{1}{i+j-1}$$

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & \cdots \\ 1/2 & 1/3 & 1/4 & \cdots & \\ 1/3 & 1/4 & \cdots & & \\ \cdots & & & & \end{bmatrix}$$

It is a symmetric matrix and positive definite, so spd. Let A_n the Hilbert matrix of order n, consider the family of systems:

$$A_n x_n = b_n \qquad b_n, x_n \in \mathbb{R}^n,\ A_n \in \mathbb{R}^{n\times n}$$

The exact solution is

$$b_n\ x_n = [1, 1, \cdots, 1]^T$$

Consider the error:

$$R_n = P_n A_n - L_n U_n \qquad (A)$$

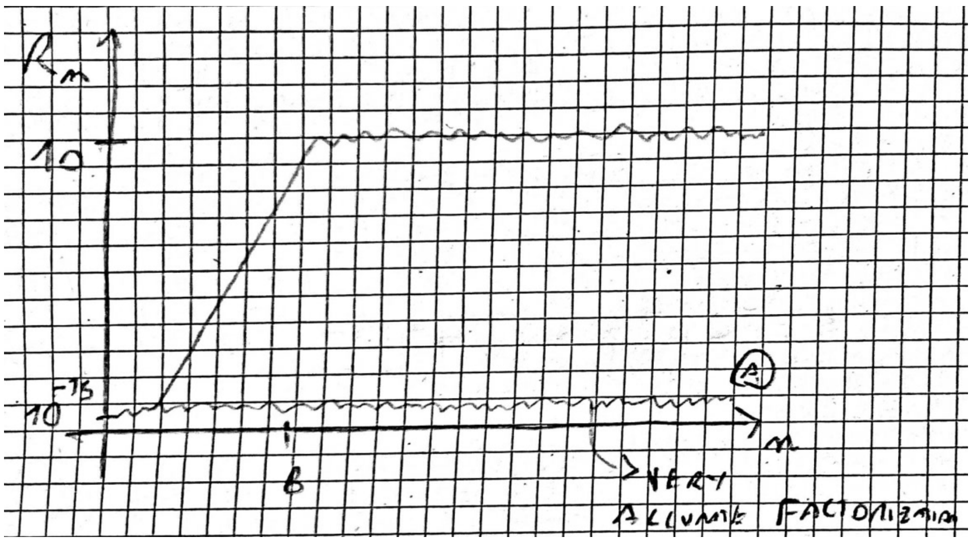
With each n we associate:

$$\max_{i,j} |r_{n_{ij}}|$$

Consider the approximation $\tilde{x}_n$, the relative error

$$E_n = \frac{||x_n - \tilde{x}_n||}{||x_n||} \qquad (B)$$

With (B) is an accurate LU factorization, but the result is not accurate



The reason is the problem, the Hilbert matrix is **ill positioned problem**. The method does not solve $Ax = b$, but retrieves the solution of the **perturbated system**:

$$(A + \underbrace{\delta A}_{\mathbb{R}^{n\times n}})(x + \delta x) = b + \underbrace{\delta b}_{\mathbb{R}^n}$$

δb and δA are perturbation on the data, matlab is computing the exact solution, but due to point floating point we will have a solution perturbation δx. We would like that for small perturbation on the data we get small perturbation on the problem (**well positioned problems**). Let the perturbated solution be

$$\tilde{x} = x + \delta x$$

We want to relate the two perturbations:

$$\frac{||\delta A||}{||A||} \qquad \frac{||\delta b||}{||b||}$$

We must firstly introduce some lemmas.

3.3.1 Lemmas

1) Let $B \in \mathbb{R}^{n\times n}$ be a spd matrix. Then it holds

$$\lambda_{\min}(B) \le \frac{x^T Bx}{x^T x} \le \lambda_{\max}(B) \qquad \forall\, x \neq 0 \in \mathbb{R}^n$$

Where λ's are the eigenvalues.

2) Let A be a nonsingular matrix. Then $A^T A$ is spd

3.3.2 Perturbations relation

Starting from $\delta A = 0$, we want to relate the two perturbations:

$$\frac{||\delta A||}{||A||} \qquad \frac{||\delta b||}{||b||}$$

- First step, we subtract term by term the exact system $Ax = b$ from the perturbated one $(A+0)(x+\delta x) = b + \delta b$:
$$A\delta x = \delta b$$

Consider the L2-norm (euclidean norm $||w||^2 = w^T w$):

$$||A\delta x||^2 = ||\delta b||^2 \to (A\delta x)^T (A\delta x) = ||\delta b||^2 \to \delta x^T A^T A \delta x = ||\delta b||^2$$

From the lemma 2 $A^T A$ is spd, so we can apply lemma 1 with $B = A^T A$

$$\lambda_{\min}(A^T A)\delta x^T \delta x \leq \delta x^T A^T A \delta x = ||\delta b||^2 \leq \lambda_{\max}(A^T A)\delta x^T \delta x$$

With $\delta x^T \delta x = ||\delta x||^2$. Consider only the left part of the inequality:

$$\lambda_{\min}(A^T A)||\delta x||^2 \leq ||\delta b||^2$$

$$||\delta x|| \leq \frac{||\delta b||}{\sqrt{\lambda_{\min}(A^T A)}}$$

- As the next step consider

$$||Ax||^2 = ||b||^2$$

  Similar reasoning:

$$\lambda_{\min}(A^T A)x^T x \leq x^T A^A x = (Ax)^T Ax = ||b||^2 \leq \lambda_{\max}(A^T A)x^T x$$

  With $x^T x = ||x||^2$. Consider only the right part of the inequality:

$$||b|| \leq \sqrt{\lambda_{\max}(A^T A)}||x||$$

$$\frac{1}{||x||} \leq \frac{\sqrt{\lambda_{\max}(A^T A)}}{||b||}$$

- We put everything together. As both inequalities are all positive quantities, we can combine them considering:

$$\begin{cases} a \leq b \\ c \leq d \end{cases} \to ab \leq cd$$

$$\frac{||\delta x||}{||x||} \leq \underbrace{\sqrt{\frac{\lambda_{max}(A^T A)}{\lambda_{min}(A^T A)}}}_{K(A) \geq 1 \text{ condition number}} \frac{||\delta b|}{||b||}$$

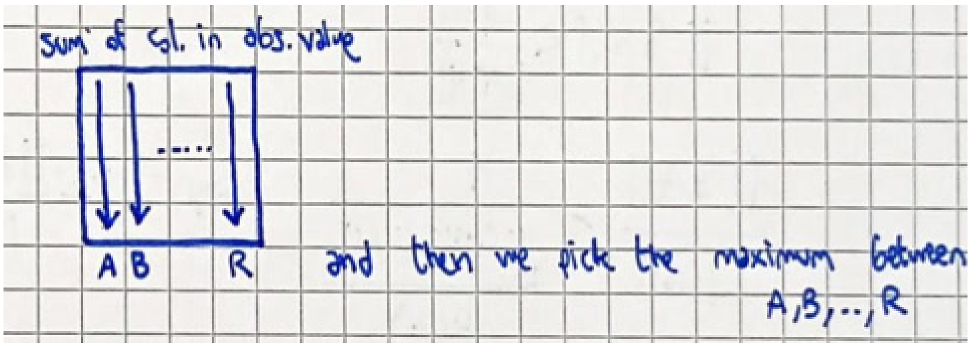
The condition number can only amplify, a small perturbation on the data will be small as well if the condition number is almost 1 (well conditioned problem), but if the condition number is very large this relationship does not hold (ill conditioned problem, just like for Hilbert)

3.3.3 Condition number

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

A norm of a matrix has 3 possible definitions

- **Norm one**, maximum of the sum by columns


sum of col. in abs. value
A B R
and then we pick the maximum between A,B,..,R

$$||A||_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

$$K(A) = ||A||_1 \cdot ||A^{-1}||_1$$

- **Infinity norm**, first compress horizontally, then compress vertically, like dual of before

  $$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

  $$K(A) = ||A||_\infty \cdot ||A^{-1}||_\infty$$

  If the matrix is symmetric the infinity norm coincides with norm one

- **2-norm or Spectral norm**, spectrum of a matrix is the collection of the eigenvalues

  $$||A||_2 = \sqrt{\lambda_{\max}(A^T A)}$$

  From before, we found that:

  $$K(A) = \sqrt{\frac{\lambda_{max}(A^T A)}{\lambda_{min}(A^T A)}} = ||A||_2 ||A^{-1}||_2$$

  So before we found a particular condition number:

  $$K_2(A) = ||A||_2 ||A^{-1}||_2$$

  - A special case, if A is symmetric ($A = A^T$), so $A^T A = A^2$

    $$\lambda_{\max}(A^2) = [\lambda_{\max}(A)]^2$$

    And

    $$||A||_2 = \lambda_{\max}(A)$$

    $$K_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

In matlab $cond(A)$ for 2-norm, $condest(A_{sparse})$ for 1-norm

Now we can rewrite:

$$\delta b = A \underbrace{(x + \delta x)}_{\text{perturbation } \tilde{x}} - b = A\tilde{x} - b = -\tilde{r}$$

We get the residual

3.3.4 Perturbation on the matrix A

What if $\delta A \neq 0$? If $||\delta A|| \cdot ||A^{-1}|| < 1$, we can:

$$\frac{||\delta x||}{||x||} \leq \frac{K(A)}{1 - K(A)\frac{||\delta A||}{||A||}} \left[\frac{||\delta A||}{||A||} + \frac{||\delta b||}{||b||} \right]$$

From the hypothesis

$$||\delta A|| \cdot ||A^{-1}|| < 1 \rightarrow ||\delta A|| < \frac{1}{||A^{-1}||} \rightarrow \frac{||\delta A||}{||A||} < \frac{1}{||A|| \cdot ||A^{-1}||} \rightarrow K(A)\frac{||\delta A||}{||A||} < 1$$

Which means

$$1 - K(A)\frac{||\delta A||}{||A||} > 0$$

In conclusion, before solving a problem check the condition number to see if it is ill conditioned or not. If condition number too huge problem

4 Systems of Linear Equations: Iterative Methods

4.1 Iterative methods

We start from an initial guess $x^{(0)} \in \mathbb{R}^n$, that enters a black box that gives a $x^{(1)}$, and so on

$$\left\{x^{(k)}\right\} \qquad x^{(k)} \simeq x \qquad x^{(k)} \in \mathbb{R}^n$$

4.1.1 Convergence

we will talk about convergence again

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

Or alternatively express convergence in terms of error

$$\begin{cases} e^{(k)} = x - x^{(k)} \\ \lim_{k\to\infty} e^{(k)} = 0 \end{cases}$$

4.1.2 Consistency

In iterative methods we will always talk about **convergence, consistency and stability**.

Consistency:

$$x = Bx + g$$

With B matrix that depends on A and g vector that depends on A and B

$$B = B(A) \qquad g = g(A,b)$$

We follow the recursive definition to create a new iteration:

$$x^{(k+1)} = Bx^{(k)} + g \qquad B \in \mathbb{R}^{n\times n} \qquad g \in \mathbb{R}^n$$

Consistency means that the equality has to hold w.r.t. the exact solution. Notice that g depends on A as:

$x = Bx + g$
$(x - Bx) = g$
$(I - B)x = g$
$(I - B)A^{-1}b = g$

4.1.3 Convergence analysis

Subtract term by term

$$x^{(k+1)} = Bx^{(k)} + g \qquad - \qquad x = Bx + g$$

$$\underbrace{x - x^{(k)}}_{e^{(k+1)}} = \underbrace{B(x - x^{(k)})}_{e^{(k)}}$$

$$e^{(k+1)} = Be^{(k)}$$

Compatibility of vector norms says that:

$$||e^{(k+1)}|| = ||Be^{(k)}|| \leq ||B||_2 \; ||e^{(k)}||$$

Use this inequality for convergence analysis

$$||e^{(k)}|| \leq ||B||_2 \; ||e^{(k-1)}|| \leq ||B||_2^2 \; ||e^{(k-2)}|| \leq \cdots \leq ||B||_2^k \; ||e^{(0)}||$$

Where the following quantity is called **spectral radius**

$$||B||_2^k = [\rho(B)]^k$$

Spectral radius is the maximum of the eigenvalue of the matrix taken in absolute value (in matlab $max(abs(eig(A)))$). A reminder, the 2-norm:

$$||B||_2 = \sqrt{\lambda_{max}(B^T B)}$$

And if B is spd (eigenvalues are real positive)

$$||B||_2 = \lambda_{max}(B)$$

Which means we are comparing λ_{max} with itself, as in this case it is both spectral radius and 2-norm of the matrix! (**only if spd**)

To summarize:

$$||e^{(k)}|| \leq [\rho(B)]^k \, ||e^{(0)}||$$

And for convergence, the spectral radius must be less to 1

$$\begin{cases} \rho(B) < 1 \\ \lim_{k\to\infty} [\rho(B)]^k \, ||e^{(0)}|| = 0 \end{cases}$$

Attention, we **assumed that the method is consistent, make sure that the method is consistent**

This inequality represents a **necessary and sufficient condition for convergence**: let us consider the iterative scheme given by $x^{(k+1)} = Bx^{(k)} + g$ and let us assume that such a scheme is **consistent**, then the scheme turns out to be convergent independently of the initial guess, $\forall\, x^{(0)} \in \mathbb{R}^n$ **if and only if** $\rho(B) < 1$

A scheme to be convergent, we must check consistency and spectral radius. If one of these does not hold, the method is not convergent.

Additionally, the smaller $\rho(B)$, the quicker the convergence.

We can also rewrite the inequality and get

$$\frac{||e^{(k)}||}{||e^{(0)}||} \leq [\rho(B)]^k \leq TOL$$

To find k=minimum number of iterations

4.2 Richardson schemes

We have to introduce

$$P \in \mathbb{R}^{n\times n} \qquad \text{nonsingular matrix (preconditioner)}$$

$$\alpha_k \neq 0 \in \mathbb{R}^n$$

We rewrite the original system ($Ax = b$) to

$$\alpha_k Ax = \alpha_k b$$

And rewrite matrix A with splitting

$$\alpha_k A = P - (P - \alpha_k A)$$

So:

$$P-(P-\alpha_k A)x=\alpha_k b$$

$$Px=\alpha_k b+(P-\alpha_k A)x$$

We arbitrary choose the left side associated to $k+1$ and the right side associated to k:

$$Px^{(k+1)}=\alpha_k b+(P-\alpha_k A)x^{(k)}$$

Notice, by repliacing $x^{(k)}$ with exact solution, we get consistency: **all the Richardson schemes are consistent by construction, so no need to check it, but we have to point it out!** We want to reach:

$$x^{(k+1)}=Bx^{(k)}+g$$

Multiplying by P^{-1} we will obtain:

$$x^{(k+1)}=\underbrace{\alpha_k P^{-1}b}_{g_{\alpha_k}}+\underbrace{\left(I-\alpha_k P^{-1}A\right)}_{B_{\alpha_k}}x^{(k)}$$

At this point we distinguish Richardson schemes to:

- Stationary, if $\alpha_k=\alpha$, which means it never changes at every iteration, a bad choice of α may hinders convergence, it must be in suitable range
- Dynamic if α_k changes at each iteration. The parameter can be tuned, it is better, but it's more computational intensive/demanding

α is the learning rate, accelerates convergence. Develop it:

$$x^{(k+1)}=\alpha_k P^{-1}(\underbrace{b-Ax^{(k)}}_{r^{(k)}\ k\text{th-residual}})+x^{(k)}$$

We can interpret it as "next iteration=previous+correction"

$$x^{(k+1)}=x^{(k)}+\alpha_k z^{(k)}$$

Where this value is the **preconditioned residual**

$$z^{(k)}=P^{-1}r^{(k)}$$

To compute the $z^{(k)}$ just solve

$$Pz^{(k)}=r^{(k)}\leftrightarrow Ax=b$$

Notice that P is nonsingular, and it is **arbitrarily chosen**, so we choose a diagonal, 3 diagonal or triangular matrix. Also as P is always the same, use LU factorization just once! We save a lot of computations!

4.3 Stationary Richardson Schemes

4.3.1 Jacobi and Gauss-Seidel methods

- **Jacobi**, we start from an example

$$\begin{cases}a_{11}x_1+a_{12}x_2+a_{13}x_3=b_1\\a_{21}x_1+a_{22}x_2+a_{23}x_3=b_2\\a_{31}x_1+a_{32}x_2+a_{33}x_3=b_3\end{cases}$$

For $a_{11} \neq 0$

$$\begin{cases} x_1 = \frac{1}{a_{11}}\left[b_1 - a_{12}x_2 - a_{13}x_3\right] \\ x_2 = \frac{1}{a_{22}}\left[b_2 - a_{21}x_1 - a_{23}x_3\right] \\ x_3 = \frac{1}{a_{33}}\left[b_3 - a_{31}x_1 - a_{32}x_2\right] \end{cases}$$

So far no approximation, consider

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{11}}\left[b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}\right] \\ \\ x_2^{(k+1)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}\right] \\ \\ x_3^{(k+1)} = \frac{1}{a_{33}}\left[b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)}\right] \end{cases}$$

Then the generic case

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

$$i = 1, \cdots, n \qquad a_{ii} \neq 0$$

This method **can be parallelized**, each variable $k+1$ depends only on variables at step k, strength point

- **Gauss-Seidel**, says to use the variables at the same step, it should converge/perform faster, but not guaranteed

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

$$i = 1, \cdots, n \qquad a_{ii} \neq 0$$

Cannot be parallelized

We now rewrite the schemes so we can reach Richardson-like form

$$x^{(k+1)} = \underbrace{\alpha_k P^{-1} b}_{g_{\alpha_k}} + \underbrace{\left(I - \alpha_k P^{-1} A\right)}_{B_{\alpha_k}} x^{(k)}$$

- **Jacobi**

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

$$D = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_{nn} \end{bmatrix}$$

$$\overrightarrow{x}^{(k)} = \left[x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}\right]^T$$

$$\vec{x}^{(k+1)} = \left[x_1^{(k+1)}, x_2^{(k+1)}, \cdots, x_n^{(k+1)}\right]^T$$
$$b = [b_1, \cdots, b_n]^T$$

So:

$$Dx^{(k+1)} = b - (A-D)x^{(k)}$$
$$x^{(k+1)} = x^{(k)} + D^{-1}(b - Ax^{(k)})$$

$$\begin{cases} x^{(k+1)} = x^{(k)} + 1 \cdot D^{-1} r^{(k)} \\ \alpha_{k_J} = 1, P_J = D \end{cases}$$
$$x^{(k+1)} = \underbrace{(I - D^{-1}A)}_{B_J} x^{(k)} + \underbrace{D^{-1}b}_{g_J}$$

- **Gauss-Seidel**

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right]$$
$$A = \begin{bmatrix} \ddots & & A-D+E \\ & D & \\ -E & & \ddots \end{bmatrix}$$

Where matrix E is strictly lower-triangular. Similar manipulations from before, we will get

$$x^{(k+1)} = x^{(k)} + (D-E)^{-1}(b - Ax^{(k)})$$

$$\begin{cases} x^{(k+1)} = x^{(k)} + (D-E)^{-1} r^{(k)} \\ \alpha_{k_{GS}} = 1, P_{GS} = (D-E) \end{cases}$$
$$x^{(k+1)} = \underbrace{(I - (D-E)^{-1}A)}_{B_{GS}} x^{(k)} + \underbrace{(D-E)^{-1}b}_{g_{GS}}$$

Both stationary, so parameter α not used to accelerate convergence, as it is constant.

4.3.2 Convergence for Jacobi and Gauss-Seidel

For Richardson schemes $x^{(k+1)} = Bx^{(k)} + g$, the requirements were for consistency and spectral radius $\rho(B) < 1$ (necessary and sufficient). The consistency property for Richardson schemes is automatically guaranteed.

Additionally, we have other sufficient conditions

- **Jacobi**
  - Necessary condition: $\rho(B_J) < 1$
  - Sufficient conditions (attention, referred to matrix **A**):

1. If A is strictly diagonally dominant by rows
2. If A is strictly diagonally dominant by columns

- **Gauss-Seidel**
  - Necessary condition: $\rho(B_{GS}) < 1$
  - Sufficient conditions (attention, referred to matrix **A**):
    1. If A is strictly diagonally dominant by rows
    2. If A is strictly diagonally dominant by columns
    3. If A is spd

We said that Gauss-Seidel does not necessarily outperforms Jacobi. Let $A \in \mathbb{R}^{n\times n}$ nonsingular, tridiagonal, with all $a_{ii} \neq 0$: both Jacobi and Gauss-Seidel are convergent or are divergent (they cannot be "discordant") and if they both are convergent

$$\rho(B_{GS}) = [\rho(B_J)]^2$$

The rate of convergent is determined by the spectral radius, which means GS converges faster, more specifically

$$\rho(B_J) = \frac{1}{4} \to \left(\frac{1}{4}\right) \leq \varepsilon \to k \geq \log_4\left(\frac{1}{\varepsilon}\right)$$

$$\rho(B_{GS}) = \frac{1}{4^2} \to \left(\frac{1}{4^2}\right) \leq \varepsilon \to 2k \geq \log_4\left(\frac{1}{\varepsilon}\right) \to k \geq \frac{1}{2}\log_4\left(\frac{1}{\varepsilon}\right)$$

$$\#\text{iterations}_{GS} \simeq \frac{\#\text{iterations}_J}{2}$$

4.3.3 Optimal acceleration parameter for stationary schemes

Let A and P be **spd matrices**. Then, the stationary Richardson scheme is convergent $\forall\, x^{(0)} \in \mathbb{R}^{n\times n}$ if and only if (necessary and sufficient)

$$0 < \alpha < \frac{2}{\lambda_{max}(P^{-1}A)}$$

Moreover

$$\alpha_{opt} = \frac{2}{\lambda_{\max}(P^{-1}A) + \lambda_{\min}(P^{-1}A)}$$

Finally

$$\underbrace{||e^{(k)}||_A}_{x-x^{(k)}} \leq \underbrace{\left(\frac{K(P^{-1}A)-1}{K(P^{-1}A)+1}\right)^k}_{<1} ||e^{(0)}||_A \qquad k \geq 0$$

With K the condition number and $||e||_A$ the energy norm:

$$||w||_A = \sqrt{w^T A w} \qquad w \leq \mathbb{R}^n$$

A proof for the necessary and sufficient condition:

- Prove that

$$0 < \alpha < \frac{2}{\lambda_{max}(P^{-1}A)}$$

  Consider $B_\alpha = I - \alpha P^{-1}A$, let λ_i eigenvalues of $P^{-1}A$ such that

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \dots \geq \lambda_n \geq 0$$

And let the generic eigenvalue of B_α:

$$1-\alpha\lambda_i$$

For convergence consistency holds since it is a Richardson scheme and $\rho(B)<1$ must hold as well, which means:

$$|1-\alpha\lambda_i|<1 \Rightarrow -1<1-\alpha\lambda_i<1 \Rightarrow \begin{cases} 1-\alpha\lambda_i<1 \\ 1-\alpha\lambda_i>-1 \end{cases}$$

The first condition is true for $\alpha>0$, aboud the second we will get $\alpha<\frac{2}{\lambda_i}\ \forall i$ which will result in:

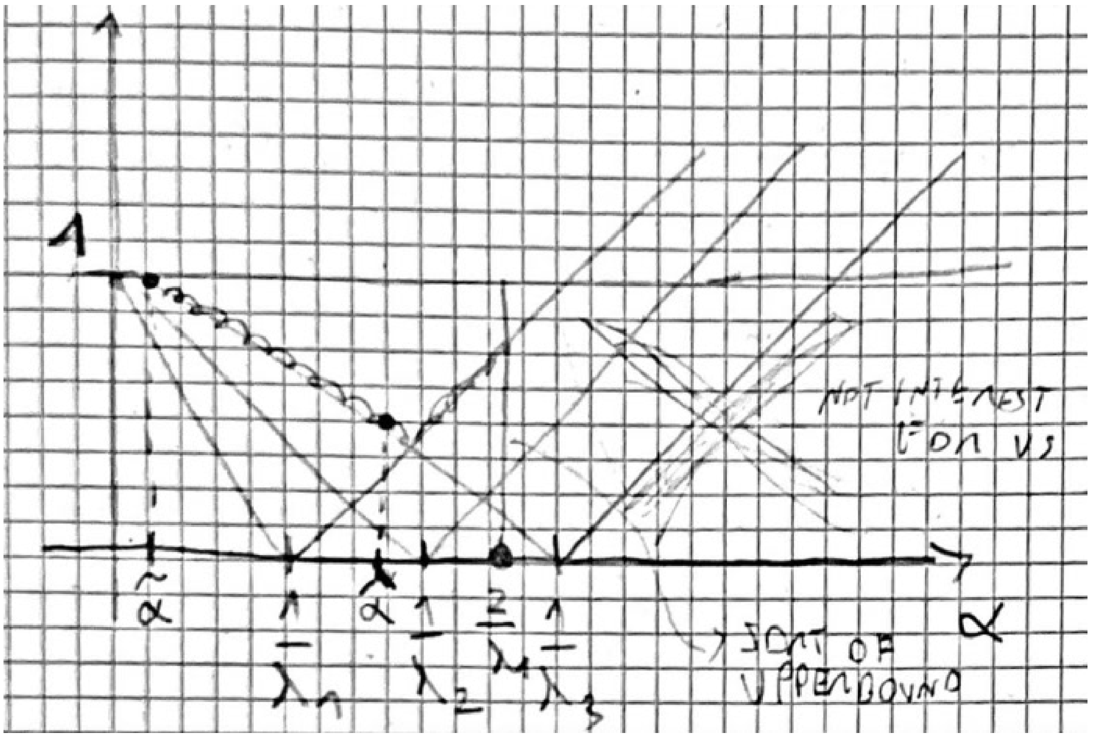
$$\frac{2}{\lambda_1}\le\frac{2}{\lambda_2}\le\cdots\le\frac{2}{\lambda_n}\le\Rightarrow \alpha<\frac{2}{\lambda_1}=\frac{2}{\lambda_{max}(P^{-1}A)}$$

- Prove that α_{opt} is that quantity. Consider a matrix $A\in\mathbb{R}^{3\times3}$ with $\lambda_1>\lambda_2>\lambda_3$. Remind that

$$B_\alpha=I-\alpha P^{-1}A$$

$$|1-\alpha\lambda_1| \qquad |1-\alpha\lambda_2| \qquad |1-\alpha\lambda_3|$$

Plotting them w.r.t. α:



Analyzing those 3 plots we can understand why the optimal value is that: we are searching for α_{opt} such that the method is as fast as possible. A method is quicker than the other if its spectral radius is smaller than the second, which means we are finding the value for α such as that the maximum eigenvalue (which is related to the spectral radius) is as small as possible.
On the graph we draw a vertical line in a $\tilde{\alpha}$ which will meet the 3 functions: we will get 3 eigenvalues, we consider the maximum.
Observing the plot, we can see that the maximum eigenvalue is on the branches highlighted, and the α that minimizes the maximum eigenvalue is given by the intersection of the two branches

$$\underbrace{1-\alpha\lambda_3}_{\text{Positive branch}} = \underbrace{\alpha\lambda_1-1}_{\text{Negative branch}} \Rightarrow \alpha_{opt}=\frac{2}{\lambda_1+\lambda_3}$$

Which is exactly

$$\alpha_{opt} = \frac{2}{\lambda_{\max}(P^{-1}A) + \lambda_{\min}(P^{-1}A)}$$

About the maximum rate of convergence, the spectral radius (with λ_n the minimum):

$$\rho(B_{\alpha_{opt}}) = 1 - \alpha_{opt}\lambda_n = 1 - \frac{2\lambda_n}{\lambda_1 + \lambda_n} = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$$

- Prove that

$$\underbrace{||e^{(k)}||_A}_{x-x^{(k)}} \leq \underbrace{\left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1}\right)^k}_{<1} ||e^{(0)}||_A \qquad k \geq 0$$

With K the condition number and $||e||_A$ the energy norm:

$$||w||_A = \sqrt{w^T A w} \qquad w \leq \mathbb{R}^n$$

We said that the preconditioner matrix P is:

  - Nonsingular
  - Easy to solve, as we have to sovle $Pz^{(k)} = r^{(k)}$
  - **Additional condition**, $K(P^{-1}A)$ small

4.4 Dynamic Richardson Scheme: preconditioned gradient methods

If A and P are spd matrices, the dynamic Richardson scheme is convergent if (sufficient)

$$\alpha_{k,opt} = \frac{\left[z^{(k)}\right]^T r^{(k)}}{\left[z^{(k)}\right]^T A z^{(k)}} \qquad \forall\, k \geq 0,\ z^{(k)} = P^{-1} r^{(k)}$$

We directly have optimal value for α, this method is called preconditioned gradient method. Moreover, we can exactly prove the same inequality

$$\underbrace{||e^{(k)}||_A}_{x-x^{(k)}} \leq \underbrace{\left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1}\right)^k}_{<1} ||e^{(0)}||_A \qquad k \geq 0$$

Remark: for $P = I$, our optimal recipe for α becomes:

$$\alpha_{k,opt} = \frac{\left[r^{(k)}\right]^T r^{(k)}}{\left[r^{(k)}\right]^T A r^{(k)}} \qquad \forall\, k \geq 0,\ z^{(k)} = P^{-1} r^{(k)}$$

Gradient method.

4.4.1 The algorithm

The algorithm has 4 steps

```
% set intial guess, we can also define associated initial residual
$x^{(\phi)} \in \mathbb{R}^n$        $r^{(\phi)} = b - Ax^{(\phi)}$
for k=0,1,...
    if $P = I$
        % got to step 2, 3 steps algorithm in this case
    else
        1) $Pz^{(k)} = r^{(k)}$
    2) $\alpha_{k,opt} = \frac{\left[z^{(k)}\right]^T r^{(k)}}{\left[z^{(k)}\right]^T A z^{(k)}}$
    3) $x^{(k+1)} = x^{(k)} + \alpha_{opt,k} z^{(k)}$
    4) $r^{(k+1)} = r^{(k)} - \alpha_{opt,k} A z^{(k)}$
```

If stationary, step 2) outside the for cycle

```
% set intial guess, we can also define associated initial residual
$x^{(\phi)} \in \mathbb{R}^n$        $r^{(\phi)} = b - Ax^{(\phi)}$
$\alpha_{opt} = \dots$
for k=0,1,...
    if $P = I$
        % got to step 3, 2 steps algorithm in this case
    else
        1) $Pz^{(k)} = r^{(k)}$
    3) $x^{(k+1)} = x^{(k)} + \alpha_{opt} z^{(k)}$
    4) $r^{(k+1)} = r^{(k)} - \alpha_{opt} A z^{(k)}$
```

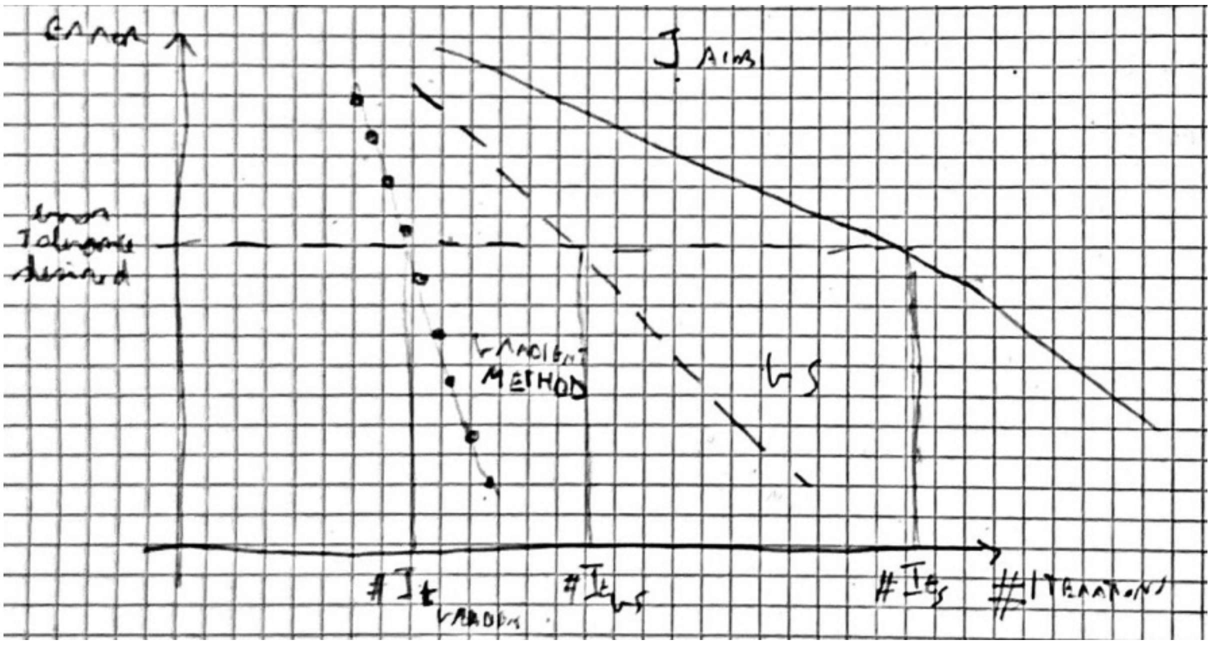
4.4.2 Stationary and dynamic: which one is the best

Now that we have found the optimal acceleration parameters α_{opt} for both stationary and dynamic cases, which one is better? For stationary we have to find the eigenvalues, but if the matrix is very big this is too demanding, though there are methods to cope with this.

It is a good idea to prefer dynamic scheme, as in the definition of the acceleration parameter we are using all parameters that are required to be computed for the solution of the system (in the stationary to find the eigenvalues we have to use other parameters unrelated and useless for the solution of our original problem).

An example

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



4.4.3 Proof for the optimal acceleration parameter

If A is spd, the solution to our system is: $Ax = b \Leftrightarrow$ equivalent to minimize the quadratic form

$$Q(x) = \frac{1}{2}x^T Ax - x^T b$$

as minimizing this quadratic form is solving the gradient w.r.t. 0:

$$\nabla Q(x) = Ax - b = 0$$

Note that the quadratic $Q(x)$ is a paraboloid and we are finding its minimum. How to proceed:

$$x^{(k+1)} = x^{(k)} + \gamma_k d^{(k)}$$

with $d^{(k)}$ the direction, **the steepest descent** and γ_k the step size. We choose the steepest direction, so the gradient:

$$d^{(k)} = -\nabla Q(x^{(k)}) = b - Ax^{(k)} = r^{(k)}$$

The steepest direction corresponds to the residual. About the step

$$Q\left(x^{(k)} + \gamma_k r^{(k)}\right) = \tilde{Q}(\gamma_k)$$

Which is the value of γ_k that minimizes Q? Compute the derivative and impose it to 0

$$\frac{d\tilde{Q}}{d\gamma_k} = 0$$

$$\tilde{Q}(\gamma_k) = Q\left(x^{(k)} + \gamma_k r^{(k)}\right) = \frac{1}{2}\left(x^{(k)} + \gamma_k r^{(k)}\right)^T A\left(x^{(k)} + \gamma_k r^{(k)}\right) - \left(x^{(k)} + \gamma_k r^{(k)}\right)^T b$$

Compute the derivative

$$\frac{d\tilde{Q}}{d\gamma_k} = \left[r^{(k)}\right]^T Ax^{(k)} + \gamma_k \left[r^{(k)}\right]^T Ar^{(k)} - \left[r^{(k)}\right]^T b = 0$$

$$\gamma_k = \frac{\left[r^{(k)}\right]^T \left(b - Ax^{(k)}\right)}{\left[r^{(k)}\right] Ar^{(k)}} = \frac{\left[r^{(k)}\right]^T r^{(k)}}{\left[r^{(k)}\right]^T Ar^{(k)}}$$

We found $\alpha_k = P\gamma_k$

$$Pz^{(k)} = r^{(k)}$$

$$\alpha^{(k)} = \frac{\left[z^{(k)}\right]^T r^{(k)}}{\left[z^{(k)}\right]^T Az^{(k)}}$$

4.5 Dynamic Richardson Scheme: conjugate gradient method

Gradient method can work with Hilbert system (matrix properly preconditioned): conjugate gradient method, 5 steps algorithm that selects a new direction $p^{(k)}$ instead of $d^{(k)}$:

$$\left[p^{(j)}\right]^T Ap^{(k+1)} = 0 \qquad j = 0, \cdots, k$$

New direction A-orthogonal (or A conjugate) w.r.t. the previous direction.

```
// set intial guess, we can also define associated initial residual
$x^{(0)} \in \mathbb{R}^n$        $r^{(0)} = b - Ax^{(0)}$
for k=0,1,...
    1) $\alpha_k = \dots$
    2) $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
    3) $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$
    4) $\beta_k = \dots$ // another constant for computing new direction
    5) $p^{(k+1)} = r^{(k+1)} - \beta_k p^{(k)}$
```

This method wants A spd, also the errors:

$$||e^{(k)}||_A \leq C^k ||e^{(0)}||_A$$

$$C := C\left(\sqrt{K(P^{-1}A)}\right) \qquad \text{Depends on sqrt root of condition number of } A$$

Consider the Hilbert problem/system, which we remind has a very bad condition number:

$$H_n x_n = b_n$$

n	$K(A_n)$	\	PG	P=D	PCG	P=D
4		$O(10^{-13})$	$O(10^{-3})$	995	$O(10^{-2})$	3
6					$O(10^{-2})$	4
8						4
10						5
12						5
14		$O(10)$	$O(10^{-3})$	1379	$O(10^{-3})$	5

If we can work in an exact arythmetic, this method becomes a direct method.

4.6 Stopping Criteria

Consider the error estimators: increment and residual

4.6.1 Residual

$$Ax = b$$
$$S = r^{(k)} = b - Ax^{(k)}$$
$$e^{(k)} = x - x^{(k)}$$

We want to relate the estimator

$$||e^{(k)}|| \qquad S$$

Normalize residual

$$\frac{||e^{(k)}||}{||x||} \leq C\frac{||r^{(k)}||}{||b||} \leq TOL \qquad x \neq 0,\ b \neq 0$$

The constant C discriminates the reliability or not of our estimator: if it's huge, our estimator is not reliable.
We want to stop at the minimum iteration $kmin$

$$\frac{||e^{(kmin)}||}{||x||} \leq \underbrace{C\frac{||r^{(kmin)}||}{||b||} \leq TOL}_{\text{Prove this}}$$

Remind that

$$\frac{||\delta x = x - \tilde{x}||}{||x||} \leq K(A)\frac{||r = b - A\tilde{x}||}{||b||}$$

If we consider $\tilde{x} = x^{(k)}$, we have that:

$$\delta x = e^{(k)} \qquad r = r^{(k)}$$

Which means

$$C = K(A)$$

So regarding the reliability, we have to look at the condition number

4.6.2 Increment

$$Ax = b$$

$$S = \delta^{(k)} = x^{(k+1)} - x^{(k)}$$

$$kmin \in \mathbb{N} \text{ s.t. } ||e^{(kmin)}|| \leq C\underbrace{||\delta^{(k)}|| \leq TOL}_{\text{Prove this}}$$

We start from the below and using triangular inequality:

$$||e^{(k)}|| = ||x - x^{(k)}|| = ||\underbrace{x - x^{(k+1)}}_{e^{(k+1)}} + \underbrace{x^{(k+1)} - x^{(k)}}_{\delta^{(k)}}|| \leq ||e^{(k+1)}|| + ||\delta^{(k)}|| \leq \rho(B)||e^{(k)}|| + ||\delta^{(k)}||$$

$$||e^{(k)}|| \leq \underbrace{\frac{1}{1-\rho(B)}}_{C}||\delta^{(k)}||$$

As we want C as close as possible to 1, we want the spectral radius as close as possible to 0. So increment is a reliable estimator if the spectral radius is very close to 0.
About kmin (minimum number of iterations):

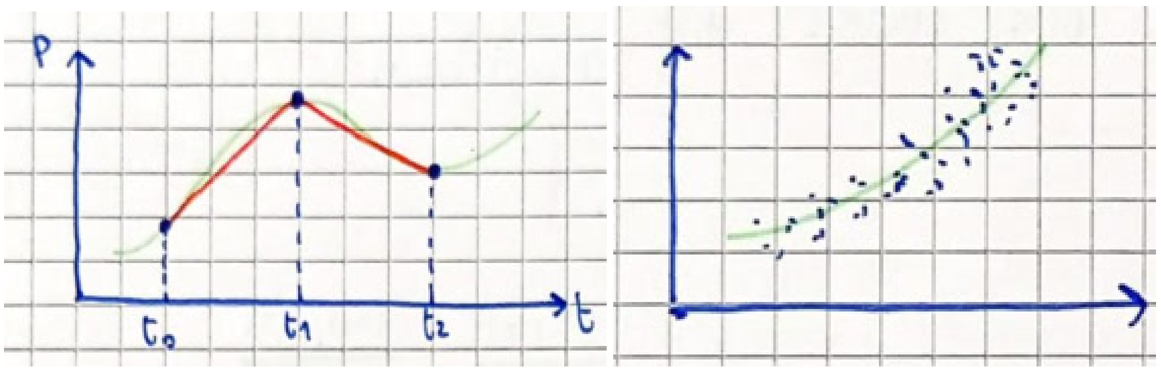
$$e^k = x - x^k$$

$$kmin = \frac{\log\left[\frac{\varepsilon\cdot(1-||B||_2)}{||x^1-x^0||}\right]}{\log||B||_2}$$

With $\varepsilon = TOL$

5 Approximation of Functions and Data

Approximation of data:



Left interpolation, right least square.

Approximation of functions like integral: approximate it to polynomial. The analytical tool to approximate a function is the Taylor expansion, but it suffers some problems

- Need higher order (more derivatives)
- It works well when we consider a neighborhood of the center of the expansion, but moving out of the neighborhood it will start to perform badly

5.1 Interpolation

Instead of Taylor, consider:

$$\begin{bmatrix} \textbf{Function} & & \textbf{Data} \\ f & \{(x_i, y_i)\} & i = 0, \cdots, n \\ y_i = f(x) & & x_i \text{ distinct} \end{bmatrix}$$

Identify a function $\tilde{f}$ s.t. $\tilde{f}(x_i) = y_i$ for $i = 0, \cdots, n$, **interpolation conditions**, $n+1$ conditions. The function $\tilde{f}$ can be:

- Polynomial
- Trigonometric expansion (Fourier expansion)
- Rational

$$\frac{a_0 + a_1 x + \cdots a_p x^p}{b_0 + b_1 x + \cdots b_q x^q}$$

We consider polynomial interpolation

5.1.1 Polynomial interpolation

Theorem: let $\{x_i, y_i\}_{i=0}^{n}$ (x_i are interpolation nodes, y_i are values to be interpolated) with x_i all distinct. Then $\exists!$ polynomial degree of $\leq n$ (interpolating/interpolation polynomial) s.t. (we guarantee interpolation condition):

$$\underbrace{\Pi_n(x_i) = y_i}_{\text{n+1 conditions}} \qquad i = 0, \cdots, n$$

Proof for uniqueness: by contradiction assume that we have 2 interpolating polynomials of order n

$$\Pi_n \in \mathbb{P}_n \qquad \Pi_n(x_i) = y_i \qquad i = 0, \cdots, n$$

$$\Pi_n^* \in \mathbb{P}_n \qquad \Pi_n^*(x_i) = y_i \qquad i = 0,\cdots,n$$

Consider the difference

$$D(x) = \Pi_n(x) - \Pi_n^*(x) \in \mathbb{P}_n$$

And

$$D(x_i) = \Pi_n(x_i) - \Pi_n^*(x_i) = y_i - y_i = 0 \qquad i = 0,\cdots,n$$

A polynomial of degree n has at most n intersections with the x-axis, but in this case $D(x)$ of degree n has $n+1$ zeros: the only way that we can satisfy the $n+1$ conditions is that $D(x)$ is identically equal to 0. which means

$$D(x) = 0 \Rightarrow \Pi_n(x) - \Pi_n^*(x) = 0 \Rightarrow \Pi_n(x) = \Pi_n^*(x)$$

Which contradicts our initial assumption.

Finding the characteristic polynomial: assume that values to be interpolated are all null except one

$$y_i = 0\ \forall i \neq k \qquad y_k = 1$$

$$\begin{Bmatrix} x_0 = 0 & x_1 = 0.5 & x_2 = 1 \\ y_0 = 0 & y_1 = 1 & y_2 = 0 \end{Bmatrix}$$

Let change notation, instead of Π_2 we use ϕ_k

$$\phi_1 \in \mathbb{P}_2 \qquad \underbrace{\phi_1(x_0 = 0) = 0}_{\text{A}} \qquad \underbrace{\phi_1(x_1 = 1/2) = 1}_{\text{B}} \qquad \underbrace{\phi_1(x_2 = 1) = 0}_{\text{C}}$$

We want to build a polynomial of degree 2 that is zero in those two points:

$$\begin{cases} (x-0)(x-1) & \text{satisfies A and B} \\ \frac{(x-0)(x-1)}{(0.5-0)(0.5-1)} & \text{satisfies C} \end{cases}$$

$$\phi_1(x) = \cdots = -4x(x-1)$$

With a generic case

$$\phi_k(x_i) = \delta_{ik} = \begin{cases} 0 & i \neq k \\ 1 & i = k \end{cases}$$

Kronecker delta, to express it as a polynomial with degree n, the characteristic polynomial:

$$\phi_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}$$

Moving to a more general case: instead of a set of arbitrary values

$$\begin{Bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ \Pi(x_0) = y_0 & \Pi(x_1) = y_1 & \Pi(x_2) = y_2 \end{Bmatrix}$$

Expressing Π_2 as linear combination of ϕ_0, ϕ_1 and ϕ_2

$$\begin{Bmatrix} \phi_0 & \phi_0(x_0) = 1 & \phi_0(x_1) = 0 & \phi_0(x_2) = 0 \\ \phi_1 & \phi_1(x_0) = 0 & \phi_1(x_1) = 1 & \phi_1(x_2) = 0 \\ \phi_2 & \phi_2(x_0) = 0 & \phi_2(x_1) = 0 & \phi_2(x_2) = 1 \end{Bmatrix}$$

$$\Pi_2(x) = a\phi_0(x) + b\phi_1(x) + c\phi_2(x)$$

Solving we will get:

$$a = y_0 \qquad b = y_1 \qquad c = y_2$$

The **Lagrange form**:

$$\Pi_n(x) = \sum_{k=0}^{n} y_k \phi_k(x)$$

$$\Pi_n(x) = \sum_{k=0}^{n} y_k \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j}$$

In matlab:

- **c = polyfit(x,y,n)**, to build interpolating polynomial
  - x vector collecting interpolation nodes, x_i
  - y is y_i
  - n is the degree of the polynomial, but is redundant as there is a strict relation between number of data and degree of polynomial (for n data, the polynomial will have degree of $n-1$), in least squares it will have a meaning

  It returns the coefficients of our interpolating polynomial

  $$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

  With c(1) coefficient of x^n, c(2) of $x(n-1)$
- **d = polyval(c,z)**, to evaluate interpolating polynomial at point z
  - If z single number $\mathbb{R}$, d will be a number
  - If z is $\mathbb{R}^q$, vector

5.1.2 Interpolation error

Error at nodes is null, at points that are not nodes? Consider a function continuous in a certain interval I:

$$f \in C^0(\bar{I}) \qquad I(x_0, x_n)$$

$$\{(x_i, y_i = f(x_i))\}_{i=0}^{n} \qquad x_i \text{ distinct}$$

Assuming

$$f \in C^{n+1}(\bar{I})$$

We define the interpolation error:

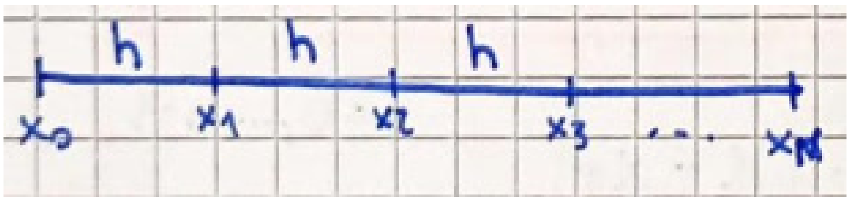
$$\forall x \in I$$

$$E_n f(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\alpha(x))}{(n+1)!} \prod_{k=0}^{n} (x - x_k)$$

And as expected $E_n f(x_i) = 0$. The weak point is that we assume regularity in the function, which depends on the number of nodes: such regularity uncommon. Another drawback is that the $\alpha(x)$ depends on x but we don't know the exact value of $\alpha(x) \in I$. In practice this result useless, so we consider the maximum value.

If we have more and more information, more samples, the degree of polynomial increases and we have more zeros (the function meets the x axis more times, like sinusoid) but the quality of the approximation

improves.
Consider an uniform partition of the interval



$$h = \frac{x_n - x_0}{n}$$

$$x_k = x_{k+1} + h \qquad k = 1, \cdots, n$$

$$x_j = x_0 + jh \qquad j = 0, \cdots, n$$

In this case we can prove:

$$\left| \prod_{k=0}^{n} (x - x_k) \right| \le n! \frac{h^{n+1}}{4}$$

Therefore:

$$\max_{x \in I} |E_n f(x)| \le \underbrace{\frac{h^{n+1}}{4(n+1)}}_{\text{A}} \cdot \underbrace{\max_{x \in I} \left| f^{(n+1)}(x) \right|}_{\text{B}}$$

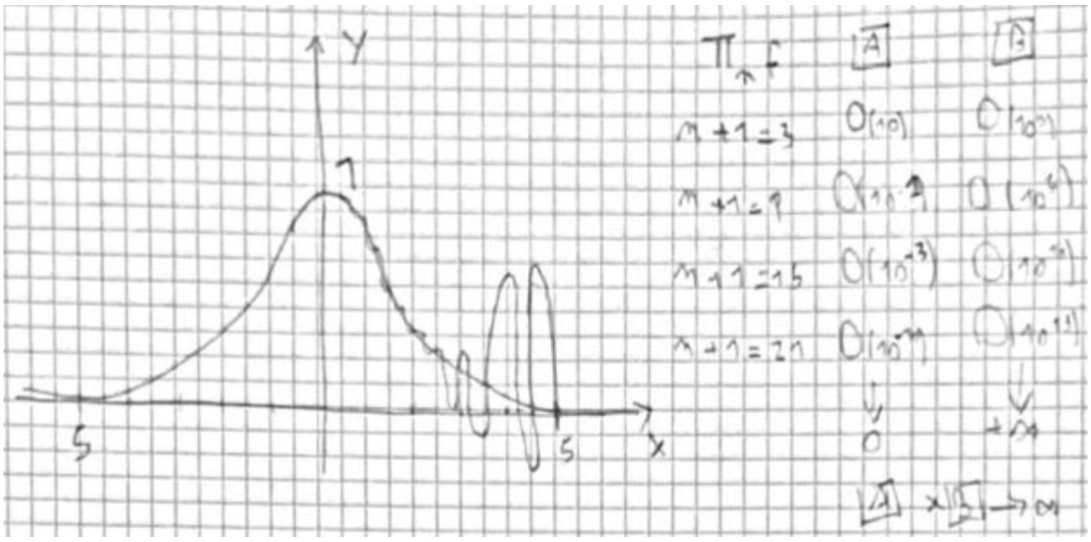
The two blocks:

- A→0 for $n \to \infty$
- B for $n \to \infty$ depends on f:

$$\text{B} \to \begin{cases} 0 \\ \text{constant} \\ +\infty \begin{cases} \text{If A goes to 0 quicker, OK} \\ \text{If B goes to } \infty \text{ quicker, not OK} \end{cases} \end{cases}$$

Consider infact the function

$$f(x) = \frac{1}{1+x^2} \qquad x \in I = [-5, 5]$$



We are in presence of spurious oscillations, in the case this error goes to ∞, we have the **Runge phenomenon: more samples, possibility of spurious oscillations**.
To deal with this we can:

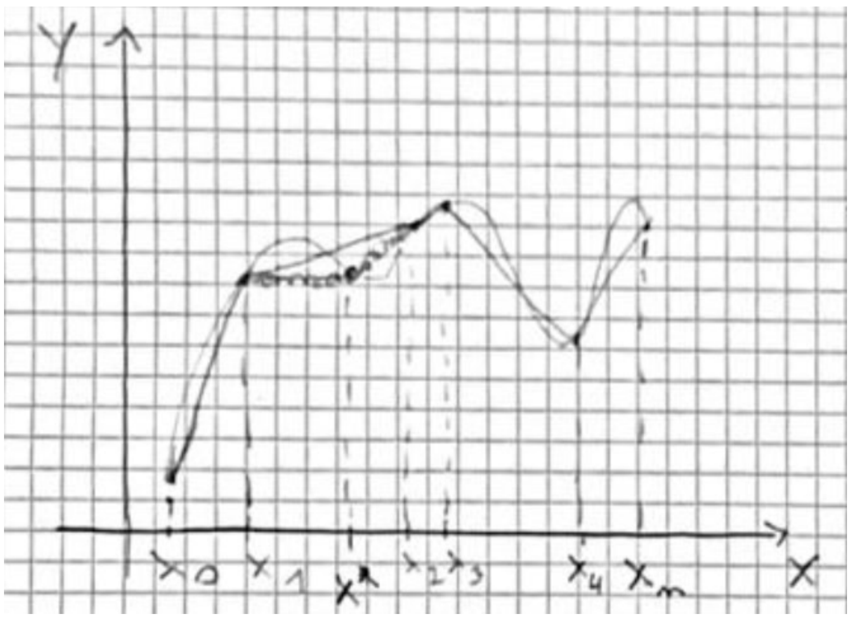
- Particular choice of nodes, Chebyshev nodes, as we chose uniform sampling, uniform distribution of nodes, not good idea: with Chebyshev we choose more nodes near the endpoints where the Runge phenomenon occurs, while at the center of the interval less nodes. To divide into n parts:
  1. n equal parts first
  $$\frac{\Pi_i}{n} \qquad i = 0, \cdots, n$$
  2. Compute the projection on the x-axis, the minus sign is in order to fix the order
  $$-f\left(\frac{\Pi_i}{n}\right)$$
  3. To use Chebyshev, use the following function that maps the interval I to the generic interval $[a,b]$, maps the nodes of previous step
  $$x_i = \frac{a+b}{2} + \frac{b-a}{2}\hat{x}_i$$
- When we increase n, a lot of x-axis crosses, so we can use low degree polynomials and work interval by interval: **piecewise linear interpolation**

5.1.3 Piecewise linear interpolation

We do not have to select an uniform distribution of nodes

$$I_j = [x_j, x_{j+1}] \qquad h_j = x_{j+1} - x_j$$
$$H = \max_j h_j$$

The idea is consider each subinterval and replace the function f with conjunction of endpoints (polynomial with low degree, so we avoid oscillations). By increasing the number of nodes, the approximation improves



We indicate linear interpolant as:

$$\Pi_1^H f \in C^0(\bar{I})$$

$$\Pi_1^H f\Big|_{I_j} \in \mathbb{P}_1(I_j)$$

$$\Pi_i^H f(x_k) = f(x_k) \qquad k = 0, \cdots, n$$

Express it as conjunction of endpoints that includes those 3 requirements:

$$\Pi_1^H f(x) = f(x_j) + \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}(x - x_j) \qquad x \in I_j$$

How to prove that the error becomes 0 for $n \to \infty$? We consider the local error on the subintervals I_j, we have (for $n = 1$):

$$\max_{x \in I_j} \left| E_1^H f(x) \right| \leq \frac{h_j^2}{4 \cdot (1+1)} \cdot \max_{x \in I_j} |f''(x)| \qquad f \in C^2(\bar{I}_j)$$

Instead of I_j and h_j, we want inequality on all I:

$$\max_{x \in I} \left| E_1^H f(x) \right| \leq \frac{H^2}{8} \cdot \max_{x \in I} |f''(x)| \qquad f \in C^2(\bar{I})$$

Now we have the error that is decreasing:

- A = $\frac{H^2}{8}$, for $n \to \infty$, the maximum length H will become smaller and smaller as we are increasing samples!
- B = $\max_{x \in I} |f''(x)|$ does not change, it's not dependent on n

We can also use piecewise parabola or cubic interpolation, when we join pieces of parabola is the approximation more regular? Is it a C^1 or still C^0? Not C^1, so by increasing local degree of polynomial we do not gain regularity, but we are improving accuracy of approximation. For example $n = 2$

$$\max_{x \in I_j} \left| E_2^H f(x) \right| \leq \frac{h_j^3}{4 \cdot (2+1)} \cdot \max_{x \in I_j} \left| f^{(3)}(x) \right| \qquad f \in C^3(\bar{I}_j)$$

$$\max_{x \in I} \left| E_1^H f(x) \right| \leq \frac{H^3}{12} \cdot \max_{x \in I} \left| f^{(3)}(x) \right| \qquad f \in C^3(\bar{I})$$

This is a problem, we would like something that is globally very smooth.

In matlab:

- **d = interp1(x,y,z)**, in some sense merges both polyfit and polyval, output will have same dimension as z
- **d = interp2(x,y,z)**, cubic interpolation

A remark: the matlab function plot is doing a sampling of the function, which is finer in the gradient of the function. This is known as **adaptive sampling**

5.1.4 Cubic spline interpolation

Again, a piecewise interpolation, but we join endpoints with a cubic function

1. $S_3\Big|_{I_j} \in \mathbb{P}$
2. Spline means function **smooth globally**, the pieces are joined so that the function is globally, $S_3 \in C^2(\bar{I})$
3. $S_3(x_i) = f(x_i) \qquad i = 0, \cdots, n$

In matlab **d = spline(x,y,z)**, build and directly evaluate. For each interval we have a polynomial of degree 3: S_3 (so a_i for $i = 4$).We have 4 unknowns for each interval, let #intervals = n, so $4n$ unknowns. The procedure:

1) $S_3(x_i) = f(x_i) \qquad i = 0, \cdots, n$

2) We demand S_3 continuous in the nodes, $S_3 \in C^0([x_0, x_n])$

$$[S_3(x_i)]^- = [S_3(x_i)]^+ \qquad i = 1, \cdots, n-1$$

3) We demand S_3' continuous in the nodes, $S_3' \in C^0([x_0, x_n])$

$$\left[S_3'(x_i)\right]^- = \left[S_3'(x_i)\right]^+ \qquad i = 1, \cdots, n-1$$

4) We demand S_3'' continuous in the nodes, $S_3'' \in C^0([x_0, x_n])$

$$\left[S_3''(x_i)\right]^- = \left[S_3''(x_i)\right]^+ \qquad i = 1, \cdots, n-1$$

In total we have $(n+1) + 3(n-1) = 4n-2$ conditions, but we need two more:

- $S_3''(x_0) = S_3''(x_n) = 0$, natural cubic interpolating spline
- Not-a-knot-condition: S_3''' continuous at x_1, x_{n-1}

$$\left[S_3'''(x_1)\right]^- = \left[S_3'''(x_1)\right]^+ \qquad \left[S_3'''(x_{n-1})\right]^- = \left[S_3'''(x_{n-1})\right]^+$$

The error:

$$\max_{x \in [x_0, x_n]} \left| f^{(r)}(x) - S_3^{(r)}(x) \right| \le C_r \cdot H^{4-r} \cdot \max_{x \in [x_0, x_n]} \left| f^{(4)}(x) \right| \qquad r = 0, 1, 2$$

5.2 Least Squared Approximation

Cloud of data $\left\{(x_i, y_i)\right\}_i^n$, x_i distinct, we find a polynomial $\tilde{f} \in \mathbb{P}_m$ of degree $m \ge 1, m << n$ and $\tilde{f}(x_i) \ne y_i$ such that:

$$\underbrace{\sum_{i=0}^{m} \left[\tilde{f}(x_i) - y_i\right]^2}_{\text{A}} \le \underbrace{\sum_{i=0}^{n} \left[p_m(x_i) - y_i\right]^2}_{\text{B}} \qquad \forall\, p_m \in \mathbb{P}_m$$

We want to minimize the right term. **To approximate the set of data we can resort to least squares approximation of a suitable degree according to the the location only (does not depend on the number) of the pairs in the Cartesian plane**.

5.2.1 Degree n

$m = n$ Lagrange interpolant, $\tilde{f} == \Pi_n$

5.2.2 Degree 1

$m = 1$ regression line

$$p_1(x) = b_0 + b_1 x \qquad b_0, b_1 \in \mathbb{R}$$
$$\tilde{f}(x) = a_0 + a_1 x \qquad a_0, a_1 \in \mathbb{R}$$

We want to find the specific polynomial $\tilde{f}$ (so the two coefficients). Consider the definition, the blocks:

- A = $\sum_{i=0}^{n} [a_0 + a_1 x_i - y_i]^2 = \Phi(a_0, a_1)$
- B = $\sum_{i=0}^{n} [b_0 + b_1 x_i - y_i]^2 = \Phi(b_0, b_1)$

So:

$$\Phi(a_0, a_1) \le \Phi(b_0, b_1) \qquad \forall\, b_0, b_1 \in \mathbb{R}$$

We compute the partial derivatives to find a_0 and a_1:

$$\frac{\partial \Phi}{\partial b_0}\bigg|_{(b_0,b_1)=(a_0,a_1)} = 0 \qquad \frac{\partial \Phi}{\partial b_1}\bigg|_{(b_0,b_1)=(a_0,a_1)} = 0$$

By developing $\Phi(b_0, b_1)$

$$\Phi(b_0, b_1) = \sum_{i=0}^{n} \left[b_0^2 + b_1^2 x_i^2 + y_i^2 + 2b_0 b_1 x_i - 2b_0 y_i - 2b_1 x_i y_i\right]$$

And the partial derivatives:

$$\frac{\partial \Phi}{\partial b_0} = \sum_{i=0}^{n} [2b_0 + 2b_1 x_i - 2y_i] \qquad \frac{\partial \Phi}{\partial b_0} = \sum_{i=0}^{n} \left[2b_1 x_i^2 + 2b_0 x_i - 2x_i y_i\right]$$

Putting a_0 and a_1

$$\begin{cases} \sum_{i=0}^{n} [2a_0 + 2a_1 x_i - 2y_i] = 0 \\ \sum_{i=0}^{n} \left[2a_1 x_i^2 + 2a_0 x_i - 2x_i y_i\right] = 0 \end{cases} \quad \rightarrow B\vec{a} = \vec{f}$$

$$\begin{cases} \sum_{i=0}^{n} a_0 + \sum_{i=0}^{n} a_1 x_i = \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} a_1 x_i^2 + \sum_{i=0}^{n} a_0 x_i = \sum_{i=0}^{n} x_i y_i \end{cases} \Rightarrow \begin{cases} a_0(n+1) + a_1 \sum_{i=0}^{n} x_i = \sum_{i=0}^{n} y_i \\ a_1 \sum_{i=0}^{n} x_i^2 + a_0 \sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i y_i \end{cases}$$

$$B = \begin{bmatrix} (n+1) & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \qquad \vec{f} = \left[\sum y_i, \sum x_i y_i\right]^T \qquad \vec{a} = [a_0, a_1]^T$$

Where B is spd (we can use gradient method)

5.2.3 Degree m generic

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$$
$$p_m(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m$$
$$\frac{\partial \Phi}{b_i}\bigg|_{(b_0,b_1,\cdots,b_m)=(a_0,a_1,\cdots,a_m)} = 0 \qquad i = 0, \cdots, m$$

Just like before, we wanto to find $a_0, \cdots, a_m$, after the calculations:

$$B = \begin{bmatrix} (n+1) & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \cdots & \sum x_i^{m+1} \\ \vdots & & & & \\ \sum x_i^m & \sum x_i^{m+1} & \sum x_i^{m+2} & \cdots & \sum x_i^{2m} \end{bmatrix}$$

$$\vec{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} \qquad \vec{f} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^m y_i \end{bmatrix}$$

5.3 Numerical Integration

$$I(f) = \int_a^b f(x)dx \qquad f \in C^0([a,b])$$

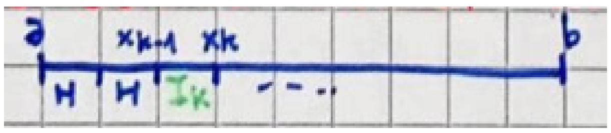
The quadrature rule: approximate a function:

$$\tilde{I}(f) \simeq I(f)$$

$$\tilde{I}(f) = \int_a^b \tilde{f}(x)dx \qquad \tilde{f}(x) \simeq f(x)$$

The steps:

- Subdivide into M intervals, the points are known as quadrature nodes

  $M+1$ points, so M intervals, x_i are quadrature nodes. The amplitude:

  $$H = \frac{b-a}{M}$$

  Uniform partition of quadrature nodes:

  $$x_{i+1} = x_i + H \qquad i = 0, \cdots, M-1$$

  $$x_i = x_0 + (i)H \qquad i = 1, \cdots, M$$

- Expand the additivity and associativity of integral:

  $$\int_a^b f(x)dx = \sum_{k=1}^{M} \int_{I_k} f(x)dx \cong \sum_{k=1}^{M} \int_{I_k} \tilde{f}(x)dx = \tilde{I}(f)$$

  With $\tilde{f}(x) \in \mathbb{P}_m$, for different choices of m different quadrature rules

5.3.1 Newton-Cotes

Interpolatory quadrature rule:

$$\tilde{I}(f) = \sum_{i=1}^{J} \alpha_i f(x_i)$$

With α_i quadrature weights, x_i quadrature nodes. We have first to define:

- **Order of accuracy**, associated only to the composite rule, "rate of convergence for the quadrature rule to zero- of the error" (it's the power of H in the composite error): **it's the convergence rate, that we can compute with piecewise by plotting H against the error, H, H^2, till n of C^n in loglog and take the line parallel to the error. Higher is the order of accuracy, faster the method converges**.

- **Degree of exactness**, associated both to composite and simple, maximum degree of the polynomials which are exactly integrated by your quadrature rule: **a method exactly integrates a polynomial with degree lower than the degree of exactness of the method**. Suppose:

$$I(f) = \int_a^b f(x)dx \qquad \tilde{I}(f)$$

  We start from polynomial of degree 0 p_0, but they are infinite: choose a representant

$$I(1) =? = \tilde{I}(1)$$

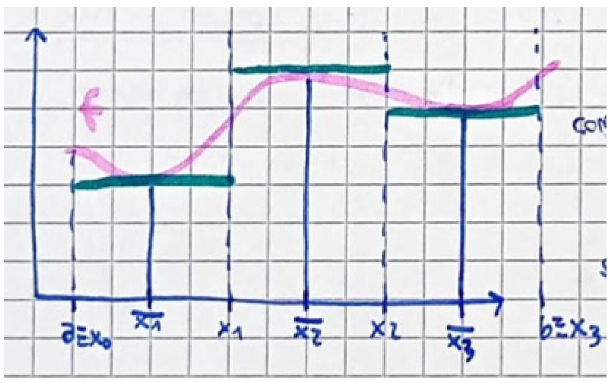
  Then continue till this check does not hold. But in practice the following equality holds:

$$de = [\text{degree of derivative}] - 1$$

  If we know the explicit expression of the error (E) we try different degrees in order to make it zero: de is the order of derivative-1. Also a formula with $n+1$ nodes has at most degree of exactness $2n+1$

We have number of subintervals M with width of $H = \frac{b-a}{M}$

- $m = 0$, **midpoint quadrature rule**, which means for each subinterval $\tilde{f}$ is a constant function



$$\begin{cases} \tilde{f} \in \mathbb{P}_0 \\ I_k = [x_{k-1}, x_k] \\ \bar{x}_k = \frac{x_{k-1}+x_k}{2} \end{cases}$$

  The composite midpoint quadrature rule:

$$\tilde{I}^c_{MP}(f) = H \sum_{k=1}^{M} f(\bar{x}_k)$$

  Where c stands for composite, the simple midpoint quadrature rule version:

$$\tilde{I}_{MP}(f) = (b-a)f\left(\frac{a+b}{2}\right)$$

  The number of subintervals is equal to the number of total nodes
  Errors:

  - Simple midpoint quadrature rule error

$$I(f) - \tilde{I}_{MP}(f) = \int_a^b [f(x) - f(\bar{x})]$$

Use the Taylor expansion centered at $\bar{x}$, $f \in C^2([a,b])$, second order:

$$f(x) - f(\bar{x}) = f'(\bar{x})(x-\bar{x}) + \frac{f''(\alpha(x))}{2}(x-\bar{x})^2$$

Making the computations...

$$\tilde{E}_{MP} = I(f) - \tilde{I}_{MP}(f) = \frac{(b-a)^3}{24} f''(\beta)$$

$$f \in C^2([a,b]) \qquad \beta \in [a,b]$$

β (from min value theorem of integral) cannot be found, in practice upperbound for worst case

– Composite midpoint quadrature rule error

$$I(f) - \tilde{I}^c_{MP}(f) = \sum_{k=1}^{M} \left[\int_{I_k} f(x)dx - \tilde{I}_{MP}\left(f\big|_{I_k}\right) \right] = \sum_{k=1}^{M} \frac{H^3}{24} f''(\beta_k)$$

Again from min value theorem of summation (dual of integral one):

$$= \frac{H^3}{24} f''(\gamma) \sum_{k=1}^{M} 1 == \frac{H^3}{24} f''(\gamma) M =$$

With $H = \frac{b-a}{M} \to M = \frac{b-a}{H}$, so:

$$\tilde{E}^c_{MP} = I(f) - \tilde{I}^c_{MP}(f) = \frac{(b-a)}{24} H^2 f''(\gamma)$$

$$f \in C^2([a,b]) \qquad \gamma \in [a,b]$$

Order of accuracy: $oa_{MP} = 2$
Degree of exactness: $de_{MP} = 1$

– p_0 degree 0

$$I(1) =?= \tilde{I}(1)$$

$$(b-a) =?= (b-a)f\left(\frac{a+b}{2}\right) = (b-a)$$

OK

– p_1 degree 1

$$I(x) =?= \tilde{I}(x)$$

$$\frac{x^2}{2}\Big|_a^b =?= (b-a)\frac{a+b}{2}$$

OK, make the computations

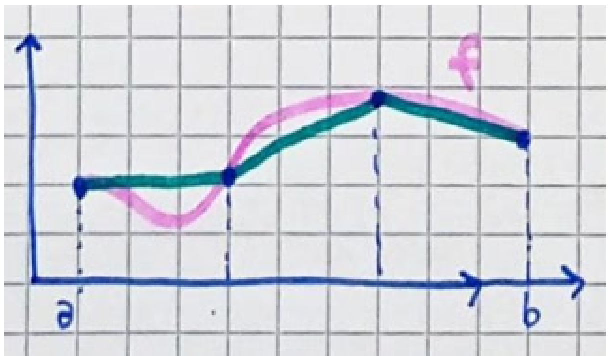
– p_2 degree 2

$$I(x^2) =?= \tilde{I}(x^2)$$

$$\frac{x^3}{3}\Big|_a^b =?= (b-a)\left[\frac{a+b}{2}^2\right]$$

KO, we stop here, so $de_{MP} = 1$

- $m = 1$, **trapezoidal quadrature rule**, for each subinterval a linear function



The composite trapezoidal quadrature rule (just basis times height):

$$\tilde{I}_T^c(f) = \frac{H}{2}\sum_{k=1}^{M}[f(x_{k-1}) + f(x_k)] = \frac{H}{2}[f(a)+f(b)] + H\sum_{k=1}^{M-1} f(x_k)$$

The first expression associated to intervals, the second to nodes. The simple trapezoidal quadrature rule version:

$$\tilde{I}_T(f) = \frac{(b-a)}{2}[f(a)+f(b)]$$

The number of subintervals is equal to the number of total nodes minus 1
Errors:

- Simple trapezoidal quadrature rule

$$I(f) - \tilde{I}_T(f) = \int_a^b [f(x) - \Pi_1(f)(x)]\,dx$$

  The term inside the integral is:

$$E_1 = \frac{f''(\eta(x))}{2}(x-a)(x-b)$$

  From

$$E_n f(x) = \frac{f^{(n+1)}(\alpha(x))}{(n+1)!}\prod_{k=0}^{n}(x-x_k)$$

  Making the computations...

$$\tilde{E}_T = I(f) - \tilde{I}_T(f) = -\frac{(b-a)^3}{12}f''(\delta)$$

$$f \in C^2([a,b]) \qquad \delta \in [a,b]$$

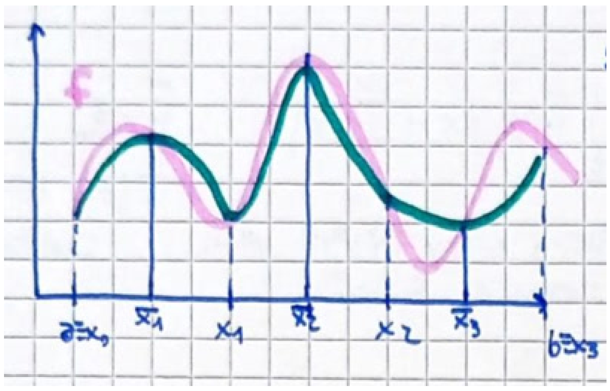
- Composite trapezoidal quadrature rule

$$\tilde{E}_T^c = I(f) - \tilde{I}_T^c(f) = -\frac{(b-a)}{12}H^2 f''(\rho)$$

$$f \in C^2([a,b]) \qquad \rho \in [a,b]$$

Order of accuracy: $oa_T = 2$
Degree of exactness: $de_T = 1$
Though *oa* and *de* same as midpoint, we see that the error is twice that of the midpoint: midpoint is easier and has lower error

- $m = 2$, **Simpson quadrature rule**



$$\tilde{I}_S^c(f) = \frac{H}{6}\sum_{k=1}^{M}\left[f(x_{k-1}) + 4f(\overline{x}_k) + f(x_k)\right] =$$

$$= \frac{H}{6}\left[f(a) + f(b)\right] + \frac{H}{3}\sum_{k=1}^{M-1} f(x_k) + \frac{2}{3}H\sum_{k=1}^{M} f(\overline{x}_k)$$

The simple Simpson quadrature rule version:

$$\tilde{I}_S(f) = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

The number of subintervals is equal to (#nodes - 1)/2
Errors:

- Simple Simpson quadrature rule

$$\tilde{E}_S = I(f) - \tilde{I}_S(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\sigma)$$

$$f \in C^4([a,b]) \qquad \sigma \in [a,b]$$

- Composite Simpson quadrature rule

$$\tilde{E}_S^c = I(f) - \tilde{I}_S^c(f) = -\frac{(b-a)}{180} H^4 f^{(4)}(\eta)$$

$$f \in C^4([a,b]) \qquad \eta \in [a,b]$$

Order of accuracy: $oa_S = 4$
Degree of exactness: $de_S = 3$

- p_0 degree 0

$$I(1) =? = \tilde{I}(1)$$

$$(b-a) =? = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] = \frac{(b-a)}{6}\left[1+4+1\right] = (b-a)$$

As f is 1 evaluated everywhere.
OK

- p_1 degree 1
$$I(x) = ? = \tilde{I}(x)$$
$$\left.\frac{x^2}{2}\right|_a^b = ? = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$
OK, make the computations

- p_2 degree 2
OK

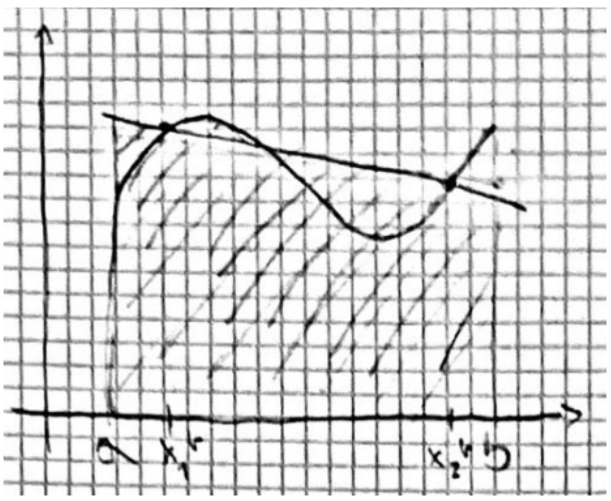
- p_3 degree 3
KO, we stop here, so $de_S = 3$

5.3.2 Gaussian quadrature rules

Till now we have seen Newton-Cotes formulas, while now we will see Gaussian formulas:

- In the Newton-Cotes when we fix the partition into subintervals for the integration we immediately know where are the quadrature nodes: e.g. when we fix the degree of the polynomial to 1, trapezoidal, the quadrature nodes are the endpoints, if degree to 2, Simpson, quadrature nodes are the endpoints and middle point
- In Gaussian we do not know a priori the position of the quadrature nodes: the quadrature nodes are neither the endpoints nor the middle point of the interval, but special points

For a certain number J of quadrature nodes, the Gaussian quadrature rules maximize the degree of exactness. If we consider $J = 2$:
$$\sum_{j=1}^{2} \alpha_j f(x_j)$$
We must solve a system of 4 unknowns, the two quadrature nodes x_j and the two quadrature weights α_j. This system of equations to find the unknowns result in ensuring that the $de = 0$ (ensured by first equation), $de = 1$ (ensured by second equation), ..., $de = 3$ (ensured by fourth equation):
$$\begin{cases} de = 0 & I(1) = b - a = \tilde{I}_G(1) = \alpha_1 + \alpha_2 \\ de = 1 & I(x) = \frac{b^2}{2} - \frac{a^2}{2} = \tilde{I}_G(x) = \alpha_1 x_1 + \alpha_2 x_2 \\ de = 2 & I(x^2) = \frac{b^3}{3} - \frac{a^3}{3} = \tilde{I}_G(x^2) = \alpha_1 x_1^2 + \alpha_2 x_2^2 \\ de = 3 & I(x^3) = \frac{b^4}{4} - \frac{a^4}{4} = \tilde{I}_G(x^3) = \alpha_1 x_1^3 + \alpha_2 x_2^3 \end{cases}$$
Making some computations, we will find that x_1 and x_2 are symmetric w.r.t. middle point of the interval:
$$\tilde{I}_G(f) = \frac{b-a}{2}\left[f(x_1^G) + f(x_2^G)\right]$$
Which is similar to the trapezoidal rule, but with different quadrature nodes. The resulting trapezoid:



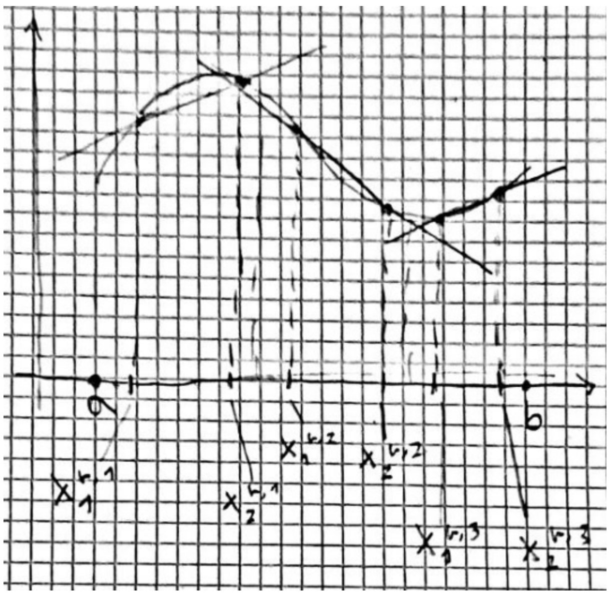
The error:

$$I(f) - \tilde{I}_G(f) = \frac{1}{4320}(b-a)^5 f^{(4)}(\rho)$$

The Gaussian rule with two nodes ($J = 2$) provides same degree of exactness and results of Simpson rule with one less node (Simpson used $J = 3$ nodes).

If we introduce intervals (composite Gaussian case), we will get discontinuous approximate function.

$$\tilde{I}_G^c(f) = \frac{H}{2}\sum_{i=1}^{M}\left[f(x_1^{G,i}) + f(x_2^{G,i})\right]$$



The error:

$$I(f) - \tilde{I}_G(f) = \frac{(b-a)}{4320}H^4 f^{(4)}(\nu)$$

And in conclusion:

$$oa = 4 \qquad de = 3$$

In general for degree J we have $2J$ unknowns and $de = 2J - 1$

5.4 Numerical Differentiation

The goal is approximate derivative of a function **at a certain single value**:

$$f'(\bar{x}) \qquad f \in C^1([a,b]) \qquad \bar{x} \in [a,b]$$

Starting from the definition of derivative:

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

5.4.1 Forward and backward finite difference

- **Forward finite difference**

$$f'(\bar{x}) \cong \delta_+ f(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x})}{h}$$

For h closer to 0, this difference will be sharper. If we build the characteristic polynomial:

$$\Pi_1 f \qquad (\bar{x}, f(\bar{x})) \qquad (\bar{x}+h, f(\bar{x}+h))$$

And derive it, we get exactly the forward finite difference:

$$\Pi_1 f(x) = f(\bar{x})\underbrace{\phi_{\bar{x}}(x)}_{\frac{x-\bar{x}-h}{\bar{x}-\bar{x}-h}} + f(\bar{x}+h)\underbrace{\phi_{\bar{x}+h}(x)}_{\frac{x-\bar{x}}{\bar{x}+h-\bar{x}}} = f(\bar{x})\frac{x-\bar{x}-h}{h} + f(\bar{x}+h)\frac{x-\bar{x}}{h}$$

$$(\Pi_1 f(x))' = -f(\bar{x})\frac{1}{h} + f(\bar{x}-h)\frac{1}{h} = \frac{f(\bar{x}+h) - f(\bar{x})}{h} = \delta_+ f(\bar{x})$$

$$(\Pi_1, f)' = \delta_+ f(\bar{x})$$

To compute the error we use the Taylor expansion:

$$f(\bar{x}+h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2} f''(\alpha) \qquad \alpha \in [\bar{x}, \bar{x}+h] \qquad f \in C^2([a,b])$$

Divide everything by h:

$$f'(\bar{x}) + \underbrace{\frac{f(\bar{x})}{h} - \frac{f(\bar{x}+h)}{h}}_{-\delta_+ f(\bar{x})} = -\frac{h}{2} f'(\alpha)$$

So:

$$-\frac{h}{2} f''(\alpha) = f'(\bar{x}) - \delta_+ f(\bar{x})$$

This scheme is a first order scheme (h has degree 1).

- **Backward finite difference**

$$f'(\bar{x}) \cong \delta_- f(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x}-h)}{h}$$

In terms of accuracy, this scheme is equivalent fo the forward finite difference, still a first order scheme. Using the Taylor expansion to compute the error:

$$f(\bar{x}-h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2} f''(\beta) \qquad \beta \in [\bar{x}-h, \bar{x}] \qquad f \in C^2([a,b])$$

Divide everything by h, make computations, we will get:

$$\frac{h}{2} f''(\beta) = f'(\bar{x}) - \delta_- f(\bar{x})$$

The same error with an inverted sign.

5.4.2 Centered finite difference

How to generate a scheme with higher accuracy, e.g. of second order? We can "combine the point before and the point after":

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

And:

$$f'(\bar{x}) \simeq \delta f(\bar{x}) = \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h}$$

Centered finite difference, for the error we considerate two Taylor expansions and we demand $f \in C^3([a,b])$:

$$\begin{cases} f(\bar{x}+h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2}f''(\bar{x}) + \frac{h^3}{6}f^{(3)}(\gamma) & \gamma \in [\bar{x}, \bar{x}+h] \\ \\ f(\bar{x}-h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2}f''(\bar{x}) - \frac{h^3}{6}f^{(3)}(\rho) & \rho \in [\bar{x}-h, \bar{x}] \end{cases}$$

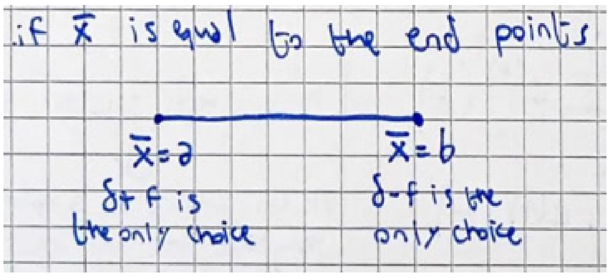
Subtract the two expansions, we will get:

$$f(\bar{x}+h) - f(\bar{x}-h) = 2hf''(\bar{x}) + \frac{h^3}{6}\left[f^{(3)}(\gamma) + f^{(3)}(\rho)\right]$$

Divide by $2h$ and making the computations we get:

$$f'(\bar{x}) - \delta f(\bar{x}) = -\frac{h^2}{12}\left[f^{(3)}(\gamma) + f^{(3)}(\rho)\right]$$

This last scheme is the best one, but we might have to use the forward or backward scheme when $\bar{x}$ is one of the endpoints:


if $\bar{x}$ is equal to the end points
$\bar{x}=a$
$\delta_+ f$ is the only choice
$\bar{x}=b$
$\delta_- f$ is the only choice

6 Matlab

6.1 Basics

```
close all
clear all
clc

addpath("../utils")

% function definitions============================================================
f = @(x) (2 + exp(1)) * x .^ 2 + (2 * exp(1) + 1) * x - cosh(x - 1);

% plot the function f============================================================
a = 0.5; b = 5.5;
x_plot = linspace(a, b, 1000);
figure, hold on, box on, grid on, axis equal;
plot(x_plot, f(x_plot), "k-", "LineWidth", 1);
xlabel("x","FontSize", 16)
ylabel("f(x)","FontSize", 16)
set(gca,"FontSize", 16)
set(gca,"LineWidth", 1.5)

% Errors behaviour================================================================
figure, semilogy(err1);

% LU decomposition================================================================
[L, U, P] = lu(A);

% Cholesky decompistion===========================================================
eigenvalues = eig(A);
isreal(eigenvalues);
all(eigenvalues > 0);
R = chol(A);

% Sparse matrix===================================================================
a = [1:10]';
c = [10:19]';
e = [102:111]';
n = length(a);
A_sparse = spdiags([e a c], [-1 0 1], n, n);
% insert into diagonal (0), just below (-1) and just on top(1)
% c on top will have first element (10) deleted
% e below will have last element (111) deleted
spy(A_sparse);

% Condition number===============================================================
cond(A); % w.r.t. 2-norm or spectral norm
cond(A,p); % w.r.t. p-norm
cond(A,'inf'); % infinitu-norm
condest(A_sparse); % 1-norm

% Lagrange interpolation=========================================================
coeffs = polyfit(x,y,n) % x nodes, y targets, n degree of polynomial, but
    redundant
                        % for n data the degree is n-1, meaning least squares
                            only
d = polyval(coeffs,z) % evaluate interpolating polynomial at point z

% Piecewise======================================================================
d = interp1(x,y,z) % piecewise polynomial interpolation
d = interp2(x,y,z) % cubic interpolation
```

```
% Cubic spline==============================================================
d = spline(x,y,z) % cubic spline interpolation
```

Some util functions:

```
function [x, x_iter] = bisection(f, a, b, tol)

function [x, x_iter] = newton(f, df, x0, tol, Nmax)

function [x, x_iter] = modified_newton(f, df, x0, tol, Nmax, m)

function [xi, x_iter_bisection, x_iter_newton] = bisection_newton(f, df, a, b,
    tol_bisection, tol_newton, maxit_newton, multiplicity)

function [xi, x_iter] = fixed_point(phi, x0, tol, maxit)

function y = forward_substitution(L, b)

function x = backward_substitution(U, b)

function [L, U, x] = thomas(A_sparse, b)

function [x, iter, incr] = stationary_method(B, g, x0, tol, max_it)

function [x, iter, incr] = prec_rich_method(A, b, P, alpha, x0, tol, max_it)

function int = midpoint(f, a, b)

function int = composite_midpoint(f, a, b, m)

function int = trapezoidal(f, a, b)

function int = composite_trapezoidal(f, a, b, m)

function int = simpson(f, a, b)

function int = composite_simpson(f, a, b, m)
```

6.2 Nonlinear equations

6.2.1 Manually retrieve roots intervals, bisection and error evaluation

Consider:

$$\cot(x) = \frac{x^2 - 1}{2x}$$

Define three intervals containing the three smallest positive roots

```
f = @(x) cot(x);
g = @(x) (x .^ 2 - 1) ./ (2 * x);
figure, hold on, box on, grid on;

epsilon = 0.01;
x_plot = linspace(0+epsilon, pi-epsilon, 1000);
plot(x_plot, f(x_plot), "b-", "LineWidth", 1);
x_plot = linspace(pi+epsilon, 2*pi-epsilon, 1000);
plot(x_plot, f(x_plot), "b-", "LineWidth", 1);
x_plot = linspace(2*pi+epsilon, 3*pi-epsilon, 1000);
plot(x_plot, f(x_plot), "b-", "LineWidth", 1);

x_plot = linspace(0+epsilon, 3*pi-epsilon, 1000);
plot(x_plot, g(x_plot), "r-", "LineWidth", 2);
```

```
% Three intervals for the roots are then [1,2], [3,4] and [6.5,7]
% 6.5 as function discontinuous!
```

Apply bisection with tolerance of 10^{-6} then plot the evolution of the error function, remind that the error function is:

$$|\alpha - x^{(k)}|$$

```
tol = 1e-6;
h = @(x) cot(x) - (x .^ 2 - 1) ./ (2 * x);
[root1, x_iter1] = bisection(h, 1, 2, tol);

% Since the exact value of the roots is NOT known, we perform
% again the bisection method with a very small tolerance, and consider
% the final approximation as the exact root of the equation.
% First root
[root1_precise, ~] = bisection(h, 1, 2, 1e-10*tol);
err = abs(root1_precise - x_iter1);
iter = numel(x_iter1);
figure, box on, hold on, grid on;
semilogy(err,"LineWidth",2);
semilogy(tol*ones(iter,1), "r--","LineWidth",2);
xlim([0 iter+1])
set(gca,"FontSize",16)
xlabel("Iteration","FontSize",16)
ylabel("Error","FontSize",16)
```

6.2.2 Fixed point method

Solve $x^2 - 5 = 0$ to find the root $\alpha = \sqrt{5}$ using the following iterative methods:

1. $x^{(k+1)} = 5 + x(k) - \left(x^{(k)}\right)^2$
2. $x^{(k+1)} = 1 + x^{(k)} - \frac{1}{5}\left(x^{(k)}\right)^2$

```
% fixed point convergent if |phi'(alpha)| < 1, if > 1 diverges, if == 1 idk
a = sqrt(5);
f = @(x) x .^ 2 - 5;

phi1 = @(x) 5 + x - x .^ 2;
% check consistency, phi(alpha) == alpha
phi1(alpha) == alpha
dphi1 = @(x) 1 - 2 * x;
abs(dphi1(a))

phi2  = @(x) 1 + x - 1/5*x.^2;
dphi2 = @(x) 1 - 2/5*x;
abs(dphi2(a))

% phi2 converges:
tol = 1e-6;
maxit = 1000;
x0 = a + 2*rand(0.001); % random initial value in [a,a+0.001]
x0 = a + 0.001; % intial guess
[x, x_iter] = fixed_point(phi2, x0, tol, maxit);
```

6.2.3 Fixed point method: scheme consistency

Given the equation:

$$f(x) = e^{x^2}\log(x+1) = 1$$

The equation is defined for $x > -1$. Consider:

$$x^{(k+1)} = \sqrt{-\log\log\left(x^{(k)}+1\right)}$$

Compute the root with tolerance of 10^{-3}.

The method is defined only if:

$$-\log\log\left(x^{(k)}+1\right) \geq 0$$

$$\log\log\left(x^{(k)}+1\right) \leq 0$$

$$0 < \log\left(x^{(k)}+1\right) \leq 1$$

$$1 < \left(x^{(k)}+1\right) \leq e$$

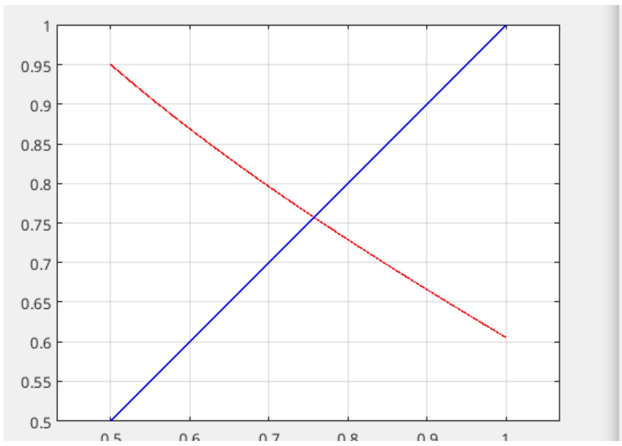
$$0 < x^{(k)} \leq e-1$$

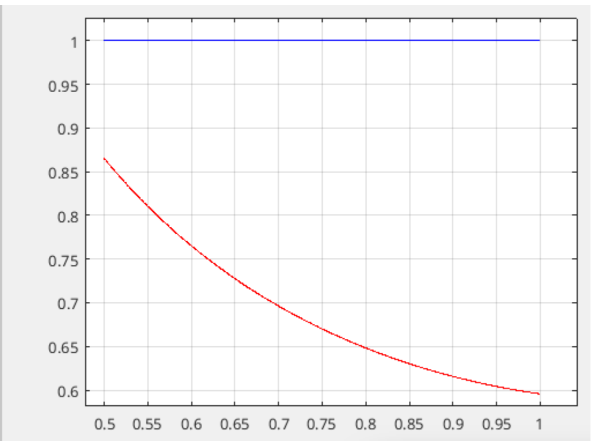
It's derivative is:

$$\frac{1}{2}\cdot\sqrt{-\log\log\left(x^{(k)}+1\right)}\cdot\left(-\frac{\frac{1}{x^{(k)}+1}}{\log\left(x^{(k)}+1\right)}\right)$$

$$\frac{1}{2\left(x^{(k)}+1\right)\log\left(x^{(k)}+1\right)\sqrt{-\log\log\left(x^{(k)}+1\right)}}$$

```
x_plot = linspace(-1+0.001,2,1000);

f = @(x) exp(x.^2).*log(x + 1) - 1;

% find the interval containing the root
figure, hold on, grid on, box on; %axis equal;
plot(x_plot,f(x_plot),"r-");
plot(x_plot,zeros(size(x_plot)),"b-");

% interval in [0.5 1]
x_plot = linspace(0.5, 1, 1000);

bisector = @(x) x;
% to check if method consistent look at the bisector/slope of derivative
% In this case in a neighborhood of the root, the absolute
% value of the derivative is less than 1
% ATTRACTOR
phi1 = @(x) sqrt(-log(log(x + 1)));
figure, hold on, grid on, box on, axis equal;
plot(x_plot,phi1(x_plot),"r-");
plot(x_plot, bisector(x_plot),"b-");

% alternatively plot the derivative
dphi1 = @(x) -1./(2*log(x + 1).*(-log(log(x + 1))).^(1/2).*(x + 1));
figure, hold on, grid on, box on, axis equal;
plot(x_plot,abs(dphi1(x_plot)),"r-");
plot(x_plot,ones(size(x_plot)),"b-");

% So here we can apply fixed point
[xi1, x1] = fixed_point(phi1, 0.9, 1e-3, 1000);
xi1
iter1 = numel(x1)
```

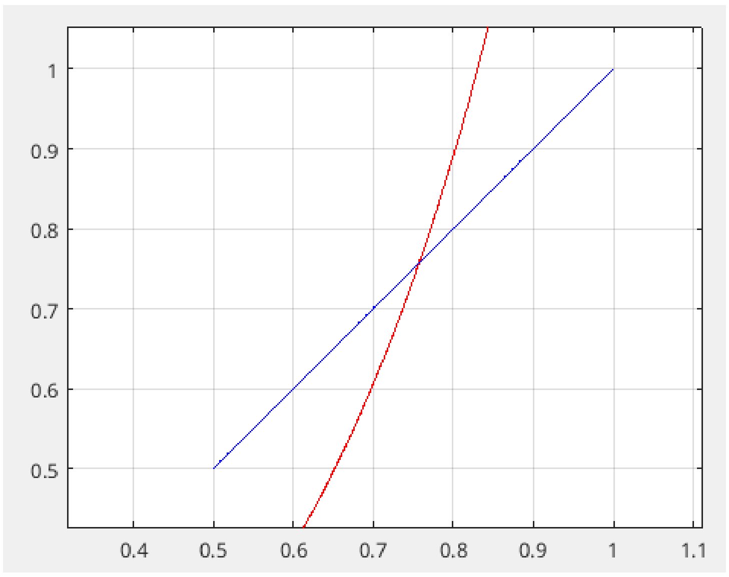




Consider this other method:

$$x^{(k+1)} = x^{(k)} e^{\left(x^{(k)}\right)^2} \log(x^{(k)} + 1)$$

```
% The comparison with the slope of the bisector shows that,
% in a neighborhood of the root, the absolute
% value of the derivative is greater than 1
% For this reason the method will not converge.
% REPULSOR
phi2 = @(x) x .* exp(x .^ 2) .* log(x + 1);
figure, hold on, grid on, box on, axis equal;
plot(x_plot, phi2(x_plot), "r-");
plot(x_plot, bisector(x_plot), "b-")
```

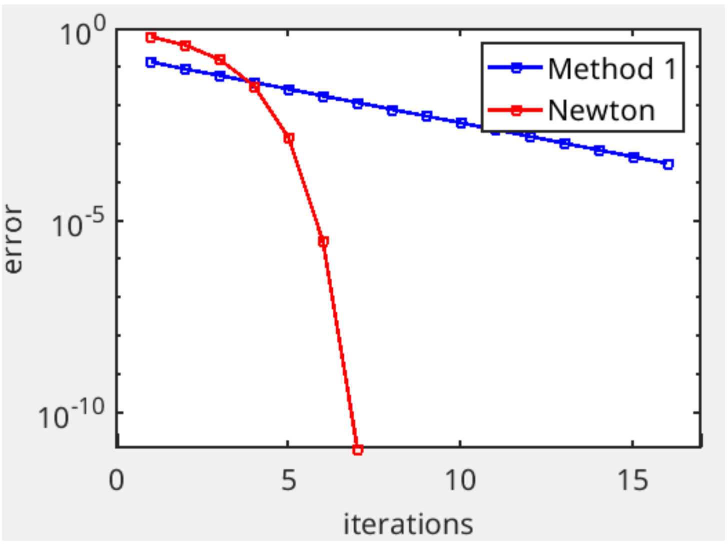


Now consider Newton method, verify the consistency of the scheme and compute the convergence orders, remid that:

$$p = \frac{\log\left[\frac{e^{k+2}}{e^{k+1}}\right]}{\log\left[\frac{e^{k+1}}{e^{k}}\right]}$$

```
phi1 = @(x) sqrt(-log(log(x + 1)));
[xi1, x_iter_fixed] = fixed_point(phi1, 0.9, 1e-3, 1000);

% The plot of f shows that f'(x) is always positive in [-1, 2], therefore the
    method is locally convergent.
df = @(x) 2.*x.*exp(x.^2).*log(x+1) + exp(x.^2)./(x+1);
```

```
[x_newton, x_iter_newton] = newton(f, df, 1.4, 1e-3, 1000);

% Rate of convergence
[xiex, xex] = newton(f, df, 1.4, 1e-12, 1000);
err1 = abs(x_iter_fixed - xiex);
err3 = abs(x_iter_newton - xiex);

% Approximate the convergence order
p1 = diff( log(err1(2:end) ) ) ./ diff( log(err1(1:end-1) ) )
p3 = diff( log(err3(2:end) ) ) ./ diff( log(err3(1:end-1) ) )

% Or compute normally
p1 = log(err1(end)/err1(end-1))./log(err1(end-1)/err1(end-2))
p3 = log(err3(end)/err3(end-1))./log(err3(end-1)/err3(end-2))

% The order of the first method is 1, Newton method is as expected of second
    order.
figure, hold on, box on;
semilogy(err1, 'bs-',"LineWidth",2)
semilogy(err3, 'rs-',"LineWidth",2)
set(gca,"LineWidth",1.5)
set(gca,"FontSize",16)
xlim([0 numel(err1)+1])
xlabel("iterations","FontSize",16)
ylabel("error","FontSize",16)
h = legend('Method 1','Newton');
set(h,"FontSize",16)
```



6.3 Systems of Linear Equations: Direct Methods

6.3.1 Solve with LU

Solve the system $Ax = b$ and compute the determinant of A

```
A = [2 10 4 0;
     1  0 2 2;
     1 4 0 2;
     1 2 1 1];
b = [10 1 3 3]';

[L, U] = lu(A);
% In this case it is not essential to include the matrix P among the output
% variables in order to have L in the form of a lower triangular matrix
% with L_ii = 1 for each i.

y = L \ b; %forward
x = U \ y; %backward

% or use
y = forward_substitution(L, b)
x = backward_substitution(U, b)

% Check
A \ b

% Thanks to the Binet-Cauchy formula, we have, for generic square matrices,
% det(A*B) = det(A)*det(B).
% In our case, since A = L*U, det(A) = det(L*U) = det(L)*det(U).
% However L_ii = 1 for each i, so det(L) = 1
% => det(A) = det(U).

detA = prod(diag(U))

% Check with the command det
det(A)
```

Solve now the system with a new *b*

```
b_new = [-22 -12 -1 -11]';

% we don't need to recompute the LU factorization
y2 = forward_substitution(L, c);
x2 = backward_substitution(U, y2);
```

6.3.2 Custom inverse

Write a function that computes the inverse of a generic square matrix

```
function [InvA] = MyInv(A)
    [L,U] = lu(A);
    %s = size(A);
    %n = s(1);
    n = length(A);
    for k = 1:n
        e = zeros(n,1);
        e(k) = 1;
        y = forward_substitution(L,e);
        InvA(:,k) = backward_substitution(U,y);
    end
end
```

6.3.3 LU conditions and pivoting

Consider matrix E with b:

$$E = \begin{bmatrix} 4 & 1 & 1 & 1 & 5 \\ 4 & 1 & 2 & 0 & 0 \\ 1 & 0 & 15 & 5 & 1 \\ 0 & 2 & 4 & 10 & 2 \\ 3 & 1 & 2 & 4 & 20 \end{bmatrix} \qquad A = \begin{bmatrix} 12 \\ 19 \\ 22 \\ 18 \\ 30 \end{bmatrix}$$

Verify that the sufficient conditions for the existence and uniqueness of the LU decomposition without pivoting are NOT fulfilled by E

```
E = [4 1 1 1 5;
    4 1 2 0 0;
    1 0 15 5 1;
    0 2 4 10 2;
    3 1 2 4 20];

b = [12 19 22 18 30]';

n = length(E(1,:));

% Let calculate the determinant of the all the minor fo the matrix E

for i = 1:n-1
    det(E(1:i, 1:i))
end

% Since one of the minor determinants is equal to zero, the LU
% factorization without pivoting does not exist!

% ATTENTION
% if a matrix E does not admit a LU factorization and you
% call lu command without the output P, matlab does not
% return the lu factorization of P*E but a factorization
% of E which is not lower-upper triangular
[L, U] = lu(E);
detA = prod(diag(U))
L % is not triangular!
spy(L)
```

Compute the LU and verify if pivoting takes place, then solve the system

```
[L, U, P] = lu(E);
detA = prod(diag(U))
spy(L)
P % is not the identity matrix => pivoting is needed

y = forward_substitution(L,P*b); %L\(P*b);
x = backward_substitution(U,y); %U\y

x2 = E \ b

x == x2 % precision errors, but should be correct
```

6.3.4 Cholesky decomposition

Verify that the Cholesky decomposition can be applied to matrix A (is spd), then solve the system

```
A = [44 15 29 26 119;
    15 33 32 18 15;
    29 32 252 112 73;
    26 18 112 124 90;
    119 15 73 90 430];
b = [1 1 1 1 1]';

% A must be spd:
% - all eigenvalues are real (s)
% - all eigenvalues are > 0 (pd)
eigenvalues = eig(A);
isreal(eigenvalues);
all(eigenvalues > 0);
```

```
R = chol(A);

y = forward_substitution(R',b); % R'\b
x = backward_substitution(R,y); % R\y
```

6.3.5 Sparse matrix and Thomas algorithm

Consider the tridiagonal matrix $A \in \mathbb{R}^{10\times 10}$

$$A = \begin{bmatrix} 1 & 11 & & & \\ 102 & 2 & 12 & & \\ & 103 & 3 & 13 & \\ & \cdots & \cdots & \cdots & \\ & & 109 & 9 & 19 \\ & & & 110 & 10 \end{bmatrix}$$

Then consider the linear system $Ax = b$ such that $x = [1 \cdots 1] \in \mathbb{R}^{10\times 1}$. Store the matrix in a sparse format and check the solution of the linear system with Thomas

```
a = [1:10]';
c = [10:19]';
e = [102:111]';
n = length(a);
A_sparse = spdiags([e a c], [-1 0 1], n, n)
% insert into diagonal (0), just below (-1) and just on top(1)
% c on top will have first element (10) deleted
% e below will have last element (111) deleted

% alternative way to create A as a sparse and full matrix
% A = diag(1:10) + diag(102:110,-1) + diag(11:19, 1)
% Asparse = sparse(A)
x = ones(n,1);
b = A_sparse*x

[L, U, x_test] = thomas(A_sparse, b);
A_sparse \ b
x_test
```

6.3.6 Condition number

Consider the linear system $Ax = b$:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \varepsilon \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Compute the condition number for $p = 1, 2, \infty$, the perturbation on the solution with $\delta b = \begin{bmatrix} 0 & 0 & \alpha \end{bmatrix}^T$ and verify the perturbation relation with the infinity norm

$$\frac{||\delta x||}{||x||} \leq \underbrace{\sqrt{\frac{\lambda_{max}(A^T A)}{\lambda_{min}(A^T A)}}}_{K(A) \geq 1 \text{ condition number}} \frac{||\delta b|}{||b||}$$

$$\varepsilon = 10^{-6} \qquad \alpha = 10^{-12},\ 10^{-6}$$

```
epsilon = rand() / 1000;
epsilon = 1e-6;

for alpha = [1e-6 1e-12]
    A = eye(3);
    A(3, 3) = epsilon;

    b = [1 0 0]';

    % exact solution
    x = A \ b;

    % condition number
    % Since A is symmetric, norm-1 == norm-inf
    K_1 = cond(A, 1)
    cond(A, 2);
    cond(A);
    K_inf = cond(A, 'inf')

    % associated solution with perturbation on b
    delta_b = [0 0 alpha]';
    delta_x = A \ delta_b;

    % verify inequality on condition number:
    norm(delta_x,inf) / norm(x,inf) <= K_inf * norm(delta_b,inf) / norm(b,inf)
end
```

6.3.7 Condition number by trials

Consider the Hilbert matrix of order $n = 10, 20, 40$, estimate $K_2(H_n)$ using the technique of perturbating a linear system.

```
for n = [10,20,40]
    H = hilb(n)
    x = ones(n, 1);
    b = H * x;
    % condition number by trials
    pert_mag = 1e-6; % Perturbation magnitude
    n_trials = 100;
    KH_est = 0;

    for ii = 1:n_trials
        deltab = randn(n, 1) * pert_mag;
        deltax = H \ deltab;
        KH_est_ii = norm(deltax) / norm(x) * norm(b) / norm(deltab);
        KH_est = max(KH_est_ii, KH_est);
    end
    K_2 = cond(H)
end
% The estimates obtained are lower. This is expected, since
% the condition number represents the maximum
% possible amplification of the relative error on data. With the first
% approach we might not have considered the worst condition,
% hence the estimate is lower than the result with cond command.

KH(ii) = cond(H);
err(ii) = norm(x - x_ex) / norm(x);
```

6.4 Systems of Linear Equations: Iterative Methods

6.4.1 Stationary Richardson schemes: Jacobi and Gauss-Seidel

Reminder, we want

$$x^{(k+1)} = \underbrace{\alpha_k P^{-1} b}_{g_{\alpha_k}} + \underbrace{\left(I - \alpha_k P^{-1} A\right)}_{B_{\alpha_k}} x^{(k)}$$

$$A = D - E + (A - (D - E))$$

With E lower triangular part, D diagonal of A and $\alpha = 1$

Jacobi

$$x^{(k+1)} = \underbrace{(I - D^{-1}A)}_{B_J} x^{(k)} + \underbrace{D^{-1}b}_{g_J}$$

Gauss-Seidel

$$x^{(k+1)} = \underbrace{(I - (D-E)^{-1}A)}_{B_{GS}} x^{(k)} + \underbrace{(D-E)^{-1}b}_{g_{GS}}$$

Study the convergece for Jacobi, then apply stationary method to check the result with $x_0 = [0,0,0]^T$, tol=10^{-6} and maxit=100

```
A = [
    2 1 -2;
    1 2 1;
    2 1 2
    ];
x = [1 2 3]';
b = inv(A) \ x;

% JACOBI
% A = D-E+(A-(D-E))
% where D=diagonal part of A=diag(A)
% where E lower triagular part
% B = inv(D)*(D-A)
% g = inv(D)*b
% BUT WE CANNOT USE inv(), but as D is diagonal, inv(D) = 1/D

D = diag(diag(A));
B_j = D \ (D - A);
g_j = D \ b;

% matrix are not particular (not spd), to check convergence
% compute the 2norm of iteration matrices
normB_j = norm(B_j);

% If bigger than 1, compute spectral radius
rhoB_j = max(abs(eig(B_j)))

% If a spectral norm > 1, the method will not converge
% Finally solve the system
x0 = [0 0 0]';
tol = 1.e-6;
max_it = 100;
[x, iter, incr] = stationary_method(B_j, g_j, x0, tol, max_it);

x1j = B_j*x0 + g_j;
k_j_min = log(tol*(1 - normB_j)/norm(x-x0)) / log(norm_Bj);
```

Study the convergece for Gauss-Seidel, then apply stationary method to check the result with $x_0 = [0,0,0]^T$, tol=10^{-6} and maxit=100

```
% GAUSS-SEIDEL
% g = inv(D-E)*b
% How to find inv(D-E) without inv()?
% If we have inv(A)w=z, we can both multiply by A
% A*inv(A)w = A*z -> w=Az, so z = A\b!!!
% So for use g = (D-E)\b
% Same holds for matrices, inv(A)*B = C -> C = A\B

E = -tril(A, -1);
% -1 will put diagonal at 0 and uppertriangular 0
% L = tril(A, -1);
% U = triu(A, 1);

B_gs = (D - E) \ (D - E - A);
g_gs = (D - E) \ b;

% matrix are not particular (not spd), to check convergence
% compute the 2norm of iteration matrices

normB_gs = norm(B_gs);

% If bigger than 1, compute spectral radius
rhoB_gs = max(abs(eig(B_gs)));

% If a spectral norm > 1, the method will not converge
% Finally solve the system
x0 = [0 0 0]';
tol = 1.e-6;
max_it = 100;
[x, iter, incr] = stationary_method(B_gs, g_gs, x0, tol, max_it);

x1gs = B_gs*x0 + g_gs;
k_gs_min= log(tol*(1 - normB_gs)/norm(x1gs-x0)) / log(normB_gs);
```

6.4.2 Jacobi iterations

Using Jacobi, compute how many iterations are required to ensure that the infinity norm of the error is less than 10^{-9}

```
n = 3;
alpha = 1e-4;
A = eye(n);
A(1, n) = 1 / alpha;

D = diag(diag(A));

B = D \ (D - A);
b = ones(n, 1);
g = D \ b;

% check if Jacobi will converge or not
norm(B)
max(abs(eig(B)))

% infinity norm on the error ~ check increment
iter = 0;
x = b;
dx = b;
while norm(dx, inf) >= 1e-9
    x_old = x;
    x = B * x + g;
    dx = x - x_old;
    iter = iter + 1;
end
```

```
iter
```

6.4.3 Choice of α of the Richardson method

Remind that:

$$B = (I - \alpha P^{-1}A) \qquad \alpha_{opt} = \frac{2}{\lambda_{\max}(P^{-1}A) + \lambda_{\min}(P^{-1}A)}$$

Check if matrices A and T are spd, then apply Richardson first with $\alpha = 0.2, 0.33$ with $P = I$ then with α_{opt} with $P = T$

```
n = 50;
diagonal = 4 * ones(n, 1);
minus = -ones(n, 1);
A = spdiags([minus minus diagonal minus minus], [-2 -1 0 1 2], n, n);
A = full(A);
T = spdiags([minus minus * -2 minus], [-1 0 1], n, n);
T = full(T);
b = 0.2 * ones(n, 1);

% spd? yes
isreal(eig(A))
all(eig(A) > 0)

isreal(eig(T))
all(eig(T) > 0)

% converge, check if spectral radius of B < 1
% with P = I -> B = I-alpha*I^-1*A
tol = 1e-6;
x0 = zeros(n, 1);
P = eye(n);
maxit = 10000;

for alpha = [0.2 0.33]
    B = eye(n) - alpha * (P \ A);
    if max(abs(eig(B))) < 1
        disp(["Converges for ", alpha])
        [x, iter, incr] = prec_rich_method(A, b, P, alpha, x0, tol, maxit);
        x
        iter
    end
end

% alpha_opt
P = T;
eig_max = max(eig(P \ A));
eig_min = min(eig(P \ A));
alpha_opt = 2 / (eig_max + eig_min);
B = eye(n) - alpha_opt * (P \ A);
max(abs(eig(B)))
[x, iter, incr] = prec_rich_method(A, b, P, alpha_opt, x0, tol, maxit);
x
iter
```

6.5 Approximation of Functions and Data

6.5.1 Equally spaced nodes: Lagrange, piecewise and cubic spline

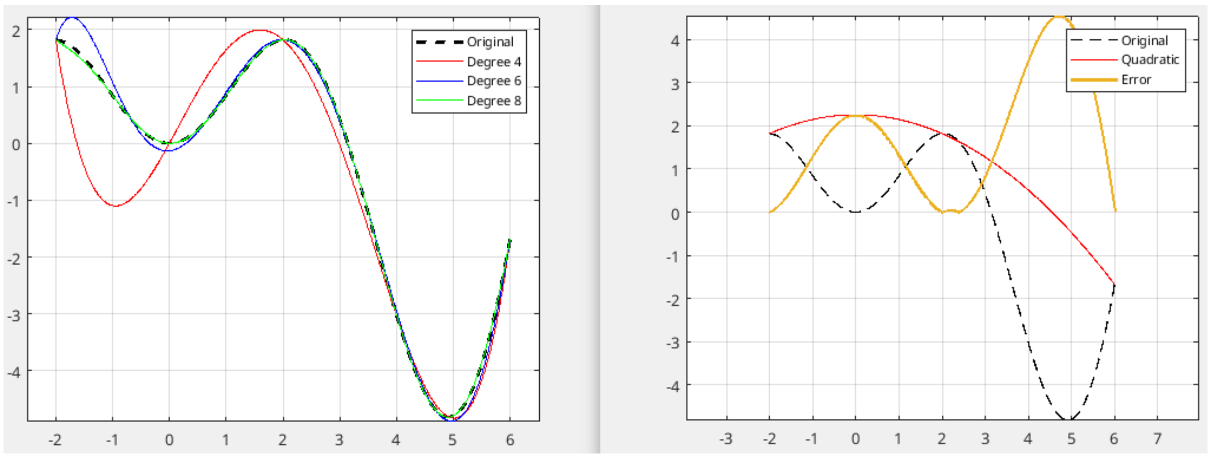
Approximate the function $f(x) = x\sin(x)$ in the interval $[-2,6]$. Consider a set of equally spaced nodes: find the interpolating polynomial using Lagrange basis polynomials of degree 4, 6, 8. Then estimate the interpolation error when resorting to quadratic interpolation ($n = 2$). Evaluate the interpolating polynomial in 1000 equally spaced points. Remind that the interpolating error:

$$h = \frac{x_n - x_0}{n}$$

$$\max_{x\in I} |E_n f(x)| \leq \underbrace{\frac{h^{n+1}}{4(n+1)}}_{\text{A}} \cdot \underbrace{\max_{x\in I} \left| f^{(n+1)}(x) \right|}_{\text{B}}$$

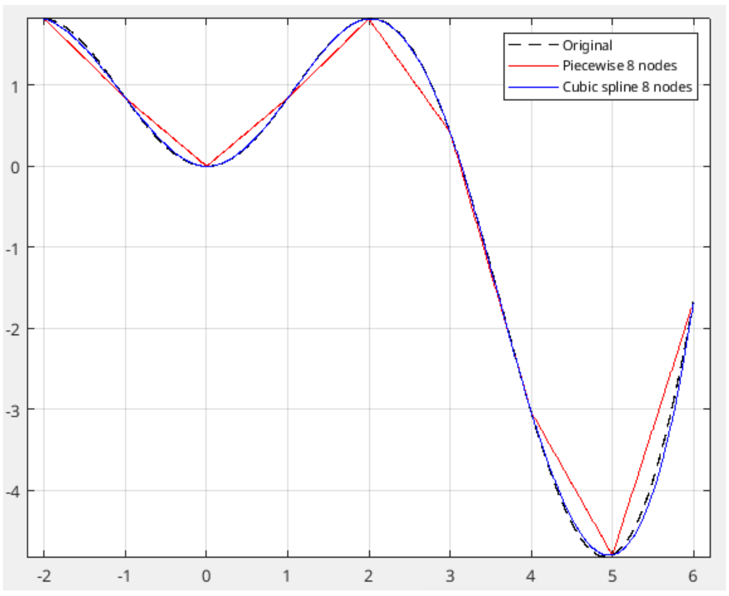
```
f = @(x) x .* sin(x);
x_plot = linspace(-2, 6, 1000);

figure, hold on, box on, grid on, axis equal;
plot(x_plot, f(x_plot), 'k--', 'Linewidth', 2);

for n = [4 6 8]
    num_nodes = n + 1;
    x_nodes = linspace(-2, 6, num_nodes);
    y_nodes = f(x_nodes);
    coeff = polyfit(x_nodes, y_nodes, n);
    interp_plot = polyval(coeff, x_plot);
    if n == 4
        plot(x_plot, interp_plot, 'r-');
    end
    if n == 6
        plot(x_plot, interp_plot, 'b-');
    end
    if n == 8
        plot(x_plot, interp_plot, 'g-');
    end
end

h = legend("Original", "Degree 4", "Degree 6", "Degree 8");
set(h, "FontSize", 12);

% interpolation error when resorting to a quadratic interpolation, n=2
% which means n+1=3 num_nodes
n = 2;
x_nodes = linspace(-2, 6, n + 1);
y_nodes = f(x_nodes);
coeff = polyfit(x_nodes, y_nodes, n);
interp_plot = polyval(coeff, x_plot);
figure, hold on, box on, grid on, axis equal, xlim auto, ylim auto;

plot(x_plot, f(x_plot), 'k--', 'Linewidth', 1);
plot(x_plot, interp_plot, 'r');

h = 6 - (-2) / n;

% compute third derivative
f_1 = @(x) sin(x) + x .* cos(x);
f_2 = @(x) 2 * cos(x) - x .* sin(x);
f_3 = @(x) -2 * sin(x) - sin(x) + x .* cos(x);

err = (h ^ (n + 1) / 4 * (n + 1)) * max(f_3(x_plot));
```

```
plot(x_plot, abs(f(x_plot) - interp_plot), 'Linewidth', 2);
h = legend("Original", "Quadratic", "Error");
set(h, "FontSize", 12);
```



Now compute the piecewise linear interpolant and the cubic spline with $n+1=6,11,21$ equally spaced nodes

```
% Piecewise linear interpolant and cubic spline with n+1=6,9,21 equally space
    nodes
for n = [5 8 20]
    figure, hold on, box on, grid on, axis equal, xlim auto, ylim auto;
    plot(x_plot, f(x_plot), 'k--', 'Linewidth', 1);

    num_nodes = n + 1;
    x_nodes = linspace(-2, 6, num_nodes);
    y_nodes = f(x_nodes);

    % (x_nodes, y_nodes) pairs, directly evaluate on x_plot
    piecewise_plot = interp1(x_nodes, y_nodes, x_plot);
    spline_plot = spline(x_nodes, y_nodes, x_plot);

    plot(x_plot, piecewise_plot, 'r-');
    plot(x_plot, spline_plot, 'b-');
    legend_1 = strcat("Piecewise ", num2str(n), " nodes");
    legend_2 = strcat("Cubic spline ", num2str(n), " nodes");
    h = legend("Original", legend_1, legend_2);
    set(h, "FontSize", 8);
end
```



6.5.2 Runge, Chebyshev nodes and convergence order

Consider the function:

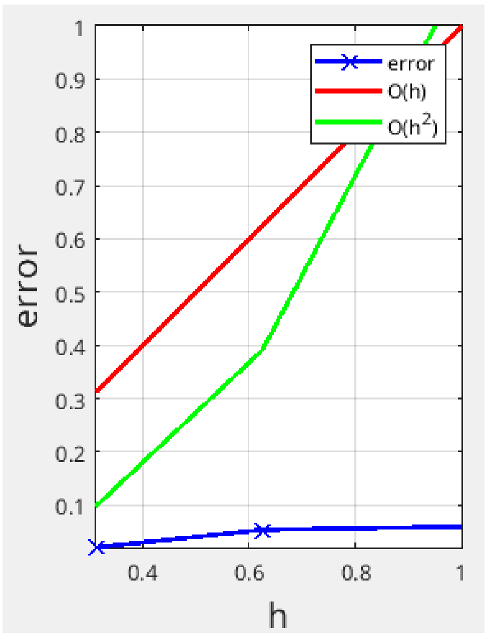
$$f(x) = \frac{1}{1+x^2} \qquad -5 \leq x \leq 5$$

Verify the Runge phenomenon

```
a = -5;
b = 5;
f = @(x) 1 ./ (1 + x .^ 2);
x_plot = linspace(a, b, 1000);
figure, hold on, grid on, axis equal, box on;
plot(x_plot, f(x_plot), 'k--');

% plot with n=2:2:10
for n = 2:2:10
    x_nodes = linspace(a, b, n + 1);
    y_nodes = f(x_nodes);
    coeff = polyfit(x_nodes, y_nodes, n);
    interp_plot = polyval(coeff, x_plot);
    plot(x_plot, interp_plot, '"r-");

    % Runge phenomenon, the max of n-derivative of f
    % goes to +inf quicker than A goes to 0
    n
    norm(f(x_plot) - interp_plot, inf)
end
```

Compute the piecewise linear interpolation with $n = 1, 2, 4, 8, 16, 32$ subintervals and plot the result. Find the error w.r.t. the infinity norm and verify that it converges quadratically as a function of the distance between two nodes

```
% piecewise linear
exp = 0:5;
n_vect = (2 * ones(1, exp(end) + 1)) .^ exp;

for i = 1:numel(n_vect)
    n = n_vect(i);
    x_nodes = linspace(a, b, n + 1);
    y_nodes = f(x_nodes);
    piece_plot = interp1(x_nodes, y_nodes, x_plot);
    % figure, hold on, grid on, axis equal, box on;
    % plot(x_plot, f(x_plot), "k--");
    % plot(x_plot, piece_plot, "r-");
    error(i) = norm(f(x_plot) - piece_plot, inf);
end

% determination of the element width
H = (b - a) ./ n_vect;
figure, hold on, grid on, axis equal, box on;
loglog(H, error, "bx-", "LineWidth", 2, "MarkerSize", 8);
loglog(H, H, "r-", "LineWidth", 2);
loglog(H, H .^ (2), "g-", "LineWidth", 2);
xlabel("h", "FontSize", 16)
ylabel("error", "FontSize", 16)
legend("error", "O(h)", "O(h^2)");
xlim([min(H), 1]);
ylim([min(error), 1]);
% THIS LEGEND REQUIRED
% to understand behavior at infinite we have to increase the number of
    subinterval
% increase n_vect [1 2 4 8 16 32 64 128];
% look at the line that at the BEGINNING is ALMOST parallel to the error line

% compute convergence order with:
order = (log(error(1:end - 1) ./ error(2:end)) / log(2))'
% or using the diff command build in in matlab
p = -diff(log(error)) / log(2)
```



Compute how many equally spaced nodes are needed in order to ensure an interpolation error with infinity norm smaller than 10^{-3}. Remind that for piecewise:

$$\max_{x\in I} \left|E_1^H f(x)\right| \leq \frac{H^2}{8} \cdot \max_{x\in I} \left|f''(x)\right| \qquad f \in C^2(\bar{I})$$

```
% or use the formula
df1 = ...;
df2 = ...;
max_val = max(df2(x_plot));
H = sqrt(8 * 1e-3 / max_val);
```

```
n = ceil((b - a) / H); % degree
n + 1
```

Compute $\Pi_n f(x)$ using Chebyshev nodes:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\hat{x}_i \qquad \hat{x}_i = -\cos(\pi i/n),\ i = 0,\cdots,n$$

```
% Chebyshev nodes
exp = 0:5;
n_vect = (2 * ones(1, exp(end) + 1)) .^ exp;

for i = 1:numel(n_vect)
    n = n_vect(i);

    % Calculation of the Chebyshev nodes
    indexes = 0:n;
    x_hat = -cos(pi * indexes / n);
    x_nodes = (a + b) / 2 + (b - a) / 2 .* x_hat;
    y_nodes = f(x_nodes);

    coeff = polyfit(x_nodes, y_nodes, n);
    chebyshev_plot = polyval(coeff, x_plot);
    % figure, hold on, grid on, axis equal, box on;
    % plot(x_plot, f(x_plot), "k--");
    % plot(x_plot, chebyshev_plot, "r-");
    error(i) = norm(f(x_plot) - chebyshev_plot, inf);
end

H = (b - a) ./ n_vect;
% A second order convergence cannot be expected, since the function is not
   smooth enough.
% Calculation of the error
order = (log(error(1:end - 1) ./ error(2:end)) / log(2))'
% or using the diff command build in in matlab
p = -diff(log(error)) / log(2)
```

6.5.3 Least squared approximation

Remind that degree $m << n$, if $m == n$ we have Lagrange.

```
a = -1;
b = 1;
f = @(x) abs(x - pi / 12);
x_plot = linspace(a, b, 1000);
figure, hold on, grid on, axis equal, box on;
plot(x_plot, f(x_plot), 'k--');

% runge
for n = 2:2:10
    x_nodes = linspace(a, b, n + 1);
    y_nodes = f(x_nodes);
    coeff = polyfit(x_nodes, y_nodes, n);
    poly_plot = polyval(coeff, x_plot);
    plot(x_plot, poly_plot, "r-");
end

% use least squares approach with number of nodes = 10
% and varying degree of interpolant
x_nodes = linspace(a, b, 10);
y_nodes = f(x_nodes);
```

```
figure, hold on, grid on, axis equal, box on;
plot(x_nodes, y_nodes, 'ro-', 'Linewidth', 2);

% to use least squares, if we have n+1 nodes,
% the least square with degree n == Lagrange interpolant
% differently from before, #nodes does not vary
for deg = 2:1:9
    coeff = polyfit(x_nodes, y_nodes, deg);
    least_square_plot = polyval(coeff, x_plot);
    plot(x_plot, least_square_plot, "b-");
end
```

6.5.4 Numerical integration

Consider the following integrals:

$$I_1 = \int_0^1 x^3 dx = \frac{1}{4} \qquad I_2 = \int_0^1 x^5 dx = \frac{1}{6}$$

Compute the integral I_1 using midpoint with 1 and 10 nodes, composite trapezoidal rule with 2 and 10 nodes and composite Simpson with 3 and 7 nodes

```
f = @(x) x .^ 3;
midpoint(f, 0, 1)
composite_midpoint(f, 0, 1, 1)
composite_midpoint(f, 0, 1, 10)

% The function to be integrated is a polynomial of order 3. Hence the rule,
% of degree 1, is not able to compute the integral exactly.
% The error decreases while increasing the number of nodes
% (i.e., increasing the number of subintervals)

% trapezoidal with 2 and 10 nodes
composite_trapezoidal(f, 0, 1, 1)
composite_trapezoidal(f, 0, 1, 9)

% Same behaviour as before, since the trapezoidal rule is of degree 1.

% Simpson with 3 and 7 nodes
composite_simpson(f, 0, 1, (3 - 1) / 2)
composite_simpson(f, 0, 1, (7 - 1) / 2)

% The result does not depend on the number of nodes.
% The reason of this is the fact that Simpson rule has degree of exactness
% equal to 3 and the integral on any subinterval is computed exactly!
```

Estimate the number of nodes needed to compute I_2 with a tolerance of 10^{-3} using the composite trapezoidal and Simpson rule

```
% compute #nodes with that tolerance using composite trapezoidal and simpson
% H = (b-a)/M

df2 = @(x) 5 * 4 * x .^ 3;
x_plot = linspace(0, 1, 1000);

% ignore negative sign
h_star = sqrt(tol * 12 / ((b - a) * (max(df2(x_plot)))))
m_star = ceil((b - a) / h_star); % number of subintervals
m_star + 1 % number of nodes

df4 = @(x) 5 * 4 * 3 * 2 * x;
```

```
h_star = (180 * tol / ((b - a) * max(df4(x_plot)))) ^ (1/4)
m_star = ceil((b - a) / h_star);
2*m_star + 1
```

Approximate the order of accuracy of the three composite methods

```
% approximate order of accuracy, either use diff or plot
for i = 0:9
    m = 2 ^ i;
    integr_m(i + 1) = composite_midpoint(f, a, b, m);
    integr_t(i + 1) = composite_trapezoidal(f, a, b, m);
    integr_s(i + 1) = composite_simpson(f, a, b, m);
end

err_m = abs(1/6 - integr_m);
err_t = abs(1/6 - integr_t);
err_s = abs(1/6 - integr_s);

p_m = -diff(log(err_m)) / log(2)
p_t = -diff(log(err_t)) / log(2)
p_s = -diff(log(err_s)) / log(2)

% or plot, e.g. for simpson
H = (b - a) ./ 2 .^ [0:9];
figure, hold on, box on, grid on;
loglog(H, err_s, "bx-", "Linewidth", 2);
loglog(H, H, "r-");
loglog(H, H .^ 2, "g-");
loglog(H, H .^ 4, "k-");
xlabel("h", "FontSize", 16);
ylabel("error", "FontSize", 16);
legend("error", "O(h)", "O(h^2)", "O(h^4)");
```

6.5.5 Compute degree of exactness

Remind that to compute the degree of exactness we must verify for polynomials till degree n in which $I(p_n)! = \tilde{I}(p_n)$. The polynomial p is any candidate, we choose the simplest ones. Consider the integral

$$I = \int_{-1}^{1} f(x)dx$$

And the following quadrature formulas:

$$Q_1 = \frac{2}{3}\left[2f\left(-\frac{1}{2}\right) - f(0) + 2f\left(\frac{1}{2}\right)\right]$$

$$Q_2 = \frac{1}{4}\left[f(-1) + 3f\left(-\frac{1}{3}\right) + 3f\left(\frac{1}{3}\right) + f(1)\right]$$

Compute the degree of exactness of these formulas

```
a = -1;
b = 1;
weights_1 = 2/3 * [2 -1 2];
nodes_1 = [-1/2 0 1/2];

for i = 0:10
    f = @(x) x .^ i;
    F = @(x) x .^ (i + 1) ./ (i + 1);

    if F(b) - F(a) == sum(weights_1 .* f(nodes_1))
```

```
        disp(["At least ", i]);
    else
        break
    end

end

de_1 = i - 1

weights_2 = 1/4 * [1 3 3 1];
nodes_2 = [-1 -1/3 1/3 1];

for i = 0:10
    f = @(x) x .^ i;
    F = @(x) x .^ (i + 1) ./ (i + 1);

    if F(b) - F(a) == sum(weights_2 .* f(nodes_2))
        disp(["At least ", i]);
    else
        break
    end

end

de_2 = i - 1

% Both formulae are exact up to degree 3.
```

Use the quadrature formulas above to approximate the integral

$$\int_1^3 \log(x)dx$$

```
% integrate
f = @(x) log(x);
% The integral can be rewritten as
% int_{-1}^{1} log(t+2) dt;
% we then apply the quadrature rule.
f = @(t) log(t + 2);
Q1 = sum(weights_1 .* f(nodes_1));
Q2 = sum(weights_2 .* f(nodes_2));
F = @(x) x * log(x) - x;
int_exact = F(3) - F(1)
err_Q1 = abs(int_exact - Q1)
err_Q2 = abs(int_exact - Q2)

% The second rule behaves better.
```

6.5.6 Numerical integration error

Consider the integral:

$$I = \int_0^1 x^\alpha dx$$

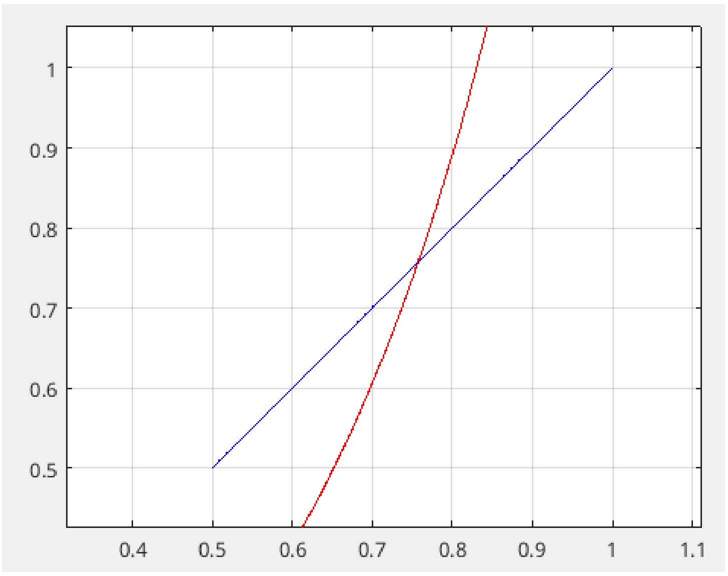
With $\alpha = 1/2, 3/2, 5/2, 7/2$. Consider the composite midpoint, trapezoidal and Simpson rules. For each value of α check if the order of accuracy predicted by the theory of each quadrature formula is obtained and, if not, motivate the different behavior

```
a = 0;
b = 1;
```

```
for alpha = [1/2 3/2 5/2 7/2]
    f = @(x) (x .^ alpha);
    % Use an increasing number of nodes in computing the integral
    for i = 0:9
        m = 2 ^ i;
        integr_m(i + 1) = composite_midpoint(f, a, b, m);
        integr_t(i + 1) = composite_trapezoidal(f, a, b, m);
        integr_s(i + 1) = composite_simpson(f, a, b, m);
    end

    int_exact = quad(f, a, b, 1e-12);

    err_m = abs(int_exact - integr_m);
    err_t = abs(int_exact - integr_t);
    err_s = abs(int_exact - integr_s);

    p_m = -diff(log(err_m)) / log(2)
    p_t = -diff(log(err_t)) / log(2)
    p_s = -diff(log(err_s)) / log(2)

end

% oa of simple and trapezoidal should be 2, for simpson is 4
% Depending on the considered value of alpha the integrand function
% is characterized by a different level of regularity.
% With alpha = 1/2, all the methods show convergence of order 3/2 (= alpha + 1)
% The regularity of the function is too low for the methods to reach their
% maximum theoretical convergence order.
% With alpha = 3/2 the error decreases quadratically in h for both midpoint
% and trapezoidal rules.
% For the Simpson rule instead, convergence is of order 5/2 (= alpha + 1).
% Again regularity is not enough for the Simpson method to reach fourth order.
% Similar situation occurs with alpha = 5/2, with Simpson rule showing
% order 7/2 (= alpha + 1).
% Finally, with alpha = 7/2 all the methods show their theoretical orders.
```

6.5.7 Numerical integration

```
plot(x_plot, bisector(x_plot), "b-")
```


1
0.9
0.8
0.7
0.6
0.5
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1