# Numerical Analysis and Computational Mathematics

MATH–456

CSE — Computational Science and Engineering, EPFL

A.Y. 2015/16

Dr. Luca Dede'

The content of these lecture notes is mainly taken and elaborated from [2] and [1].
Written with LaTeX, based on "The Legrand Orange Book LaTeXTemplate"
*Lausanne, December 15, 2015*

# Contents

# 1. Introduction to Numerical Analysis

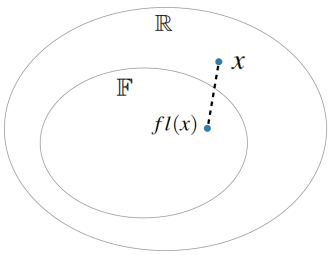
## 1.1 Machine Representation of Real Numbers

The computer can only represent and operate with a finite set of real numbers $x \in \mathbb{R}$.

### 1.1.1 Floating point numbers

**Definition 1.1** The set of *floating point numbers* $\mathbb{F}$ is the subset of real numbers which can be represented at the computer, i.e. $\mathbb{F} \subset \mathbb{R}$, with $\dim(\mathbb{F}) < +\infty$. In general, $\mathbb{F} = \mathbb{F}_0 \cup \{0\}$, with $\mathbb{F}_0$ the floating point numbers excluding the zero.

We indicate with $fl(x)$ the *floating point representation* of the real number $x \in \mathbb{R}$.




♦ **Example 1.1** $x = \frac{1}{3} = 0.\overline{3} = 0.\underbrace{333\cdots3}_{\infty \text{ digits}}$, while $fl(x) = 0.\underbrace{333\cdots3}_{N \text{ digits}}$, with $N < +\infty$. ♦

The set $\mathbb{F}_0 = \mathbb{F}_0(\beta, t, L, U)$ is characterized by four parameters $\beta$, $t$, $L$, and $U$ such that any real number $x \in \mathbb{F}_0$ can be written as:

$$x = (-1)^s \, m \beta^{e-t} = (-1)^s \, (a_1 a_2 \cdots a_t)_\beta \, \beta^{e-t},$$

where:

- $\beta$ is the *base* (the numerical system);
- $m = (a_1 a_2 \cdots a_t)_\beta$ is the *mantissa* with $t$ *number of digits* such that $0 < a_1 \leq \beta - 1$ and $0 \leq a_i \leq \beta - 1$ for $i = 2, \ldots, t$;
- $e \in \mathbb{Z}$ is the *exponent* such that $L \leq e \leq U$, with $L < 0$ and $U > 0$;
- $s = \{0, 1\}$ is the *sign*.

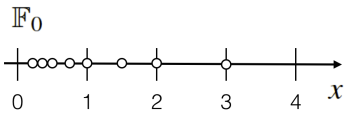
Once $\mathbb{F}_0(\beta,t,L,U)$ is characterized, $x \in \mathbb{F}_0$ is fully represented by $s$, $m$, and $e$. The minimum and maximum positive real numbers which can be represented at the calculator are $x_{min} = \beta^{L-1}$ and $x_{max} = \beta^U\left(1-\beta^{-t}\right)$, respectively.

**Definition 1.2** The *epsilon machine* $\varepsilon_M := \beta^{1-t}$ is the minimum real number greater than zero such that $fl(1+\varepsilon_M) > 1$. The *roundoff error* $\frac{1}{2}\varepsilon_M$ is an upper bound of the relative error in the representation of a real number $x \in \mathbb{R}\setminus\{0\}$, i.e.:

$$\frac{|x-fl(x)|}{|x|} \leq \frac{1}{2}\varepsilon_M.$$

**Remark 1.1** Even if the roundoff error $\frac{1}{2}\varepsilon_M$ is "small", i.e. the relative error is "small", the absolute error $|x-fl(x)|$ may be very "large", especially if $|x|$ is "large".

♦ **Example 1.2** We consider the set of floating point numbers $\mathbb{F}_0(2,2,-1,2)$, i.e. with $\beta = 2$ (numerical system in base 2), $t = 2$ (number of digits), $L = -1$, and $U = 2$. Then, we have $\varepsilon_M = \beta^{1-t} = \frac{1}{2}$, $x_{min} = \beta^{L-1} = \frac{1}{4}$, and $x_{max} = \beta^U\left(1-\beta^{-t}\right) = 3$. The values allowed for the exponent $e$ are $-1$, 0, 1, and 2. The mantissa is $m = (a_1a_2)_\beta$ since $t = 2$; then, since $\beta = 2$, we have $a_1 = 1$, while $a_2$ is either 0 or 1. The values allowed for $m$ are therefore $m = (10)_2 = 2$ or $(11)_2 = 3$. For the sign $s = 0$, the positive real numbers in $\mathbb{F}_0$ are $x = m\beta^{e-t} = m2^{e-2}$ as summarized in the following table.

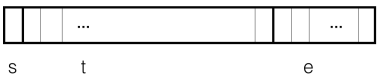



| $e$ | $-1$ | 0 | 1 | 2 |
|---|---|---|---|---|
| $m = (10)^2 = 2$ | $\frac{1}{4}$ | $\frac{1}{2}$ | 1 | 2 |
| $m = (11)^2 = 3$ | $\frac{3}{8}$ | $\frac{3}{4}$ | $\frac{3}{2}$ | 3 |

♦

**Remark 1.2** The larger is $|fl(x)|$, the lesser dense are the numbers in $\mathbb{R}$.

**Remark 1.3** For 64–bit computing (CPUs) with base $\beta = 2$, 1 digit is reserved for the sign $s$, 52 digits are generally used for $t$, while 11 for the exponent $e$.




**Remark 1.4** In MATLAB, for 64–bit CPUs, the number of digits $t$ used for the mantissa $m$ is actually $52+1 = 53$. Indeed, since the base $\beta = 2$ is used, the first digit $a_1$ is always equal to 1. Therefore, we have $\varepsilon_M = 2^{1-53} \simeq 2\cdot 10^{-16}$; in addition, we have $x_{min} \simeq 10^{-308}$, and $x_{max} \simeq 10^{308}$.

### 1.1.2 Floating point arithmetic

Algebraic operations on floating point numbers $\mathbb{F}$ do not enjoy the same properties of real numbers $\mathbb{R}$. *Round–off errors* may propagate and grow depending on the count and type of algebraic operations involved in the computations.

♦ **Example 1.3** For any $x \in \mathbb{R}\setminus\{0\}$, we have $\dfrac{(1+x)-1}{x} \equiv 1$. However, in floating point arithmetic $\dfrac{fl(1+fl(x))-1}{fl(x)} = y$, where $y$ is a real number generally different than 1. If we try to verify the

first identity in MATLAB, we obtain that $y \neq 1$, with the following errors depending on the chosen value of $x$.

| $x$ | $10^{-10}$ | $10^{-14}$ | $10^{-15}$ | $10^{-16}$ |
|---|---|---|---|---|
| relative error | $8 \cdot 10^{-6}\%$ | $8 \cdot 10^{-2}\%$ | $11\%$ | $100\%$ |

♦

The term *flops* is used to indicate the number of floating point operations.

## 1.2 Mathematical and Numerical Problems

We recall some basic notions of numerical approximation of mathematical problems.

### 1.2.1 The mathematical problem

Let us start by considering a *physical problem* (PP) endowed with a *physical solution*, symbolically indicated with $x_{ph}$, which depends on some data $d$. Then, the *mathematical problem* (MP) represents the mathematical formulation of the PP with the *mathematical solution* $x$. We indicate the MP as:

$$F(x;d) = 0, \tag{1.1}$$

with $x \in \mathscr{X}$ and $d \in \mathscr{D}$, where $\mathscr{X}$ and $\mathscr{D}$ are suitable spaces. The error between the physical and mathematical solutions is called *model error* $e_m := x_{ph} - x$.

♦ **Example 1.4** We consider as PP a body falling under the action of external forces, including the gravity, and as physical solution $x_{ph}$ the velocity of the body at a given time $t_f > 0$. In order to define the associated MP, we recall the following model:

$$\text{find } V(t) \quad : \quad \begin{cases} m\dot{V}(t) = f_{ext}(t) & \text{for } t > 0, \\ V(0) = 0, \end{cases}$$

where $V(t)$ is the body velocity, $m$ its mass, and $f_{ext}(t)$ the external forces. By identifying the mathematical solution $x$ with $V(t_f)$, i.e. $x = V(t_f)$, we have the following MP:

$$F(x;d) = x - \int_0^{t_f} \frac{f_{ext}(t)}{m}\, dt = 0,$$

where the data are $d = \{t_f, m, f_{ext}(t)\}$ . ♦

**Remark 1.5** Before solving a MP, one needs to ensure that it is *well–posed*.

Before introducing the notion of well–posedness of a MP, we recall the following definition.

**Definition 1.3** The solution $x \in \mathscr{X}$ of the MP $F(x;d) = 0$ is *continuously dependent on the data* $d \in \mathscr{D}$ if and only if, for all $\delta d$ such that $(d + \delta d) \in \mathscr{D}$ and $\delta x$ such that $F(x + \delta x; d + \delta d) = 0$, there exist two constants $\eta_0 = \eta_0(d) > 0$ and $K_0 = K_0(d)$ such that

$$\|\delta d\| \leq \eta_0 \Longrightarrow \|\delta x\| \leq K_0 \|\delta d\|, \tag{1.2}$$

with $\|\cdot\|$ a suitable norm.

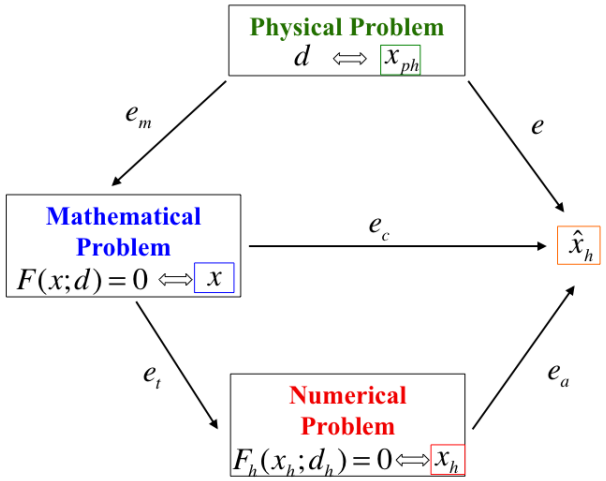



Figure 1.1: Schematic representation of the physical, mathematical, and numerical problems and their solutions.

♦ **Example 1.5** We consider the following MPs.

- $F(x;d) = x - \gamma d = 0$ for some $\gamma \in \mathbb{R}$ positive, $x \in \mathscr{X} \equiv \mathbb{R}$, and $d \in \mathscr{D} \equiv \mathbb{R}$. We have $F(x+\delta x; d+\delta d) = (x+\delta x) - \gamma(d+\delta d) = 0$, from which we obtain $\delta x = \gamma \delta d$. Hence, if we choose $\eta_0 = 1$ and $K_0 = \gamma$, the condition (1.2) is satisfied and $x \in \mathscr{X}$ is continuously dependent on $d \in \mathscr{D}$.
- $F(x;d) = x^2 - d = 0$ for $x \in \mathscr{X} \equiv \mathbb{R}$ and $d \in \mathscr{D} \equiv \mathbb{R}$. We observe that $x = \pm\sqrt{d}$ if $d \geq 0$, while $\pm\sqrt{|d|}\,i$ if $d < 0$; in this latter case, $x \in \mathbb{C}$, i.e. $x \notin \mathscr{X}$, for which $x$ is not continuously dependent on $d \in \mathscr{D} \equiv \mathbb{R}$.

♦

**Definition 1.4** The MP $F(x;d) = 0$ of Eq. (1.1) is *well–posed* (stable) if and only if there exists a unique solution $x \in \mathscr{X}$ which is continuously dependent on the data $d \in \mathscr{D}$.

MP which are formally well–posed may exhibits "large" variations of the solution $x$ even for "small" changes of the data $d$. A measure of this sensitivity is given by the conditioning number of the MP.

**Definition 1.5** The *relative conditioning number* of the MP $F(x;d) = 0$ for the data $d \in \mathscr{D}$ is:

$$K(d) := \sup_{\substack{\delta d \,:\, (d+\delta d)\in\mathscr{D} \\ \text{and } \|\delta d\|\neq 0}} \left\{ \frac{\|\delta x\|/\|x\|}{\|\delta d\|/\|d\|} \right\}.$$

**Remark 1.6** If the relative conditioning number $K(d)$ of a MP is "small", the MP is *well–conditioned*; if $K(d)$ is "large", the MP is *ill–conditioned*.

♦ **Example 1.6** We consider the MP $F(x;d) = dx - \alpha = 0$ for some $\alpha \in \mathbb{R}$, with $x \in \mathscr{X} \equiv \mathbb{R}$ and $d \in \mathscr{D} \equiv \mathbb{R}$. We have $F(x+\delta x; d+\delta d) = (d+\delta d)(x+\delta x) - \alpha = 0$, from which we obtain $\dfrac{\delta x}{x} = -\dfrac{d}{d+\delta d}\dfrac{\delta d}{d}$. We have $K(d) \simeq \sup\limits_{\substack{\delta d \,:\, (d+\delta d)\in\mathscr{D} \\ \text{and } \|\delta d\|\neq 0}} \left|\dfrac{d}{d+\delta d}\right|$, which can be "large" if $\delta d \simeq -d$. ♦

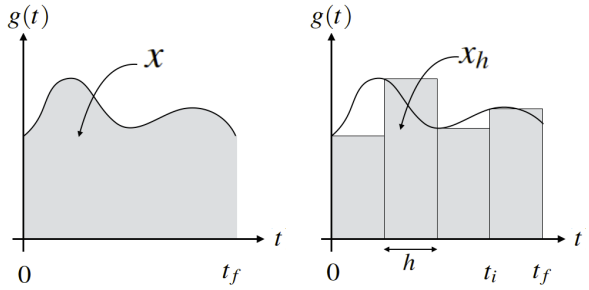
### 1.2.2 The numerical problem

The *numerical problem* (NP) is an approximation of the MP (1.1); we indicate its *numerical solution* as $x_h$, with $h$ a suitable *discretization* parameter (in other instances, $n$ is used). The error between the mathematical and numerical solutions is called *truncation error* $e_t := x - x_h$ (see also Fig. 1.1). We refer to the NP as:

$$F_h(x_h; d_h) = 0, \tag{1.3}$$

where $x_h \in \mathscr{X}_h$ and $d_h \in \mathscr{D}_h$, with $\mathscr{X}_h$ and $\mathscr{D}_h$ suitable spaces. The final solution $\widehat{x}_h$ is generally affected by the roundoff error $e_r := x_h - \widehat{x}_h$. The truncation and roundoff errors determine the *computational error* $e_c := x - \widehat{x}_h = e_t + e_r$. We remark that typically $|e_r| \ll |e_t|$, for which $e_t$ is often identified with $e_c$.

♦ **Example 1.7** For the MP $F(x;d) = x - \displaystyle\int_0^{t_f} g(t)\,dt = 0$ with the data $d = \{t_f, g(t)\}$, we can consider the NP $F_h(x_h;d_h) = x_h - h\displaystyle\sum_{i=0}^{n-1} g(t_i) = 0$, where $t_i = ih$ for $i = 0,\ldots,n$, with $h = \dfrac{t_f}{n}$.




♦

**Remark 1.7** As for the MP, also for the NP we need to ensure that it is well–posed.

**Definition 1.6** The NP problem $F_h(x_h;d_h) = 0$ of Eq. (1.3) is *well–posed* (stable) if and only if there exists a unique solution $x_h \in \mathscr{X}_h$ which is continuously dependent on the data $d_h \in \mathscr{D}_h$.

**Definition 1.7** The *relative conditioning number* of the NP $F_h(x_h;d_h) = 0$ for the data $d_h \in \mathscr{D}_h$ is:

$$K_h(d_h) := \sup_{\substack{\delta d_h \,:\, (d_h+\delta d_h)\in\mathscr{D}_h \\ \text{and } \|\delta d_h\|\neq 0}} \left\{ \frac{\|\delta x_h\|/\|x_h\|}{\|\delta d_h\|/\|d_h\|} \right\}.$$

For a NP the concept of *consistency* with the MP must be introduced.

**Definition 1.8** The NP (1.3) is *consistent* if and only if $\lim_{h\to 0} F_h(x;d) = F(x;d) = 0$, with $d \in \mathscr{D}_h$.

**Definition 1.9** The NP (1.3) is *strongly consistent* if and only if $F_h(x;d) \equiv F(x;d) = 0$ for all $h > 0$, with $d \in \mathscr{D}_h$.

♦ **Example 1.8** We consider two different NP associated to the MP $F(x;d) = x - d = 0$, with $d = \sqrt{2}$, for which $x = \sqrt{2}$.

- We define the NP $F_n(x_n;d) = x_{n+1} - \frac{3}{4}x_n - \frac{1}{2x_n} = 0$ for $n \geq 0$, with $x_0 = 1$; in this case $n$ stands for the discretization parameter and indicates the iterate number. Since $F_n(x;d) = \sqrt{2} - \frac{3}{4}\sqrt{2} - \frac{1}{2\sqrt{2}} = 0$ for all $n \geq 0$, the NP is strongly consistent.
- We set now the NP $F_n(x_n;d) = x_{n+1} - \frac{3}{4}x_n - \frac{1}{2x_n} + \frac{1}{(1+n)^5} = 0$ for $n \geq 0$, with $x_0 = 1$. We observe that $F_n(x;d) = \frac{1}{(1+n)^5} \neq 0$ for $n \geq 0$, for which the NP is not strongly consistent. However, NP is consistent since $\lim_{n\to+\infty} F_n(x;d) = 0$.

♦

Another important concept of a NP is the *convergence*.

**Definition 1.10** Let $x(d)$ be the mathematical solution of the MP $F(x(d);d) = 0$ of Eq. (1.1) and $x_h(d+\delta d_h)$ be the numerical solution of the NP $F_h(x_h(d+\delta d_h);d+\delta d_h) = 0$. Then, the NP (1.3) is *convergent* if and only if for all $\varepsilon > 0$, there exists $h_0 = h_0(\varepsilon) > 0$ and $\Delta = \Delta(h_0,\varepsilon)$ such that, for all $h < h_0$ and for all admissible $\delta d_h$ for which $\|\delta d_h\| \leq \Delta$, the condition $\|x(d) - x_h(d+\delta d_h)\| \leq \varepsilon$ is satisfied.

**Remark 1.8** If the NP is convergent, the computational error tends to zero, i.e. $\lim_{\substack{h\to 0 \\ (\text{or } n\to+\infty)}} e_c = 0$.

An important aspect related to the converge of a NP is the *convergence order*; with this aim, we redefine in the following the computational error $e_c$ as $e_c = |x - \widehat{x}_h|$.

**Definition 1.11** If the computational error $e_c \leq C h^p$, with $C$ a positive constant independent of $h$ and $p$, then the NP is *convergent* with *order* $p$.

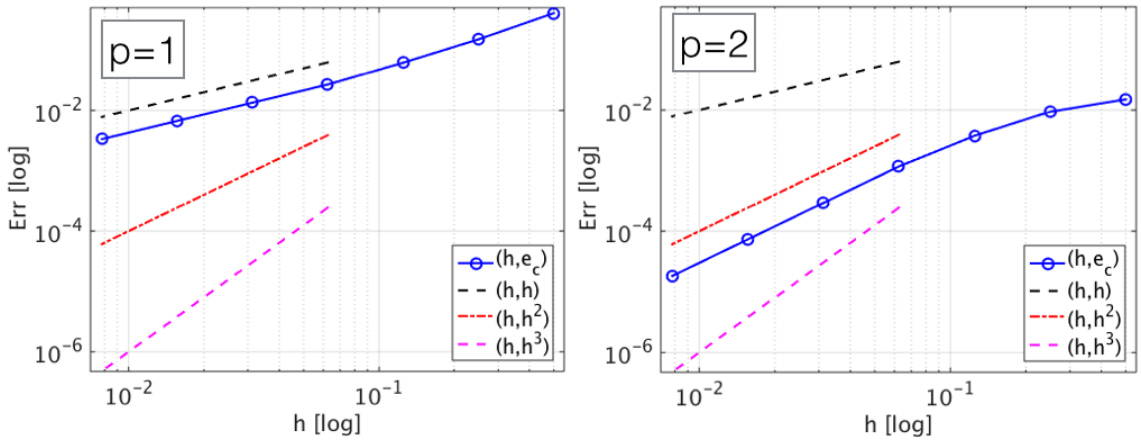
**Remark 1.9** If there exists a positive constant $\widetilde{C} \leq C$ independent of $h$ and $p$ such that $\widetilde{C}h^p \leq e_c \leq Ch^p$, then we can write $e_c \simeq Ch^p$.

**Remark 1.10** If one can assume that $e_c \simeq Ch^p$, then the convergence order $p$ can be estimated in two manners by considering a MP for which the (exact) mathematical solution $x$ is known.

- *Algebraically*. First, the errors $e_{c1}$ and $e_{c2}$ associated to two values of the discretization parameters $h_1$ and $h_2$ (which are sufficiently "small") are computed, respectively; then, by setting $e_{c1} \simeq Ch_1^p$ and $e_{c2} \simeq Ch_2^p$ and observing that $\frac{e_{c1}}{e_{c2}} = \left(\frac{h_1}{h_1}\right)^p$, $p$ is estimated as:

$$p = \frac{\log(e_{c1}/e_{c2})}{\log(h_1/h_2)}.$$

- *Graphically*. The errors $e_c$ computed for different values of $h$ are plotted vs. $h$ in *log–log scales*. Since $\log e_c = \log(Ch^p) = \log C + p\log h$, we have $p = \text{atan}(\theta)$, where $\theta$ is the slope of the $(h,e_c)$ curve, which is a straight line in log–log scales. Instead of computing the angle $\theta$, one can graphically verify if the curves $(h,e_c)$ and $(h,h^p)$ are *parallel* in log–log scales.




**Remark 1.11** A NP must be well–posed (well–conditioned), consistent, and convergent.

**Theorem 1.1 — Lax–Richtmeyer, equivalence.** If the NP $F_h(x_h; d_h) = 0$ for $x_h \in \mathscr{X}_h$ and $d_h \in \mathscr{D}_h$ is consistent, then it is well–posed if and only if it is convergent (i.e. $x_h \to x$).

The equivalence theorem is useful since it allows to verify only two of the required properties of a NP problem to yield the third one; specifically, we observe that in general it is "easy" to show the consistency of a NP, while it may be "difficult" to show its well–posedness and/or convergence.

**Remark 1.12** Following the equivalence theorem, if the NP is consistent and well–posed then it is also convergent; similarly, if the NP is consistent and convergent then it is also well–posed.

### 1.2.3 Choice of a numerical method

The choice of a numerical method (NP) to approximate the solution $x$ of a MP should take into account for:

- the (mathematical) properties of the MP;
- the computational efficiency in terms of: expected convergence order of the error, flops involved in the computation, CPU available, memory access and storage.

**Remark 1.13** If $m$ indicates the size of the NP, the flops may depend on $m$ in different manners, according to the following Table.

| | $O(1)$ | $O(m)$ | $O(m^\gamma)$ | $O(\gamma^m)$ | $O(m!)$ |
|---|---|---|---|---|---|
| flops | independent | linear | polynomial | exponential | factorial |

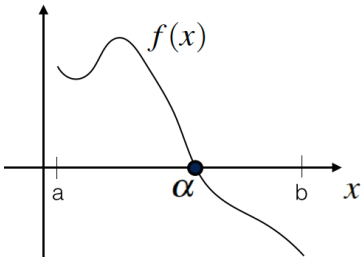
♦ **Example 1.9** For $A \in \mathbb{R}^{m\times m}$, the computation of $\det(A)$ by means of the Cramer rule involves approximately $O(m!)$ flops. The estimated times associated to the computation of $\det(A)$ for matrices of size $m$ by means of a calculator with a $1\,GHz = 10^9$ flops/s CPU are reported below.

| $m$ | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| $m!$ | 120 | $\sim 10^6$ | $\sim 10^{12}$ | $\sim 10^{18}$ |
| CPU time | $\sim 10^{-7}$ s | $\sim 10^{-3}$ s | $\sim 30$ min | $\sim 77$ years |

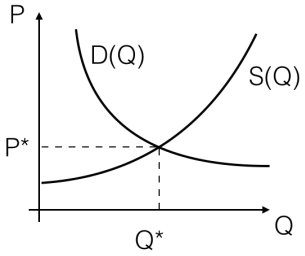
♦

# 2. Nonlinear Equations

The goal is to *numerically approximate* the *zero* $\alpha \in \mathbb{R}$ of a function $f(x)$ in the interval $I = (a,b) \subseteq \mathbb{R}$. The problem is also commonly referred as the numerical solution of a *nonlinear equation*.




♦ **Example 2.1** *Supply and demand*: microeconomic model of price determination of a good in a competitive market. The unit price $P$ of a good varies until an equilibrium between supply and demand quantities ($Q$) is met.




$P = S(Q)$ is the function representing the supply of the good; the quantity of the good increases if the price increases. $P = D(Q)$ is the demand function; the demand of the good increases if its price decreases. $(Q^*, P^*)$ is the equilibrium point for which $P^* = S(Q^*) = D(Q^*)$; if $x = Q$, one needs to solve the nonlinear equation $f(x) = S(x) - D(x) = 0$.

♦

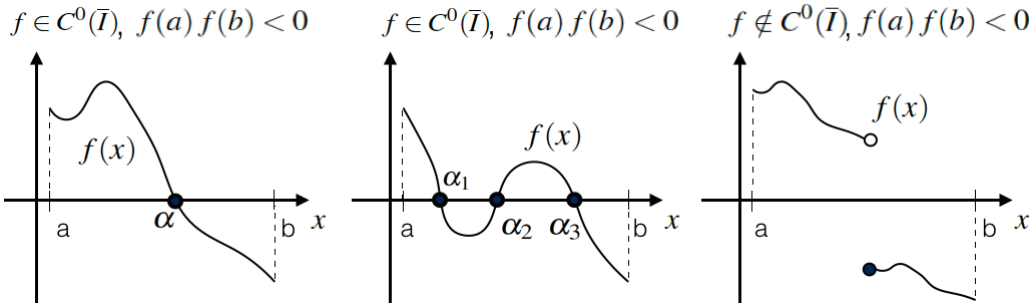
## 2.1 Bisection Method

We consider the *bisection method* for the approximation of the zero $\alpha \in I$ of a function $f(x)$.

### 2.1.1 Foundation of the bisection method

**Theorem 2.1 — Zeros of a continuous function.** Let $f(x)$ be a continuous function in the interval $I = (a,b)$, that is $f \in C^0(\bar{I}) \equiv C^0([a,b])$. If $f(a)\,f(b) < 0$, then there exists *at least* a zero $\alpha \in I$ of the function $f(x)$.

♦ **Example 2.2** We illustrate some examples for functions $f(x)$ such that $f(a)\,f(b) < 0$.



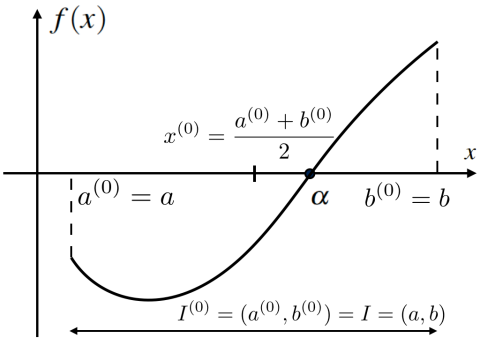

♦

Let us assume that there exists an *unique* zero $\alpha \in I$ of a function $f \in C^0(\bar{I})$ such that $f(a)\,f(b) < 0$. Then, the bisection method searches the zero $\alpha$ by recursively approximating it with the sequence of *mid–points* of the subintervals $I^{(k)}$ of $I$ for which the function $f(x)$ features changes of sign.

♦ **Example 2.3** We illustrate the bisection method and algorithm in the following pictures.

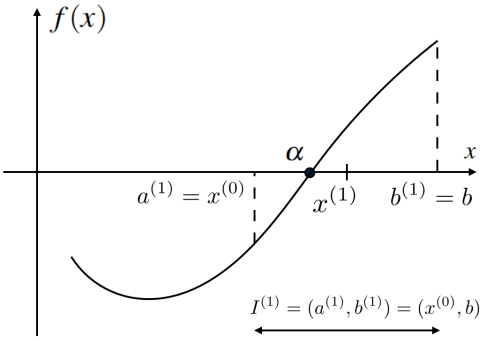
Step 0.

$I^{(0)} = \left(a^{(0)}, b^{(0)}\right) = I = (a,b)$ and
$x^{(0)} = \dfrac{a^{(0)} + b^{(0)}}{2} = \dfrac{a+b}{2}$



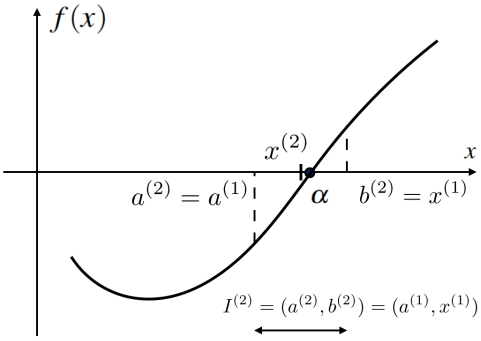

Step 1.

Since $f\left(x^{(0)}\right) f\left(b^{(0)}\right) < 0$:
$a^{(1)} = x^{(0)}$, $b^{(1)} = b$,
$I^{(1)} = (a^{(1)}, b^{(1)}) = (x^{(0)}, b)$, and
$x^{(1)} = \dfrac{a^{(1)} + b^{(1)}}{2} = \dfrac{x^{(0)} + b}{2}$.




Step 2.

Since $f\left(x^{(1)}\right) f\left(a^{(1)}\right) < 0$:
$a^{(2)} = a^{(1)}$, $b^{(2)} = x^{(1)}$,
$I^{(2)} = (a^{(2)}, b^{(2)}) = (a^{(1)}, x^{(1)})$, and
$x^{(2)} = \dfrac{a^{(2)} + b^{(2)}}{2} = \dfrac{a^{(1)} + x^{(1)}}{2}$.




♦

### 2.1.2 Algorithm and properties

We report the algorithm and the numerical properties of the bisection method.

**Algorithm 2.1:** Bisection method

set $k = 0$, $a^{(0)} = a$, $b^{(0)} = b$, and $x^{(0)} = \dfrac{a^{(0)} + b^{(0)}}{2}$;
**for** $k = 1, 2, \ldots$, *until a stopping criterion is satisfied* **do**
  **if** $f\left(x^{(k-1)}\right) = 0$ **then**
    set $\alpha = x^{(k-1)}$ and terminate the loop;
  **else**
    **if** $f\left(x^{(k-1)}\right) f\left(a^{(k-1)}\right) < 0$ **then**
      set $a^{(k)} = a^{(k-1)}$ and $b^{(k)} = x^{(k-1)}$;
    **end**
    **if** $f\left(x^{(k-1)}\right) f\left(b^{(k-1)}\right) < 0$ **then**
      set $a^{(k)} = x^{(k-1)}$ and $b^{(k)} = b^{(k-1)}$;
    **end**
    set $x^{(k)} = \dfrac{a^{(k)} + b^{(k)}}{2}$;
  **end**
**end**

**Remark 2.1** For the subinterval $I^{(k)} = \left(a^{(k)}, b^{(k)}\right))$ and its midpoint $x^{(k)} = \dfrac{a^{(k)} + b^{(k)}}{2}$ we have that both $x^{(k)}$ and $\alpha \in I^{(k)}$ for all $k \geq 0$. Moreover, since $\left|I^{(k)}\right| := b^{(k)} - a^{(k)} \equiv \dfrac{\left|I^{(k-1)}\right|}{2}$ for all $k \geq 1$, we have:

$$\left|I^{(k)}\right| = \frac{\left|I^{(0)}\right|}{2^k} = \frac{b-a}{2^k} \qquad \text{for all } k \geq 0.$$

Let us indicate the (computational) *error* associated to the bisection method as $e^{(k)} := \left|x^{(k)} - \alpha\right|$.

**Remark 2.2** The error $e^{(k)} = \left|x^{(k)} - \alpha\right|$ can be bounded from above by the size of the subinterval $I^{(k+1)}$ for all $k \geq 0$ which plays the role of *error bound*, also known as *error estimator*. We have:

$$e^{(k)} \leq \widetilde{e}^{(k)} := \left|I^{(k+1)}\right| = \frac{b-a}{2^{k+1}} \qquad \text{for all } k \geq 0. \tag{2.1}$$

This implies that the bisection method is *convergent*; indeed $\lim_{k \to +\infty} e^{(k)} = 0$ since $e^{(k)} \leq \widetilde{e}^{(k)}$ for all $k \geq 0$ and $\lim_{k \to +\infty} \widetilde{e}^{(k)} = \lim_{k \to +\infty} \dfrac{b-a}{2^{k+1}} = 0$.

**Remark 2.3** Given a tolerance $tol > 0$, one can compute the *minimum number* of *iterations* of the bisection method, say $k_{min}$, ensuring that the error $e^{(k_{min})}$ is smaller than $tol$, i.e. $e^{(k_{min})} < tol$.

Indeed, from Eq. (2.1) we have that $k_{min}$ is the smallest integer number such that $\frac{b-a}{2^{k_{min}+1}} < tol$, for which $k_{min} > \log_2\left(\frac{b-a}{tol}\right) - 1$.

We already know that the bisection method is convergent. However, we aim at characterizing such convergence. With this aim, we provide the following definition.

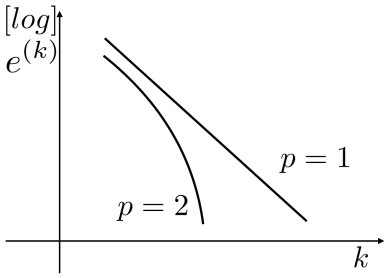
**Definition 2.1** An iterative method for the approximation of the zero $\alpha$ of the function $f(x)$ is *convergent* with *order* $p$ if and only if

$$\lim_{k\to+\infty} \frac{\left|x^{(k+1)}-\alpha\right|}{\left|x^{(k)}-\alpha\right|^p} = \mu, \tag{2.2}$$

with $\mu > 0$ a real number independent of $k$, which is called *asymptotic convergence factor*. In the case of *linear* convergence, i.e. for $p = 1$, we need $0 < \mu < 1$.

♦ **Example 2.4** We report in the following a typical plot of the sequence of errors $e^{(k)}$ vs. the iteration number $k$ for hypothetical iterative methods exhibiting convergence orders $p = 1$ and 2.

The *logarithmic scale* is used on the error axis, while the linear scale on the axis of the iteration number. We notice that the *linear* convergence ($p = 1$) is graphically represented by a straight line, whose slope depends on the asymptotic convergence factor $\mu$. A parabola is obtained instead for the *quadratic* convergence ($p = 2$).




♦

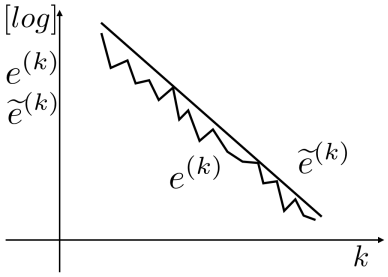
**Remark 2.4** For the bisection method the error may not be monotonically convergent, i.e. it is possible that $e^{(k+1)} \geq e^{(k)}$ for some $k \geq 0$; therefore, even if the bisection method is convergent, a convergence order can not be established according to Eq. (2.2). Similarly, the *absolute residual* $r^{(k)} := \left|f\left(x^{(k)}\right)\right|$ is not monotonically decreasing, in general.

**Remark 2.5** For the bisection method, the sequence of error estimators $\left\{\widetilde{e}^{(k)}\right\}$ is *linearly convergent* according to Eq. (2.2) with $p = 1$ and $\mu = \frac{1}{2}$; indeed:

$$\frac{\widetilde{e}^{(k+1)}}{\widetilde{e}^{(k)}} = \frac{(b-a)/2^{k+2}}{(b-a)/2^{k+1}} = \frac{1}{2} \qquad \text{for all } k \geq 0.$$

♦ **Example 2.5** We highlight the typical behavior of the sequence of errors $e^{(k)}$ and error estimators $\widetilde{e}^{(k)}$ vs. the iteration number $k$ obtained for the *bisection method*.

The *logarithmic scale* is used on the error axis, while the linear scale on the axis of the iteration number. Following Remarks 2.4 and 2.5 we graphically highlight that a convergence order cannot be established for the error, while the convergence is linear for the error estimator.




♦

### 2.1.3 Stopping criterion

The *stopping criterion* of the bisection algorithm is based on the *error estimator* (bound) $\widetilde{e}^{(k)}$ of Eq. (2.1). In Algorithm 2.1, we considered the following stopping criterion in pseudocode (in place of the *for* loop) by indicating with $tol$ a prescribed tolerance and $k_{max}$ the maximum number of iterations allowed.

**Algorithm 2.2:** Bisection method. Stopping criterion

```
...;
while (ẽ^(k) ≥ tol & k < k_max) do
    ...;
end
```

## 2.2 Newton Method

We consider the Newton method and its variants for approximating the zero $\alpha$ of the function $f(x)$.

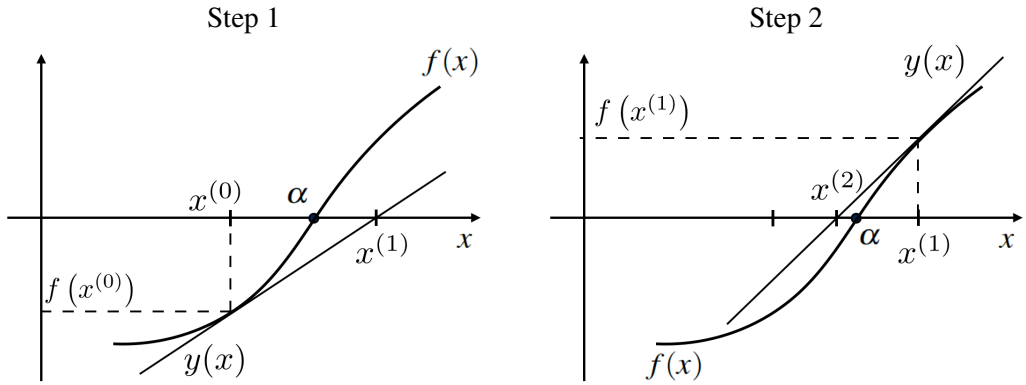
### 2.2.1 Newton method

Let us assume that $f \in C^0(I)$ and is *differentiable* in the interval $I = (a,b) \subseteq \mathbb{R}$. Given a generic iterate $x^{(k)} \in I$, the equation of the tangent line to the curve $(x, f(x))$ at the coordinate $x^{(k)}$ is $y(x) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\left(x - x^{(k)}\right)$. If we assume that $y\left(x^{(k+1)}\right) = 0$, then we compute the iterate $x^{(k+1)}$ as:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad \text{for all } k \geq 0, \tag{2.3}$$

given some *initial guess* $x^{(0)}$ and provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$. Eq. (2.3) is named *Newton iterate*. One obtains the zero $\alpha$ as the limit of the sequence of iterates $\left\{x^{(k+1)}\right\}_{k=0}^{+\infty}$ which solve the tangent equation to the curve $(x, f(x))$ in $\left\{x^{(k)}\right\}_{k=0}^{+\infty}$, respectively.

♦ **Example 2.6** We graphically highlight the Newton method in the following pictures by reporting the first two Newton iterates.




♦

In summary, the *Newton method* is applicable to a function $f \in C^0(I)$ which differentiable in $I$; then, given $x^{(0)} \in I$, the Newton method consists in sequentially applying the Newton iterate (2.3), provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$.

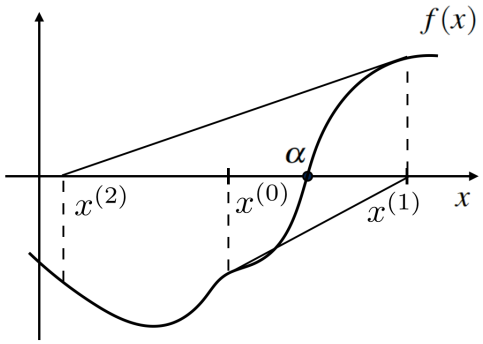
**Algorithm 2.3:** Newton method

set $k = 0$ and the initial guess $x^{(0)}$;
**while** *(stopping criterion is false)* **do**
  $x^{(k+1)} = x^{(k)} - \dfrac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}$;
  set $k = k+1$;
**end**

**Remark 2.6** Let us assume that $f \in C^2(I)$, then the *Taylor* expansion of $f(x)$ around $x^{(k)}$ reads $f\left(x^{(k+1)}\right) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\delta^{(k)} + O\left((\delta^{(k)})^2\right)$, where $\delta^{(k)} := x^{(k+1)} - x^{(k)}$ for $k \geq 0$ is the *difference of successive iterates*. If $f\left(x^{(k+1)}\right) = 0$, then the Newton method represents the first order approximation of the Taylor expansion of $f(x)$ around $x^{(k)}$; we observe that in order to satisfy this assumption, one needs $\delta^{(k)}$ to be "small".

**Remark 2.7** The choice of the *initial guess* $x^{(0)}$ is crucial for the success of the Newton method. Indeed, one needs to choose $x^{(0)}$ *"sufficiently" close* to the zero $\alpha$. As a matter of fact, the sequence of Newton iterates $\left\{x^{(k+1)}\right\}_{k=0}^{+\infty}$ may diverge, instead of converging to $\alpha$, if the initial guess $x^{(0)}$ is *not* "sufficiently" close to the zero $\alpha$. Since $\alpha$ is unknown, the choice of $x^{(0)}$ may not be trivial; in this respect, the plot of the function or the *bisection* method can be used to select $x^{(0)}$ "sufficiently" close to $\alpha$.

♦ **Example 2.7** The following example illustrates that the Newton iterates are not converging to the zero $\alpha$, due to the fact that $x^{(0)}$ is not "sufficiently" close to $\alpha$.




♦

**Remark 2.8** For (linear) affine functions in the form $f(x) = cx + d$ with $c$ and $d \in \mathbb{R}$, the Newton method converges to the zero $\alpha = -\dfrac{d}{c}$ in 1 iteration regardless of the choice of $x^{(0)}$. Indeed, we have from Eq. (2.3) that $x^{(1)} = x^{(0)} - \dfrac{f\left(x^{(0)}\right)}{f'\left(x^{(0)}\right)} = x^{(0)} - \dfrac{cx^{(0)} + d}{c} = -\dfrac{d}{c} = \alpha$ for all $x^{(0)} \in \mathbb{R}$.

We characterize in the following the convergence properties of the Newton method.

**Proposition 2.2 — Convergence of the Newton method.** If $f \in C^1(I)$, $x^{(0)}$ is "sufficiently" close to $\alpha \in I$, and $f'(\alpha) \neq 0$, then the Newton method is *convergent* to $\alpha$, provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$.

**Proposition 2.3 — Convergence order of the Newton method.** Let us indicate with $I_\alpha$ a neighborhood of $\alpha$. If $f \in C^2(I_\alpha)$, $x^{(0)}$ is "sufficiently" close to $\alpha$, and $f'(\alpha) \neq 0$, then the Newton method is *convergent with order* 2 (quadratically) to $\alpha$, provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$. In particular, we have:

$$\lim_{k \to +\infty} \frac{x^{(k+1)} - \alpha}{\left(x^{(k)} - \alpha\right)^2} = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)};$$

following Eq. (2.2), $p = 2$ is the convergence order and $\mu = \frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)}$ the asymptotic convergence factor.

*Proof.* The proof is based on the interpretation of the Newton method as a fixed point iterations method; see Sec. 2.3.5. ∎

**Definition 2.2** Let $f \in C^m(I_\alpha)$, with $m \in \mathbb{N}$ such that $m \geq 1$. The *zero* $\alpha \in I_\alpha$ is said to be of *multiplicity* $m$ if $f^{(i)}(\alpha) = 0$ for all $i = 0, \ldots, m-1$ and $f^{(m)}(\alpha) \neq 0$. If the previous condition is satisfied for $m = 1$, the zero $\alpha$ is called *simple*, otherwise is *multiple*.

**Proposition 2.4 — Convergence order of the Newton method, zero multiple.** If $f \in C^2(I_\alpha) \cap C^m(I_\alpha)$ and $x^{(0)}$ is "sufficiently" close to the zero $\alpha$ of multiplicity $m > 1$, then the Newton method is *convergent with order* 1 (linearly) to $\alpha$, provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$. In particular, following Eq. (2.2), we have:

$$\lim_{k \to +\infty} \frac{\left|x^{(k+1)} - \alpha\right|}{\left|x^{(k)} - \alpha\right|} = \mu,$$

with $p = 1$ the convergence order and $\mu \in (0, 1)$ the asymptotic convergence factor.

**Remark 2.9** If the zero $\alpha$ is simple $m = 1$, the Newton method converges at least quadratically according to Proposition 2.3. Conversely, if the zero $\alpha$ is multiple ($m > 1$), the Newton method only converges linearly according to Proposition 2.4. We observe that, generally, the higher is the convergence order, fewer iterations are necessary to reach a prescribed value of the error, i.e. the method results to be more efficient.

Examples of linear and quadratic convergence are graphically highlighted in Example 2.4.

### 2.2.2 Modified Newton method

Let us assume that $f \in C^m(I_\alpha)$, with $\alpha \in I_\alpha$ and $m \geq 1$ the multiplicity of $\alpha$. The $k$–the iterate of the *modified Newton method* reads:

$$x^{(k+1)} = x^{(k)} - m \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad \text{for all } k \geq 0, \tag{2.4}$$

given the initial guess $x^{(0)}$ and provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$. Following Algorithm 2.3, we have the following one for the modified Newton method.

**Algorithm 2.4:** Modified Newton method

choose $m$;
set $k = 0$ and the initial guess $x^{(0)}$;
**while** *(stopping criterion is false)* **do**
  $x^{(k+1)} = x^{(k)} - m\dfrac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}$;
  set $k = k+1$;
**end**

**Remark 2.10** The modified Newton method requires the a priori knowledge of the multiplicity $m$ of the zero $\alpha$. The latter can be eventually estimated by means of suitable numerical approaches.

The convergence properties of the modified Newton method are characterized by the following Proposition.

**Proposition 2.5 — Convergence order of the modified Newton method.** If $f \in C^2(I_\alpha) \cap C^m(I_\alpha)$, with $m \geq 1$ the multiplicity of the zero $\alpha \in I_\alpha$, and $x^{(0)}$ is "sufficiently" close to $\alpha$, then the modified Newton method is *convergent with order* 2 (quadratically) to $\alpha$, provided that $f'\left(x^{(k)}\right) \neq 0$ for all $k \geq 0$.

♦ **Example 2.8** We approximate the zero $\alpha = 0$ of the function $f(x) = \sin^m(x)$ in the interval $I = \left(-\dfrac{\pi}{2}, \dfrac{\pi}{2}\right)$, with $m = 1, 2, 3, \ldots$; with this aim we consider the Newton and modified Newton methods. We observe that $f'(x) = m\sin^{m-1}(x)\cos(x)$, for which $f'(\alpha) = 1$ if $m = 1$ and $f'(\alpha) = 0$ for $m \geq 2$; the zero $\alpha$ is simple for $m = 1$, but multiple ($m$ times) for $m \geq 2$. If we set the initial guess $x^{(0)} = \dfrac{\pi}{6}$, the first iterate of the Newton method yields $x^{(1)} = \dfrac{\pi}{6} - \dfrac{1}{\sqrt{3}m}$, for which $x^{(1)}$ is farther and farther from $\alpha$ (and closer and closer to $x^{(0)}$), the larger is $m$. Conversely, for the modified Newton method we have from Eq. (2.4) that $x^{(1)} = \dfrac{\pi}{6} - \dfrac{1}{\sqrt{3}}$, regardless of the value of $m \geq 1$. ♦

### 2.2.3 Stopping criterion for Newton method

We consider different stopping criterion for the Newton method and its variants. Since the zero $\alpha$ is in general unknown, the error $e^{(k)} = \left|x^{(k)} - \alpha\right|$ is also unknown; therefore, we need a suitable *error estimator* (error indicator) $\widetilde{e}^{(k)}$ such that $\widetilde{e}^{(k)} \simeq e^{(k)}$. By referring e.g. to the Newton and modified Newton Algorithms 2.3 and 2.4, the iterations are stopped for $k = k_{min}$ such that $\widetilde{e}^{(k_{min})} < tol$, with $tol$ a prescribed tolerance, or when the maximum number of iterations is reached; see e.g. Algorithm 2.2.

First, we consider the criterion based on the *difference of successive iterates*, for which the error estimator is chosen as:

$$\widetilde{e}^{(k)} = \begin{cases} \left|\delta^{(k-1)}\right| & \text{if } k \geq 1 \\ tol + 1 & \text{if } k = 0 \end{cases} \qquad \text{with } \delta^{(k)} := x^{(k+1)} - x^{(k)} \qquad \text{for } k \geq 0.$$

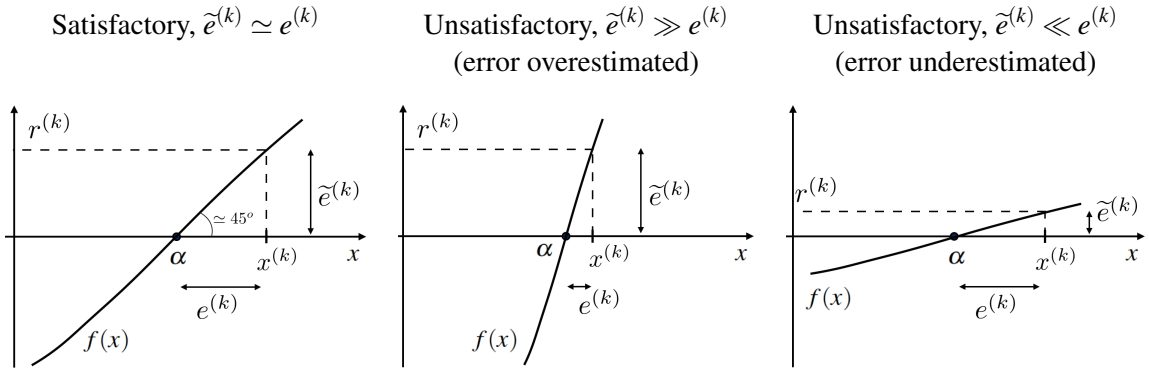
The criterion is *satisfactory* if the zero $\alpha$ is simple; this can be shown by interpreting the Newton method as a fixed point iterations method; see Sec. 2.3.5.

Another criterion is based on the absolute *residual*, for which:

$$\widetilde{e}^{(k)} = \left| r^{(k)} \right| \qquad \text{with } r^{(k)} := f\left(x^{(k)}\right) \qquad \text{for } k \geq 0.$$

This criterion is *satisfactory* if $\left|f'(x)\right| \simeq 1$ for $x \in I_\alpha$ a neighborhood of $\alpha$, for which $\widetilde{e}^{(k)} \simeq e^{(k)}$. Conversely, the criterion is *unsatisfactory* if $\left|f'(x)\right| \gg 1$ or $\left|f'(x)\right| \simeq 0$ for $x \in I_\alpha$. Specifically, if $\left|f'(x)\right| \gg 1$ for $x \in I_\alpha$, the error is *overestimated* by the error estimator ($\widetilde{e}^{(k)} \gg e^{(k)}$), for which more Newton iterations than necessary are performed; therefore, the stopping criterion is inefficient. If $\left|f'(x)\right| \simeq 0$ for $x \in I_\alpha$, the error is *underestimated* by the error estimator ($\widetilde{e}^{(k)} \ll e^{(k)}$), for which the Newton iterations are prematurely stopped since the error is larger than predicted by the estimator.

♦ **Example 2.9** The following examples graphically illustrate the situations for which the criterion based on the *residual* is satisfactory or unsatisfactory.




♦

## 2.2.4 Inexact and quasi–Newton methods

The Newton and modified Newton methods require the evaluation of the first derivative of the function $f(x)$; see Eqs. (2.3) and (2.4). However, in some cases of practical interest, the evaluation of $f'(x)$ may be "difficult" or computationally expensive. Therefore, by referring e.g. to the Newton iterate (2.3), $f'\left(x^{(k)}\right)$ can be approximated by a computationally feasible quantity $q^{(k)} \simeq f'\left(x^{(k)}\right)$. *Inexact* or *quasi–Newton methods* are based on the use of an approximate value of $f'\left(x^{(k)}\right)$. The general quasi–Newton iterate reads:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{q^{(k)}} \qquad \text{for all } k \geq 0,$$

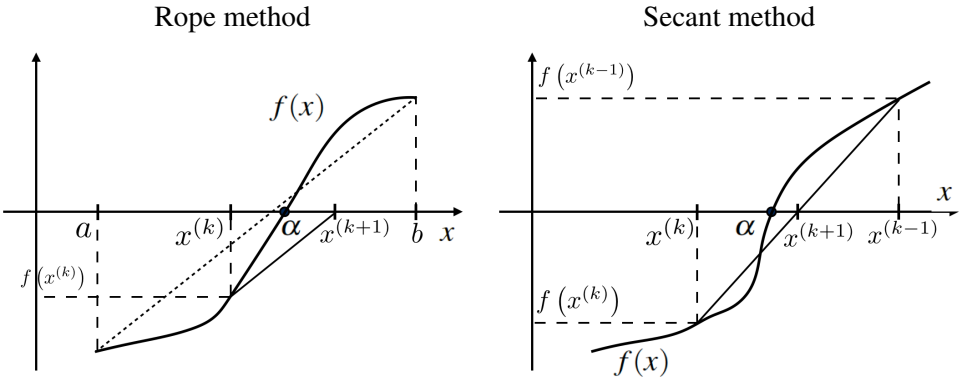
with $q^{(k)}$ determining the method. We consider the following cases:

- for $q^{(k)} \equiv f'\left(x^{(k)}\right)$, we obtain the standard Newton method;
- for $q^{(k)} = \dfrac{f(b)-f(a)}{b-a}$ for all $k \geq 0$, with $\alpha \in (a,b)$, we have the *rope method*;
- for $q^{(k)} = \dfrac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}$ for all $k \geq 1$, we have the *secant method*[1].

We observe that the secant method converges with order $p = 1.6$ if the zero $\alpha$ is simple.

[1] For the secant method, $q^{(0)}$ can be set as for the rope method.

♦ **Example 2.10** The following examples graphically illustrate the *rope* and *secant* methods at the general iterate $x^{(k)}$.




♦

### 2.2.5 Newton method for systems of nonlinear equations

The Newton method can be used to approximate the solution of systems of nonlinear equations. Given $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^d$, for some $d \geq 1$, the problem consists in finding the vector $\boldsymbol{\alpha} \in \mathbb{R}^d$ such that $\mathbf{F}(\boldsymbol{\alpha}) = \mathbf{0}$. More specifically, we have:

$$\mathbf{x} = \begin{Bmatrix} x_1 \\ \vdots \\ x_d \end{Bmatrix} \quad \text{and} \quad \mathbf{F}(\mathbf{x}) = \begin{Bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_d(\mathbf{x}) \end{Bmatrix} = \begin{Bmatrix} f_1(x_1, \ldots, x_d) \\ \vdots \\ f_d(x_1, \ldots, x_d) \end{Bmatrix}.$$

**Definition 2.3** Let $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^d$ be differentialble in $I_{\mathbf{x}} \subseteq \mathbb{R}^d$ a neighborhood of $\mathbf{x} \in \mathbb{R}^d$, then its *Jacobian* in $\mathbf{x}$ is $J_{\mathbf{F}} : \mathbb{R}^d \to \mathbb{R}^{d \times d}$ such that $(J_{\mathbf{F}}(\mathbf{x}))_{ij} = \dfrac{\partial f_i}{\partial x_j}(\mathbf{x})$ for $i, j = 1, \ldots, d$.

The *Newton method* is applicable to a system of equations $\mathbf{F} \in C^0(I_{\boldsymbol{\alpha}})$ which is differentiable in $I_{\boldsymbol{\alpha}} \subseteq \mathbb{R}^d$, a neighborhood of $\boldsymbol{\alpha}$; then, given $x^{(0)} \in I_{\boldsymbol{\alpha}}$, the Newton method consists in sequentially applying the following Newton iterate:

$$\text{solve } J_{\mathbf{F}}\left(\mathbf{x}^{(k)}\right) \boldsymbol{\delta}^{(k)} = -\mathbf{F}\left(\mathbf{x}^{(k)}\right) \quad \text{and} \quad \text{set } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}^{(k)} \qquad \text{for all } k \geq 0, \tag{2.5}$$

provided that $\det\left(J_{\mathbf{F}}\left(\mathbf{x}^{(k)}\right)\right) \neq 0$ for all $k \geq 0$. The Newton algorithm using the stopping criterion based on the difference of successive iterates is reported in the following.

**Remark 2.11** At each Newton iterate, one needs to solve a linear system, unless $d = 1$ for which $J_{\mathbf{F}}\left(x^{(k)}\right) \equiv f'\left(x^{(k)}\right)$. We also observe that the Newton iterate (2.5) can be obtained by the first order expansion of $\mathbf{F}(\mathbf{x})$ around $\mathbf{x}^{(k)}$ as $\mathbf{F}\left(\mathbf{x}^{(k)}\right) + J_{\mathbf{F}}\left(\mathbf{x}^{(k)}\right)\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = \mathbf{0}$.

Regarding the convergence of the Newton method for systems of nonlinear equations, we state the following.

**Proposition 2.6** If $\mathbf{F} \in C^2(I_{\boldsymbol{\alpha}})$, with $I_{\boldsymbol{\alpha}} \subseteq \mathbb{R}^d$ a neighborhood of $\boldsymbol{\alpha}$, $\mathbf{x}^{(0)} \in \mathbb{R}^d$ is "sufficiently" close to $\boldsymbol{\alpha}$, and $\det\left(J_{\mathbf{F}}(\boldsymbol{\alpha})\right) \neq 0$, then the Newton method converges with order $p = 2$.

♦ **Example 2.11** Let us consider the system of nonlinear equations $\mathbf{F}(\mathbf{x}) = \begin{Bmatrix} \sin(x_1x_2)+x_2 \\ x_1+x_2-\frac{1}{2}e^{-x_1x_2} \end{Bmatrix}$, with the zero $\boldsymbol{\alpha} = \begin{Bmatrix} \frac{1}{2} \\ 0 \end{Bmatrix}$. Its Jacobian is $J_{\mathbf{F}}(\mathbf{x}) = \begin{bmatrix} x_2\cos(x_1x_2) & x_1\cos(x_1x_2)+1 \\ 1+\frac{x_2}{2}e^{-x_1x_2} & 1+\frac{x_1}{2}e^{-x_1x_2} \end{bmatrix}$, for which $\det(J_{\mathbf{F}}(\boldsymbol{\alpha})) = -\frac{3}{2} \neq 0$. ♦

## 2.3 Fixed Point Iterations

We consider the fixed point iterations method both to find the fixed point of an iteration function, as well as to solve nonlinear equations.

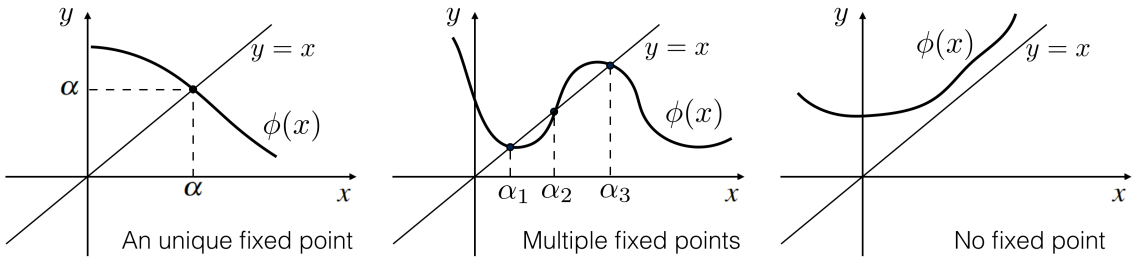
### 2.3.1 Nonlinear equations, zeros, fixed points, and iteration functions

Given a function $f : \mathbb{R} \to \mathbb{R}$, we aim at finding the zero $\alpha$ (i.e. such that $f(\alpha) = 0$). With this goal, we can transform the problem of finding the zero $\alpha$ into a *fixed point iterations problem*.

**Definition 2.4** Given the *iteration function* $\phi : [a,b] \subseteq \mathbb{R} \to \mathbb{R}$, we say that $\alpha \in \mathbb{R}$ is a *fixed point* of $\phi$ if and only if $\phi(\alpha) = \alpha$.

♦ **Example 2.12** For the iteration function $\phi(x) = \cos(x)$ in the interval $[0.1, 1.1]$, we have the fixed point $\alpha = \cos(\alpha) \simeq 0.7391$. ♦

♦ **Example 2.13** We graphically illustrate the fixed points of some iteration functions.




♦

**Remark 2.12** The goal is to find the zero $\alpha$ of the nonlinear function $f(x)$. We transform this problem in a fixed point iterations problem by suitably choosing a fixed point iterations function $\phi(x)$ such that $f(\alpha) = 0$ if and only if $\phi(\alpha) = \alpha$ for $\alpha \in [a,b]$. We remark that there are different iterations functions $\phi(x)$ and multiple manners to obtain them in order to achieve this goal.

♦ **Example 2.14** The simplest manner to obtain $\phi(x)$ from $f(x)$ is based on the following steps. Since $f(\alpha) = 0$, we have $f(\alpha)+\alpha = \alpha$, for which we can set $\phi(x) = f(x)+x$. We remark that this is very often not a "good" choice for the iteration function. ♦

♦ **Example 2.15** We consider $f(x) = 2x^2 - x - 1$ for which we are interested in the zero $\alpha = 1$. A possibility consists in setting, following the previous Remark, $\phi_1(x) = f(x)+x = 2x^2 - 1$. A second possibility can be derived by setting $f(x) = 0$, from which we have $x^2 = \frac{x+1}{2}$ and then $x = \pm\sqrt{\frac{x+1}{2}}$; in this case, we can take $\phi_2(x) = \sqrt{\frac{x+1}{2}}$. ♦

### 2.3.2 Fixed point iterations algorithm

We state the fixed point iterations algorithm, based on the fixed point iterate:

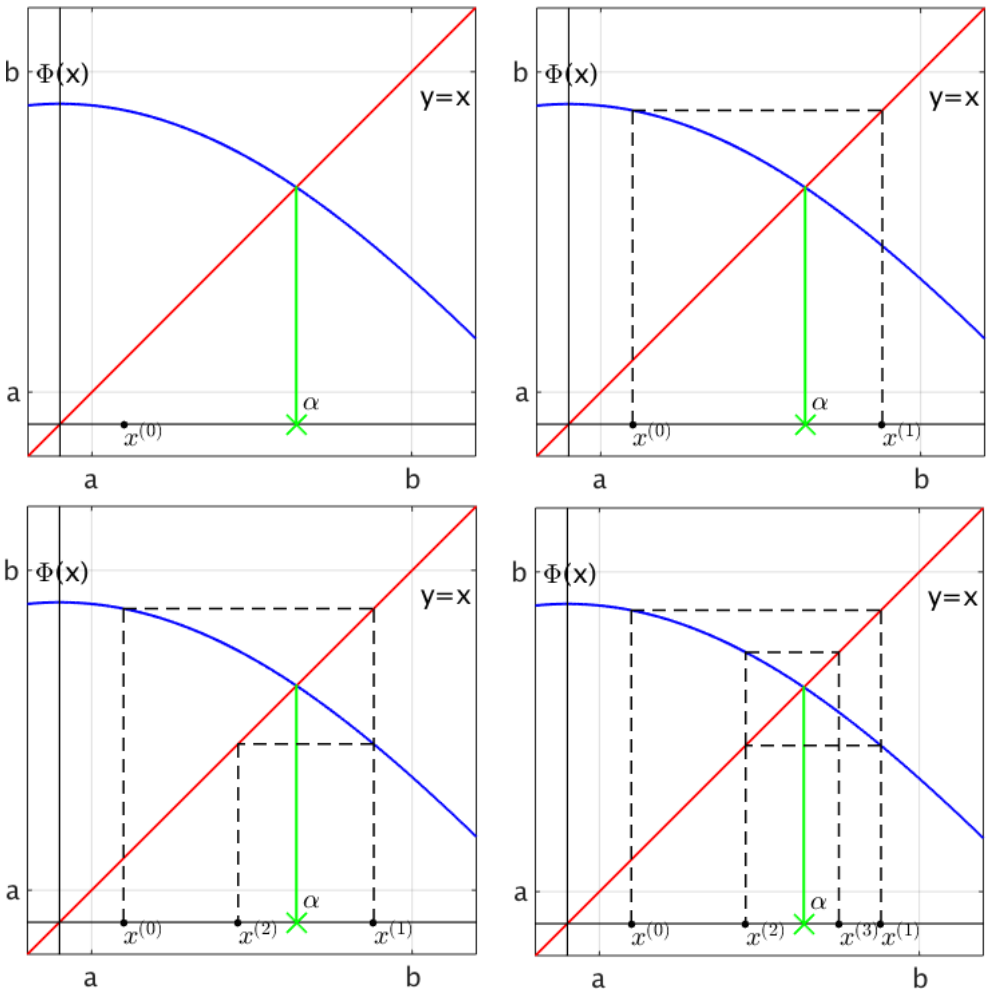
$$x^{(k+1)} = \phi\left(x^{(k)}\right) \qquad k \geq 0, \tag{2.6}$$

for some initial guess $x^{(0)}$.

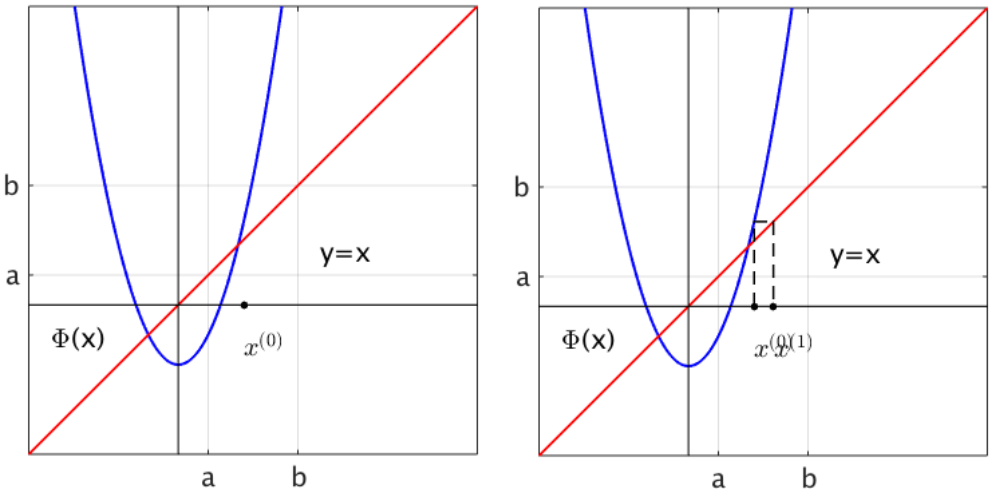
**Algorithm 2.5:** Fixed point iterations

set $k = 0$ and the initial guess $x^{(0)}$;
**while** *(stopping criterion is false)* **do**
  $x^{(k+1)} = \phi\left(x^{(k)}\right)$;
  set $k = k + 1$;
**end**

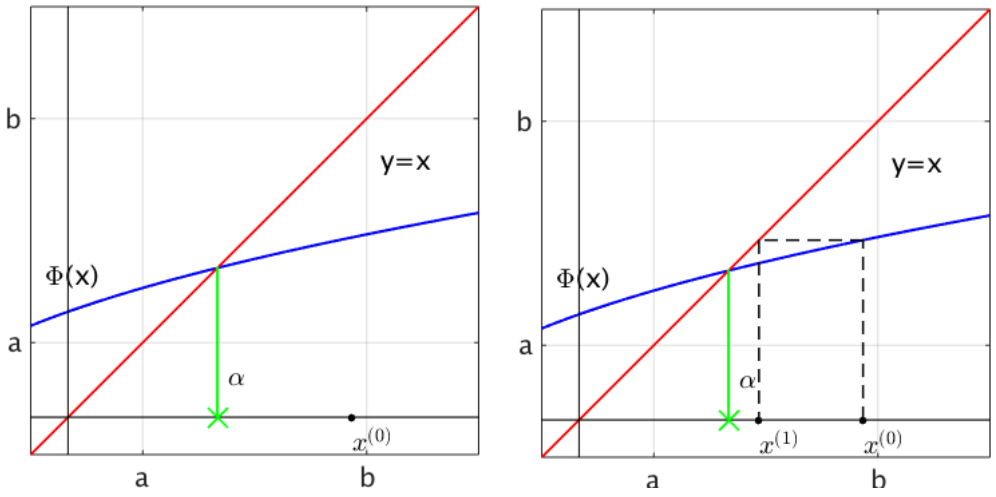
♦ **Example 2.16** We graphically illustrate the fixed point iterations algorithm in the following. First, we consider $\phi(x) = \cos(x)$, with $a = 0.1$, $b = 1.1$, and $x^{(0)} = 0.2$, for which we observe that the algorithm is converging to $\alpha = \cos(\alpha) \simeq 0.7391$.




Then, we consider $\phi(x) = 2x^2 - 1$, with $a = 0.5$, $b = 2$, and $x^{(0)} = 1.1$, for which the algorithm is diverging from the fixed point $\alpha = 1$; we observe that $\phi(x)$ corresponds to the iteration function $\phi_1(x)$ of Example 2.15.




Finally, we consider $\phi(x) = \sqrt{\dfrac{1+x}{2}}$, with $a = 0.5$, $b = 2$, and $x^{(0)} = 1.9$, for which the algorithm converges to $\alpha = 1$; in this case, $\phi(x)$ corresponds to $\phi_2(x)$ in Example 2.15.




♦

### 2.3.3 Convergence properties of fixed point iterations

We need to identify the properties of the iteration function $\phi(x)$ in terms of existence and uniqueness of the fixed point $\alpha$, as well as the convergence of the fixed point iterations method.

**Proposition 2.7 — Global convergence in an interval.** Let us consider the iteration function $\phi : \mathbb{R} \to \mathbb{R}$ and the fixed point iterations of Eq. (2.6).

1. If $\phi \in C^0([a,b])$ and $\phi(x) \in [a,b]$ for all $x \in [a,b]$, then there *exists at least* a fixed point $\alpha \in [a,b]$ of $\phi(x)$.
2. If, in addition to the hypothesis of (1), there exists a constant $L \in [0,1)$ such that $|\phi(x_1) - \phi(x_2)| \leq L\,|x_1 - x_2|$ for all $x_1, x_2 \in [a,b]$, then the fixed point $\alpha$ is *unique* in $[a,b]$ and the fixed point iterations algorithm *converges* ($\lim_{k\to+\infty} x^{(k)} = \alpha$) for all the initial guesses $x^{(0)} \in [a,b]$.

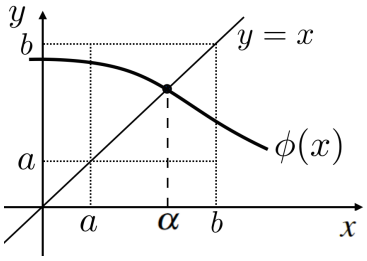
*Proof.* (1) We show the existence of $\alpha \in [a,b]$ according to the hypothesis (1). We introduce a function $g(x) = \phi(x) - x$ such that $g(\alpha) = 0$. Since $\phi \in C^0([a,b])$, also $g \in C^0([a,b])$. In addition,

since $\phi(x) \in [a,b]$ for all $x \in [a,b]$, we have $g(a) = \phi(a) - a \geq 0$ and $g(b) = \phi(b) - b \leq 0$, for which $g(a)\,g(b) \leq 0$. Since $g(x)$ is satisfying the hypotheses of Theorem 2.1, there exists at least a zero $\alpha$ of $g(x)$ in $[a,b]$; the latter is also a fixed point of $\phi(x)$ (indeed, $g(\alpha) = \phi(\alpha) - \alpha = 0$).

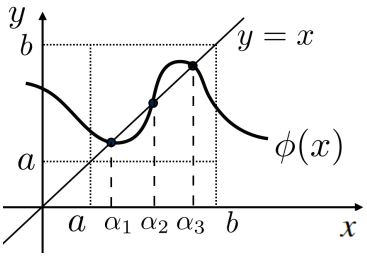
(2) We show the uniqueness of $\alpha \in [a,b]$ and the convergence of the method for all $x^{(0)} \in [a,b]$ according to the hypotheses (2). We assume, by absurd, that there exist two distinct fixed points $\alpha_1 \neq \alpha_2$ such that $\phi(\alpha_1) = \alpha_1$ and $\phi(\alpha_2) = \alpha_2$. According to this assumption, we have $0 < |\alpha_1 - \alpha_2| = |\phi(\alpha_1) - \phi(\alpha_2)| \leq L\,|\alpha_1 - \alpha_2|$; since $L < 1$ by hypothesis, we have $0 < |\alpha_1 - \alpha_2| < |\alpha_1 - \alpha_2|$, which is absurd. Therefore, $\alpha_1 \equiv \alpha_2 = \alpha$, i.e. the fixed point is unique. Regarding the convergence of the method, we observe that the error $e^{(k+1)} = \left|x^{(k+1)} - \alpha\right| = \left|\phi\left(x^{(k)}\right) - \phi(\alpha)\right| \leq \left|x^{(k)} - \alpha\right| = L e^{(k)}$. By recursion, $e^{(k)} \leq L^k e^{(0)}$ for all $k \geq 0$; since $L < 1$, we have $\lim_{k \to +\infty} e^{(k)} = 0$, i.e. the method is convergent for all $x^{(0)} \in [a,b]$. ∎

♦ **Example 2.17** We illustrate the results of Proposition 2.7 with the following examples.

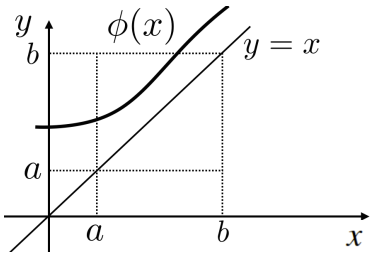
The hypotheses (1) and (2) of Proposition 2.7 are satisfied, therefore there exists an unique fixed point $\alpha \in [a,b]$ and the method converges to $\alpha$ for all $x^{(0)} \in [a,b]$.




The hypotheses (1) are satisfied, but not the hypotheses (2) of Proposition 2.7; therefore, we can only guarantee that there exists at least a fixed point $\alpha \in [a,b]$.




The hypotheses (1) and (2) of Proposition 2.7 are not satisfied, therefore there may not exist any fixed point $\alpha \in [a,b]$.




♦

We consider the following result on the global convergence in the interval $[a,b]$ which uses more restrictive hypotheses on the iteration function $\phi(x)$ with respect to Proposition 2.7.

**Proposition 2.8 — Global convergence in an interval.** If $\phi \in C^1([a,b])$, $\phi(x) \in [a,b]$ for all $x \in [a,b]$, and $\left|\phi'(x)\right| < 1$ for all $x \in [a,b]$, then there *exists an unique* fixed point $\alpha \in [a,b]$ and the fixed point iterations method *converges* for all $x^{(0)} \in [a,b]$ with *order* at least equal to 1 (linearly), i.e.:

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

with $\phi'(\alpha)$ the asymptotic convergence factor.

We illustrate now some results on the local convergence to the fixed point $\alpha$, i.e. in a neighborhood of $\alpha$. First, we recall the Lagrange theorem.

**Theorem 2.9 — Lagrange, mean value.** If the function $g \in C^1([a,b])$, then there exists $\xi \in (a,b)$ such that $g(a) - g(b) = g'(\xi)(a-b)$.

**Proposition 2.10 — Ostrowski, local convergence in a neighborhood of the fixed point.** If $\phi \in C^1(I_\alpha)$, with $I_\alpha$ a neighborhood of the fixed point $\alpha$ of $\phi(x)$, and $\left|\phi'(\alpha)\right| < 1$, then, if the initial guess $x^{(0)}$ is "sufficiently" close to $\alpha$, the fixed point iterations method *converges* with *order* at least equal to 1 (linearly), i.e.:

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

with $\phi'(\alpha)$ the asymptotic convergence factor.

*Proof.* We only show that the method is at least linearly convergent. Under the hypothesis of the Lagrange Theorem 2.9, we have $x^{(k+1)} - \alpha = \phi\left(x^{(k)}\right) - \phi(\alpha) = \phi'\left(\xi^{(k)}\right)\left(x^{(k)} - \alpha\right)$, for some $\xi^{(k)}$ between $\alpha$ and $x^{(k)}$. If $\lim_{k\to+\infty} x^{(k)} = \alpha$, also $\lim_{k\to+\infty} \xi^{(k)} = \alpha$ and hence $\lim_{k\to+\infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \lim_{k\to+\infty} \phi'\left(\xi^{(k)}\right) = \phi'(\alpha)$. ∎

**Remark 2.13** Following Proposition 2.10, we observe for $\phi \in C^1(I_\alpha)$ that:

- if $\left|\phi'(\alpha)\right| < 1$, the fixed point iterations converge to $\alpha$ with order at least equal to 1, if $x^{(0)}$ is "sufficiently" close to $\alpha$;
- if $\left|\phi'(\alpha)\right| \equiv 1$, the convergence of the method to $\alpha$ depends on the properties of $\phi(x)$ in the neighborhood $I_\alpha$ and the choice of the initial guess $x^{(0)}$ (as a matter of fact, the method may converge or diverge);
- if $\left|\phi'(\alpha)\right| > 1$, the convergence of the method to $\alpha$ is impossible, unless $x^{(0)} \equiv \alpha$.

**Proposition 2.11 — Local convergence in a neighborhood of the fixed point.** If $\phi \in C^2(I_\alpha)$, with $I_\alpha$ a neighborhood of the fixed point $\alpha$ of $\phi(x)$, $\phi'(\alpha) = 0$, and $\phi''(\alpha) \neq 0$, then, if the initial guess $x^{(0)}$ is "sufficiently" close to $\alpha$, the fixed point iterations method *converges* with *order* 2 (quadratically), i.e.:

$$\lim_{k\to+\infty} \frac{x^{(k+1)} - \alpha}{\left(x^{(k)} - \alpha\right)^2} = \frac{1}{2}\phi''(\alpha),$$

with $\frac{1}{2}\phi''(\alpha)$ the asymptotic convergence factor.

The following result generalizes the previous ones.

**Proposition 2.12 — Local convergence in a neighborhood of the fixed point.** If $\phi \in C^p(I_\alpha)$ for $p > 1$, with $I_\alpha$ a neighborhood of the fixed point $\alpha$ of $\phi(x)$, $\phi^{(i)}(\alpha) = 0$ for all $i = 1, \ldots, p-1$, and $\phi^{(p)}(\alpha) \neq 0$, then, if the initial guess $x^{(0)}$ is "sufficiently" close to $\alpha$, the fixed point iterations method *converges* with *order* $p$, i.e.:

$$\lim_{k \to +\infty} \frac{x^{(k+1)} - \alpha}{\left(x^{(k)} - \alpha\right)^p} = \frac{1}{p!}\phi^{(p)}(\alpha),$$

with $\frac{1}{p!}\phi^{(p)}(\alpha)$ the asymptotic convergence factor.

### 2.3.4 Stopping criterion for fixed point iterations

We need to consider a stopping criterion to terminate the fixed point iterations of Algorithm 2.5. With this aim, we introduce a suitable *error estimator* $\widetilde{e}^{(k)}$ of the error $e^{(k)} := \left|x^{(k)} - \alpha\right|$. Such error estimator is based on the *difference of successive iterates*, i.e.:

$$\widetilde{e}^{(k)} = \begin{cases} \left|\delta^{(k-1)}\right| & \text{if } k \geq 1 \\ tol + 1 & \text{if } k = 0 \end{cases} \qquad \text{with } \delta^{(k)} := x^{(k+1)} - x^{(k)} \qquad \text{for } k \geq 0;$$

$tol > 0$ is a suitable tolerance. The fixed point iterations algorithm is stopped at the first iteration $\bar{k}$ such that $\widetilde{e}^{(\bar{k})} < tol$ or when $\bar{k} = k_{max}$, with $k_{max}$ the maximum number of iterations allowed.

We observe that $\alpha - x^{(k+1)} = \alpha - x^{(k)} + x^{(k)} - x^{(k+1)} = \left(\alpha - x^{(k)}\right) - \delta^{(k)}$. Moreover, if $\phi \in C^1(I_\alpha)$, we have from Theorem 2.9 that $\alpha - x^{(k+1)} = \phi'\left(\xi^{(k)}\right)\left(\alpha - x^{(k)}\right)$ for some $\xi^{(k)}$ between $x^{(k)}$ and $\alpha$. Therefore, we have $x^{(k)} - \alpha = \phi'\left(\xi^{(k)}\right)\left(x^{(k)} - \alpha\right) - \delta^{(k)}$ and hence:

$$x^{(k)} - \alpha = -\frac{1}{1 - \phi'\left(\xi^{(k)}\right)}\delta^{(k)}, \tag{2.7}$$

for some $\xi^{(k)}$ between $x^{(k)}$ and $\alpha$. We use the previous result to determine if the stopping criterion based on the difference of successive iterates is satisfactory or not. If $\phi'(x) \simeq 0$ in a neighborhood of $\alpha$ ($\phi'(\alpha) \simeq 0$), the criterion is *satisfactory* since $e^{(k)} \simeq \widetilde{e}^{(k+1)}$. If $\phi'(x) > -1$, but $\phi'(x) \simeq -1$ in a neighborhood of $\alpha$, the criterion is still *satisfactory* since $e^{(k)} \simeq \frac{1}{2}\widetilde{e}^{(k+1)}$ (the error is *overestimated* by the estimator by a factor 2). Conversely, if $\phi'(x) < 1$, but $\phi'(x) \simeq 1$ in a neighborhood of $\alpha$, the criterion is *unsatisfactory* since $e^{(k)} \ll \widetilde{e}^{(k+1)}$, that is the error is *underestimated* by the error estimator.

### 2.3.5 The Newton method as a fixed point iterations method

The *Newton method* (see Sec. 2.2) can be used to find the zero $\alpha$ of a general function $f(x)$, for which the Newton iterate is specified in Eq. (2.3). The problem of finding the zero $\alpha$ of $f(x)$ with the Newton method can be recast in a fixed point iterations method by using the iteration function $\phi_N(x)$ such that $\phi_N(\alpha) = \alpha$. From Eqs. (2.3) and (2.6) the iteration function associated to the Newton method reads:

$$\phi_N(x) = x - \frac{f(x)}{f'(x)}. \tag{2.8}$$

It follows that the properties of the Newton method, including convergence to $\alpha$ can be deduced from those of the iteration function $\phi_N(x)$.

**Proposition 2.13** If $f \in C^m(I_\alpha)$, with $I_\alpha$ a neighborhood of the zero $\alpha$ and $m \geq 1$ the multiplicity of $\alpha$, for the iteration function $\phi_N(x)$ of Eq. (2.8), we have $\phi'_N(\alpha) = 1 - \frac{1}{m}$.

*Proof.* The proof, reported in Exercises Series 4, is based on rewriting $f(x)$ as $f(x) = (x-\alpha)^m g(x)$ for $x \in I_\alpha$, with the function $g(x)$ such that $g(\alpha) \neq 0$ and $g^{(i)}(\alpha) = 0$ for all $i = 1, \ldots, m$. ∎

**Corollary 2.14** If $f \in C^2(I_\alpha)$, $\alpha$ is a zero simple ($m = 1$), and $x^{(0)}$ is "sufficiently" close to $\alpha$, then the Newton method converges with order 2 (quadratically), indeed:

$$\lim_{k \to +\infty} \frac{x^{(k+1)} - \alpha}{\left(x^{(k)} - \alpha\right)^2} = \frac{1}{2}\phi''_N(\alpha) = \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)}.$$

*Proof.* The result follows from Propositions 2.11 and 2.12, Eq. (2.8), and Proposition 2.13. ∎

**Remark 2.14** From Proposition 2.13, if $\alpha$ is a zero simple ($m = 1$), we have $\phi'_N(\alpha) = 1 - \frac{1}{m} \equiv 0$.

**Corollary 2.15** If $f \in C^m(I_\alpha)$, $\alpha$ is a zero of multiplicity $m > 1$, and $x^{(0)}$ is "sufficiently" close to $\alpha$, then the Newton method converges with order 1 (linearly), indeed:

$$\lim_{k \to +\infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'_N(\alpha) = 1 - \frac{1}{m} \neq 0.$$

*Proof.* The result follows from Proposition 2.10, Eq. (2.8), and Proposition 2.13. ∎

Similarly, to the *modified Newton method* (see Sec. 2.2.2), based on the iterate of Eq. (2.4), we associate the iteration function $\phi_{mN}(x)$ defined as:

$$\phi_N(x) = x - m\frac{f(x)}{f'(x)}, \tag{2.9}$$

where $m$ is the multiplicity of the zero $\alpha$.

**Proposition 2.16** If $f \in C^m(I_\alpha)$, with $I_\alpha$ a neighborhood of the zero $\alpha$ and $m \geq 1$ the multiplicity of $\alpha$, for the iteration function $\phi_{mN}(x)$ of Eq. (2.9), we have $\phi'_{mN}(\alpha) = 1 - m\frac{1}{m} \equiv 0$ for all $m \geq 1$.

*Proof.* The result follows analogously to that of Proposition 2.13. ∎

**Corollary 2.17** If $f \in C^2(I_\alpha) \cap C^m(I_\alpha)$, $\alpha$ is a zero of multiplicity $m \geq 1$, and $x^{(0)}$ is "sufficiently" close to $\alpha$, then the modified Newton method converges with order 2 (quadratically).

*Proof.* The result follows from Propositions 2.11 and 2.12, Eq. (2.9), and Proposition 2.16. ∎

Regarding the quality of the *stopping criterion* based on the *difference of successive iterates* for the Newton method discussed in Sec. 2.2.3, we recall the properties presented in Sec. 2.3.4 for fixed point iterations. From Eq. (2.7) and Proposition 2.13, we remark that:

$$e^{(k)} = \left|x^{(k)} - \alpha\right| \simeq \left|\frac{1}{1 - \phi'_N(\alpha)}\right| \widetilde{e}^{(k+1)} = m\,\widetilde{e}^{(k+1)},$$

where the error estimator $\widetilde{e}^{(k+1)} = \delta^{(k)} = \left|x^{(k+1)} - x^{(k)}\right|$ and $m \geq 1$ is the multiplicity of the zero $\alpha$. Therefore, if the zero $\alpha$ is simple ($m = 1$), we have $e^{(k)} \simeq \widetilde{e}^{(k+1)}$ and the stopping criterion based on the difference of successive iterates is *satisfactory*. Otherwise, for a zero of multiplicity $m > 1$, and especially for $m \gg 1$, the criterion is *unsatisfactory* since the error is *underestimated* by the error estimator, i.e. $e^{(k)} \gg \widetilde{e}^{(k+1)}$. By using similar arguments for the *modified Newton method*, the stopping criterion based on the difference of successive iterates is *satisfactory*, since $e^{(k)} \simeq \widetilde{e}^{(k+1)}$ regardless of the multiplicity $m \geq 1$ of the zero.

### 2.3.6 Fixed point iterations for vector valued functions

The fixed point iterations method can be used with vector valued iteration functions $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^d$, for some $d \geq 1$. In this case, the problem consists in finding the vector $\boldsymbol{\alpha} \in \mathbb{R}^d$, the fixed point, such that $\boldsymbol{\phi}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}$. The fixed point iterations method consists in sequentially applying the following fixed point iterate:

$$\mathbf{x}^{(k+1)} = \boldsymbol{\phi}\left(\mathbf{x}^{(k)}\right) \qquad \text{for all } k \geq 0,$$

given the initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^d$; as stopping criterion the one based on the *difference of successive iterates* can be used similarly to Sec. 2.3.4, i.e. $\widetilde{\mathbf{e}}^{(k)} = \left\|\boldsymbol{\delta}^{(k-1)}\right\|_2 < tol$ for $k \geq 1$, with $tol$ a prescribed tolerance and $\boldsymbol{\delta}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$.

## Algorithms for Chapter 2

**Algorithm 1** Bisection method (Algorithms 2.1 and 2.2)

**Input:** $f, a, b, tol, k_{max}$
**Output:** $\alpha$

1: Set $k = 0$, $a^{(0)} = a$, $b^{(0)} = b$, $x^{(0)} = (a+b)/2$, $\widetilde{e}^{(0)} = (b-a)/2$
2: **while** ($\widetilde{e}^{(k)} > tol$ and $k < k_{max}$) **do**
3: **if** $f\left(x^{(k)}\right) = 0$ **then**
4: Set $\alpha = x^{(k)}$
5: Return
6: **else if** $f\left(a^{(k)}\right) f\left(x^{(k)}\right) < 0$ **then**
7: Set $a^{(k+1)} = a^{(k)}$, $b^{(k+1)} = x^{(k)}$
8: **else if** $f\left(b^{(k)}\right) f\left(x^{(k)}\right) < 0$ **then**
9: Set $a^{(k+1)} = x^{(k)}$, $b^{(k+1)} = b^{(k)}$
10: **end if**
11: Set $x^{(k+1)} = (a^{(k+1)} + b^{(k+1)})/2$
12: Set $\widetilde{e}^{(k+1)} = (b^{(k+1)} - a^{(k+1)})/2$
13: Set $k = k+1$
14: **end while**
15: Set $\alpha = x^{(k)}$

**Algorithm 2** Newton method (Algorithm 2.3)

**Input:** $f, x^{(0)}, tol, k_{max}$
**Output:** $\alpha$

1: Set $k = 0$, $\widetilde{e}^{(0)} = tol + 1$
2: **while** ($\widetilde{e}^{(k)} > tol$ and $k < k_{max}$) **do**
3: Set $x^{(k+1)} = x^{(k)} - f\left(x^{(k)}\right) / f'\left(x^{(k)}\right)$
4: Set $\widetilde{e}^{(k+1)} = \|x^{(k+1)} - x^{(k)}\|$
5: Set $k = k+1$
6: **end while**
7: Set $\alpha = x^{(k)}$

The algorithm for the modified Newton method is analogous, adding the multiplicity $m$ in the input, and modifying line 3 as in Algorithm 2.4.

**Algorithm 3** Newton method for systems of nonlinear equations

**Input:** $\mathbf{F}, \mathbf{x}^{(0)}, tol, k_{max}$
**Output:** $\alpha$

Set $k = 0$, $\widetilde{e}^{(0)} = tol + 1$
**while** ($\widetilde{e}^{(k)} > tol$ and $k < k_{max}$) **do**
    Set $\delta^{(k)} = -\left(J_{\mathbf{F}}\left(\mathbf{x}^{(k)}\right)\right)^{-1} \mathbf{F}\left(\mathbf{x}^{(k)}\right)$
    Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta^{(k)}$
    Set $\widetilde{e}^{(k+1)} = \|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|$
    Set $k = k + 1$
**end while**
Set $\alpha = \mathbf{x}^{(k)}$

**Algorithm 4** Fixed point iterations (Algorithm 2.5)

**Input:** $\phi, x^{(0)}, tol, k_{max}$
**Output:** $\alpha$

Set $k = 0$, $\widetilde{e}^{(0)} = tol + 1$
**while** ($\widetilde{e}^{(k)} > tol$ and $k < k_{max}$) **do**
    Set $x^{(k+1)} = \phi\left(x^{(k)}\right)$
    Set $\widetilde{e}^{(k+1)} = \|x^{(k+1)} - x^{(k)}\|$
    Set $k = k + 1$
**end while**
Set $\alpha = x^{(k)}$

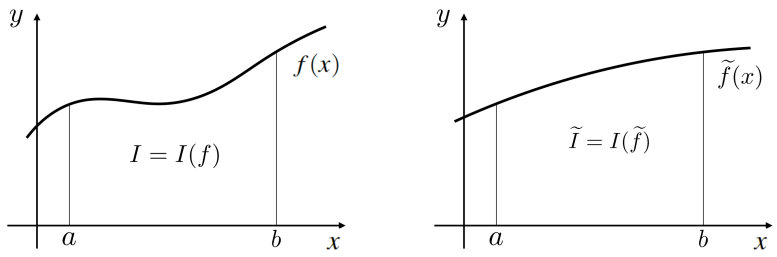
# 3. Approximation of Functions and Data

We consider the approximation of functions and data, specifically by means of *interpolation* and the *least–squares* method.

## 3.1 Motivations and Examples

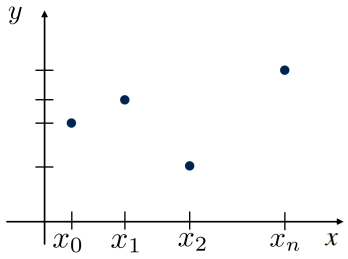
We illustrate through examples the motivations behind the need to approximate functions and data.

♦ **Example 3.1** Let us assume to be interested in computing the integral $I$ of a function $f(x)$ in the interval $[a,b]$, i.e. $I = I(f) = \int_a^b f(x)\,dx$, but we are unable to provide a closed form solution for the function at hand. One possibility consists in approximating $f(x)$ with another function $\widetilde{f}(x)$ which can be integrated in closed form as $\widetilde{I} = I(\widetilde{f}) = \int_a^b \widetilde{f}(x)\,dx$ such that $\widetilde{I} \simeq I$.



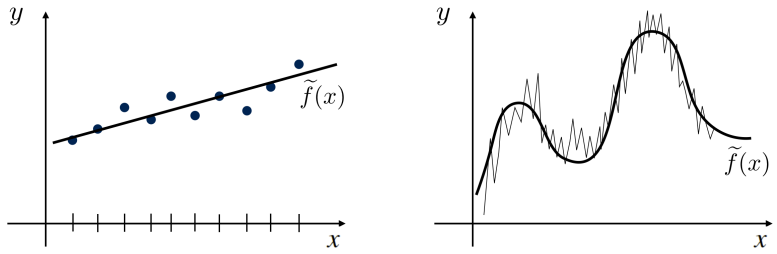

♦

♦ **Example 3.2** By assuming that a function $f(x)$ is known only through its evaluation in a set of $n+1$ nodes $\{x_i\}_{i=0}^n$, i.e. the data couples $\{(x_i, f(x_i))\}_{i=0}^n$, we may be interested in defining an approximation $\widetilde{f}(x)$ of the unknown function $f(x)$.




♦

♦ **Example 3.3** Given a set of data couples $\{(x_i, y_i)\}_{i=0}^{n}$, we may want to determine the intermediate values or make predictions outside the set of $n+1$ nodes $\{x_i\}_{i=0}^{n}$.




♦

### 3.1.1 Approximation of functions by Taylor's polynomials

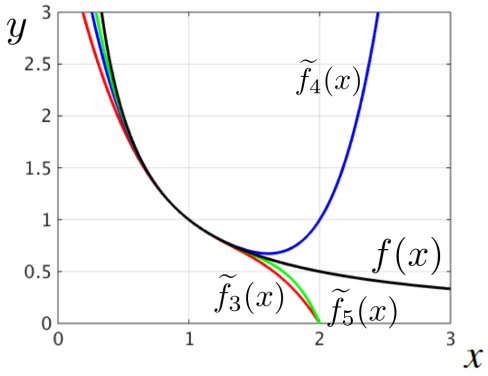
A manner to approximate a function $f \in C^n(I_{x_0})$ in a neighborhood $I_{x_0}$ of a point $x_0 \in \mathbb{R}$ is based on the *Taylor's polynomial* (expansion) of order $n$. The Taylor's expansion of $f(x)$ around $x_0$ reads:

$$\widetilde{f}(x) = f(x_0) + \sum_{i=1}^{n} \frac{1}{i!} f^{(i)}(x_0)\,(x - x_0)^i.$$

However, the approximation of $f(x)$ with $\widetilde{f}(x)$ presents some drawbacks. First, the evaluation of $n$ derivatives of $f(x)$ is required, which may be computationally expensive. Then, the Taylor's expansion is accurate only in a neighborhood $I_{x_0}$ of $x_0$, while generally inaccurate "far" from $x_0$.

♦ **Example 3.4** We consider the Taylor's expansion of $f(x) = \dfrac{1}{x}$ of order $n$, say $\widetilde{f}_n(x)$, around $x_0 = 1$.

Since $f^{(i)}(x) = (-1)^i\, i!\, x^{-(i+1)}$ for $i = 0, 1, \ldots, n$, we have $\widetilde{f}(x) = \widetilde{f}_n(x) = 1 + \sum_{i=1}^{n} (-1)^n\,(x-1)^i$. As reported in the figure, the approximations provided by $\widetilde{f}_n(x)$ may be very inaccurate "far" from $x_0 = 1$.




♦

## 3.2 Interpolation

We introduce and define the concept of interpolation and list different types of interpolations.

**Definition 3.1** We consider a set of $n+1$ data couples $\{(x_i,y_i)\}_{i=0}^n$ with $\{x_i\}_{i=0}^n$ $n+1$ *distinct nodes*, i.e. such that $x_i \neq x_j$ for all $i \neq j$ for $i,j=0,\dots,n$; in the case the function $f(x)$ is known, we set $y_i = f(x_i)$ for all $i=0,\dots,n$. *Interpolating* the data couples $\{(x_i,y_i)\}_{i=0}^n$ means determining the approximate function $\widetilde{f}(x)$ such that $\widetilde{f}(x_i) = y_i$ for all $i=0,\dots,n$ or, if $f(x)$ is known, such that $\widetilde{f}(x_i) = f(x_i)$ for all $i=0,\dots,n$. The function $\widetilde{f}(x)$ is called *interpolant* of the data at the nodes.

There exist different types of interpolation. For example:

- *polynomial* interpolation, for which $\widetilde{f}(x) = a_0 + a_1 x + \cdots + a_n x^n$ for some $n+1$ coefficients $a_0, a_1, \dots$;
- *rational* interpolation, for which $\widetilde{f}(x) = \dfrac{a_0 + a_1 x + \cdots + a_k x^k}{a_{k+1} + a_{k+2} x + \cdots + a_{k+n+1} x^n}$ for some coefficients $a_0, a_1, \dots$ with $k, n \geq 0$;
- *trigonometric* interpolation, for which $\widetilde{f}(x) = \sum_{j=-M}^{M} a_j e^{\imath j x}$, being $\imath$ the imaginary unit ($\imath^2 = -1$) and $e^{\imath j x} = \cos(jx) + \imath \sin(jx)$, for some $M$ and complex coefficients $a_j$.
- *piecewise polynomial* interpolation;
- *splines*;
- ...

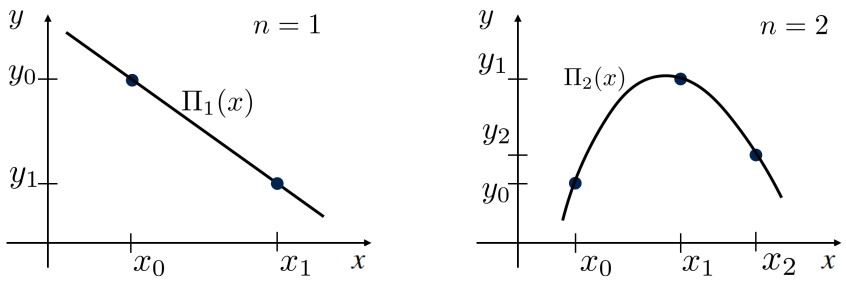
### 3.2.1 Lagrange interpolating polynomials

We consider the *polynomial* interpolation, specifically determined as *Lagrange interpolating polynomials*. The polynomial interpolation is based on the following result which determines the correspondence between the number of distinct nodes and the degree of the interpolant.

**Proposition 3.1** For any set of data couples $\{(x_i,y_i)\}_{i=0}^n$, being $\{x_i\}_{i=0}^n$ $n+1$ distinct nodes, there exists an *unique* polynomial, say $\Pi_n(x)$, of degree less than or equal to $n$, such that $\Pi_n(x_i) = y_i$ for all $i=0,\dots,n$. $\Pi_n(x) \in \mathbb{P}_n$ is called *interpolating polynomial* of the data at the nodes $\{x_i\}_{i=0}^n$. If $f(x)$ is a *continuous* function for which $y_i = f(x_i)$ for all $i=0,\dots,n$, then $\Pi_n f(x) \in \mathbb{P}_n$ is the *interpolating polynomial* of the *function* $f(x)$ at the nodes $\{x_i\}_{i=0}^n$.

We recall that $\mathbb{P}_n$ indicates the set of polynomials of degree less than or equal to $n$.

♦ **Example 3.5** We illustrate two cases for which $n=1$ (left) and $n=2$ (right).




♦

We need to determine the interpolating polynomial $\Pi_n(x)$ (or $\Pi_n f(x)$), which assumes the expression $\Pi_n(x) = a_0 + a_1 x + \cdots + a_n x^n$; the goal consists in computing the coefficients $\{a_i\}_{i=0}^n$ of such

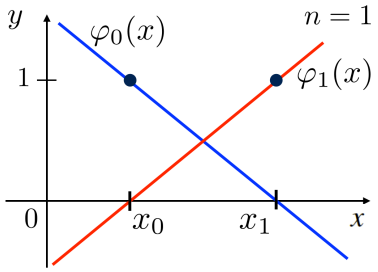
polynomial of degree $n$. With this aim, we consider a special family of polynomials associated to the $n+1$ distinct nodes $\{x_i\}_{i=0}^n$.

**Definition 3.2** For a set of $n+1$ distinct nodes $\{x_i\}_{i=0}^n$, the *Lagrange characteristic* function associated to the node $x_k$, say $\varphi_k \in \mathbb{P}_n$, is a *polynomial* of degree $n$ such that $\varphi_k(x_i) = \delta_{ki}$ for all $i = 0, \dots, n$, where $\delta_{ki} = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$, for which:

$$\varphi_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}.$$

The set $\{\varphi_k(x)\}_{k=0}^n$ is the basis of *Lagrange characteristic polynomials*.

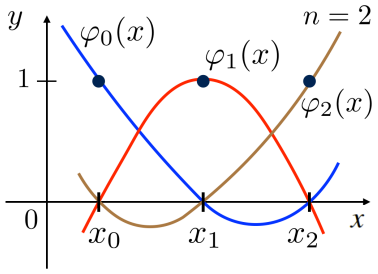
♦ **Example 3.6** We illustrate the bases of Lagrange characteristic polynomials for $n = 1$ and $n = 2$.




$n = 1$

$$\varphi_0(x) = \frac{x - x_1}{x_0 - x_1} \in \mathbb{P}_1$$

$$\varphi_1(x) = \frac{x - x_0}{x_1 - x_0} \in \mathbb{P}_1$$




$n = 2$

$$\varphi_0(x) = \frac{x - x_1}{x_0 - x_1} \frac{x - x_2}{x_0 - x_2} \in \mathbb{P}_2$$

$$\varphi_1(x) = \frac{x - x_0}{x_1 - x_0} \frac{x - x_2}{x_1 - x_2} \in \mathbb{P}_2$$

$$\varphi_2(x) = \frac{x - x_0}{x_2 - x_0} \frac{x - x_1}{x_2 - x_1} \in \mathbb{P}_2$$

♦

**Definition 3.3** Given the basis of Lagrange characteristic polynomials $\{\varphi_k(x)\}_{k=0}^n$ associated to the $n+1$ distinct nodes $\{x_i\}_{i=0}^n$, the *Lagrange interpolating polynomial* of the data couples $\{(x_i, y_i)\}_{i=0}^n$ is:

$$\Pi_n(x) = \sum_{k=0}^{n} y_k \, \varphi_k(x).$$

If the function $f(x)$ is given and is continuous, the *Lagrange interpolating polynomial* of the function $f(x)$ at the nodes $\{x_i\}_{i=0}^n$ is:

$$\Pi_n f(x) = \sum_{k=0}^{n} f(x_k) \, \varphi_k(x).$$

**Remark 3.1** The Lagrange interpolating polynomial $\Pi_n(x)$ interpolates the data at the nodes; indeed, $\Pi_n(x_i) = \sum_{k=0}^{n} y_k \, \varphi_k(x_i) = \sum_{k=0}^{n} y_k \, \delta_{ki} = y_i$ for all $i = 0, \dots, n$. Analogously, $\Pi_n f(x)$ interpolates the function $f(x)$ at the nodes.

The Lagrange interpolating polynomial $\Pi_n(x) \in \mathbb{P}_n$ uses the basis of Lagrange characteristic polynomials $\{\varphi_k(x)\}_{k=0}^n$ to determine the coefficients $\{a_i\}_{i=0}^n$ of such polynomial of degree $n$, i.e. $\Pi_n(x) = \sum_{k=0}^{n} y_k\,\varphi_k(x) = a_0 + a_1 x + \cdots + a_n x^n$.

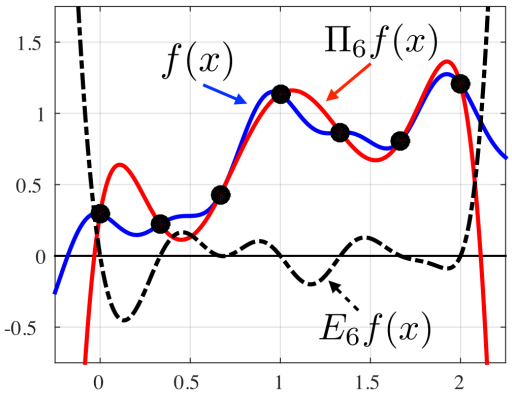
♦ **Example 3.7** We consider the Lagrange polynomial interpolation of the data couples $\{(1,3)\}$, $\{(2,2)\}$, and $\{(4,6)\}$ for which $n=2$. Following Example 3.6, we have $x_0=1$, $x_1=2$, and $x_2=4$, for which $\varphi_0(x) = \frac{1}{3}(x-2)(x-4) = \frac{1}{3}x^2 - 2x + \frac{8}{3}$, $\varphi_1(x) = -\frac{1}{2}(x-1)(x-4) = -\frac{1}{2}x^2 + \frac{5}{2}x + 2$, and $\varphi_2(x) = \frac{1}{6}(x-1)(x-2) = \frac{1}{6}x^2 - \frac{1}{2}x + \frac{1}{3}$. The Lagrange interpolating polynomial of the data is the polynomial of degree $n=2$ $\Pi_2(x) = y_0\,\varphi_0(x) + y_1\,\varphi_1(x) + y_2\,\varphi_2(x) = x^2 - 4x + 6$, being $y_0=3$, $y_1=2$, and $y_3=6$. ♦

**Definition 3.4** For a continuous function $f(x)$ and the interval $I=[a,b]$ such that the $n+1$ nodes are ordered as $a = x_0 < x_1 < \cdots < x_n = b$, we define the *error function* $E_n f(x) := f(x) - \Pi_n f(x)$ associated to the interpolating polynomial $\Pi_n f(x)$. The *error* is $e_n(f) := \max_{x\in I} |E_n f(x)|$.

**Remark 3.2** $\Pi_n f(x)$ interpolates $f(x)$ at the nodes, indeed $E_n f(x_i) = 0$ for all $i = 0, \ldots, n$.

♦ **Example 3.8** Given the function $f(x) = \sin(x) + \frac{1}{4}\sin\left(2\pi x + \sqrt{3}\right) + \frac{1}{10}\sin\left(4\pi x + \sqrt{7}\right)$, we consider its polynomial interpolation over $n+1$ equally spaced nodes in $I=[0,2]$.

We set the polynomial degree $n=6$ for which we obtain the interpolating polynomial $\Pi_6 f(x)$ of $f(x)$ at the nodes $x_0=0$, $x_1=\frac{1}{3}$, $x_2=\frac{2}{3}$, $x_3=1$, $x_4=\frac{4}{3}$, $x_5=\frac{5}{3}$, and $x_6=2$. We also plot the error function $E_6 f(x) = f(x) - \Pi_6 f(x)$, for which we observe that $E_6 f(x_i) = 0$ for all $i = 0, \ldots, 6$.




♦

**Proposition 3.2** Let us consider $n+1$ distinct nodes $\{x_i\}_{i=0}^n$ in an interval $I=[a,b]$ such that $a = x_0 < x_1 < \cdots < x_n = b$ and the polynomial interpolant $\Pi_n f(x)$ of a function $f(x)$ in such nodes. Then, if $f \in C^{n+1}(I)$, for all $x \in I$ there exists $\xi = \xi(x) \in I$ such that:

$$E_n f(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi(x))\,\omega_n(x), \tag{3.1}$$

where $\omega_n(x) := \prod_{i=0}^{n}(x - x_i)$. Moreover, the error $e_n(f)$ is bounded by the error estimator $\widetilde{e}_n(f)$ as:

$$e_n(f) \le \widetilde{e}_n(f) := \frac{1}{(n+1)!} \max_{x\in I} \left| f^{(n+1)}(x) \right| \max_{x\in I} |\omega_n(x)|\,. \tag{3.2}$$

**Proposition 3.3** Let us consider $n+1$ *equally spaced* nodes $\{x_i\}_{i=0}^n$ in the interval $I=[a,b]$ such that $x_i = x_0 + ih$ for $i=0,\dots,n$ with $x_0=a$, $x_n=b$, and $h=\dfrac{b-a}{n}$, then, by recalling the definition of $\omega_n(x)$ in Proposition 3.2, we have:

$$\max_{x\in I}|\omega_n(x)| \le \frac{n!}{4}h^{n+1} = \frac{n!}{4}\left(\frac{b-a}{n}\right)^{n+1}.$$

Therefore, from Eq. (3.2), the error $e_n(f)$ is bounded as:

$$e_n(f) \le \widetilde{e}_n(f) := \frac{h^{n+1}}{4(n+1)}\max_{x\in I}\left|f^{(n+1)}(x)\right| = \frac{1}{4(n+1)}\left(\frac{b-a}{n}\right)^{n+1}\max_{x\in I}\left|f^{(n+1)}(x)\right|. \quad (3.3)$$

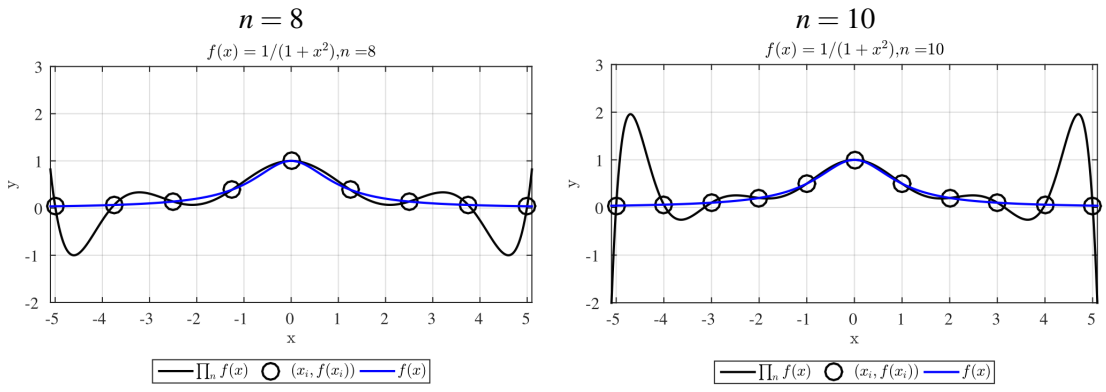
**Corollary 3.4** Under the same hypotheses of Proposition 3.3, we have:

$$\max_{x\in I}\left|f'(x)-(\Pi_n f)'(x)\right| \le C_n h^n \max_{x\in I}\left|f^{(n+1)}(x)\right|$$

for some positive constant $C_n$.

**Remark 3.3** If the $n+1$ nodes are equally spaced in the interval $I$, the error $e_n(f)$ may tend to zero or not for $n\to+\infty$ depending on the function $f(x)$ to be interpolated. From Eq. (3.3), we observe that $\lim_{n\to+\infty}\dfrac{h^{n+1}}{4(n+1)}=0$. Conversely, $\max_{x\in I}\left|f^{(n+1)}(x)\right|$ may grow with $n$; indeed, there exist functions for which $\lim_{n\to+\infty}\max_{x\in I}\left|f^{(n+1)}(x)\right|=+\infty$. In these cases, the growth of $\max_{x\in I}\left|f^{(n+1)}(x)\right|$ may not be compensated by the decrease of $\dfrac{h^{n+1}}{4(n+1)}$ with $n$, for which $\lim_{n\to+\infty}\widetilde{e}_n(f)=+\infty$; hence, the error estimator $\widetilde{e}_n(f)$ "blows up" and typically the error $e_n(f)$ behaves in a similar manner. The so called *Runge phenomenon* is an instance of such behavior, for which the error function $E_n f(x)$ tends to "blow up" for increasing values of $n$ in proximity of the borders of the interval $I$ when equally spaced nodes are used for the polynomial interpolation.

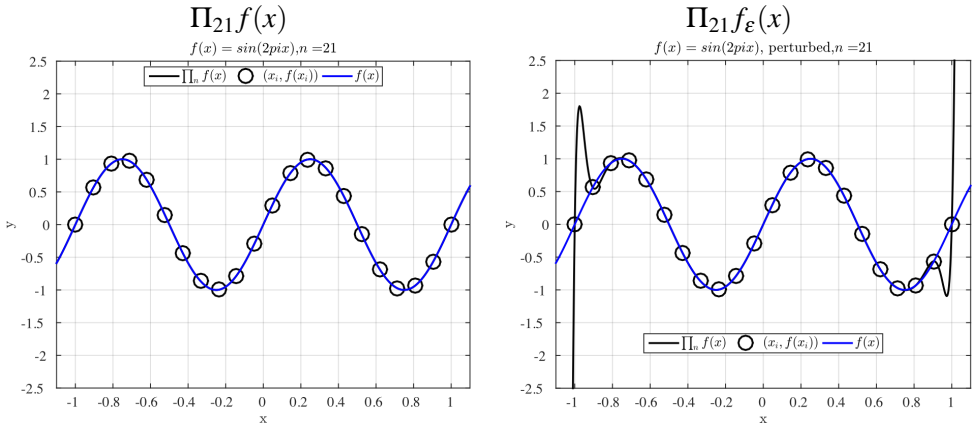
♦ **Example 3.9** We consider the polynomial interpolation of the *Runge function* $f(x)=\dfrac{1}{1+x^2}$ over $n+1$ equally spaced nodes in the interval $I=[-5,5]$. In this case, the interpolating polynomials $\Pi_n f(x)$ of $f(x)$ exhibit the so called Runge phenomenon for increasing values of $n$ as it is visible in proximity of the boundaries of the interval $I$. Moreover, $\lim_{n\to+\infty} e_n(f)=+\infty$.




♦

**Remark 3.4** An other important issue which may arise with polynomial interpolation on *equally spaced* nodes concerns the *stability of the interpolating polynomial*. Indeed, using equally spaced $n+1$ nodes in the interval $I$ may lead to a significant sensitivity of the interpolating polynomial $\Pi_n(x)$, or $\Pi_n f(x)$ if the function $f(x)$ is known, to *perturbations* on the data.

♦ **Example 3.10** We highlight such stability issue by considering the polynomial interpolation of $f(x) = \sin(\pi x)$ over $n+1$ equally spaced nodes in the interval $I = [-1,1]$. By setting $n = 21$ we obtain the polynomial interpolant $\Pi_{21} f(x)$ which qualitatively coincides with $f(x)$. We apply now the polynomial interpolation to a perturbed function $f_\varepsilon(x) = f(x) + \varepsilon(x)$, with $\varepsilon(x)$ a random function such that $|\varepsilon(x)| < 10^{-3}$ for all $x \in I$. Its polynomial interpolant of degree $n = 21$ $\Pi_{21} f_\varepsilon(x)$ is very sensible to this "small" perturbation, being very different from $\Pi_{21} f(x)$.




♦

A remedy to mitigate the Runge phenomenon and stability issues for polynomial interpolation consists in using nodes which are *not* equally spaced in the interval $I$. The following definition provides a special family of nodes that can be used for polynomial interpolation.

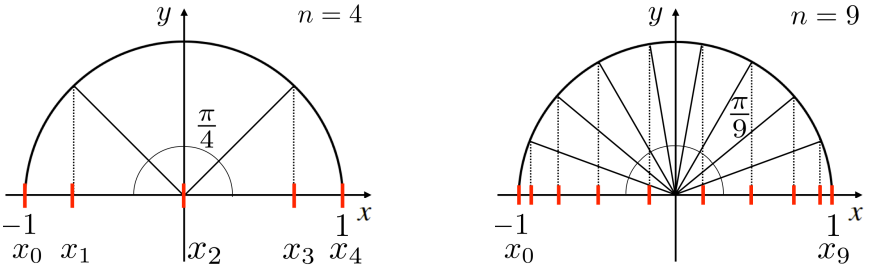
**Definition 3.5** For a given $n \geq 1$, the $n+1$ *Chebyshev–Gauss–Lobatto nodes* in the reference interval $\widehat{I} = [-1,1]$ are:

$$\widehat{x}_i = -\cos\left(\frac{\pi}{n} i\right) \qquad i = 0, \ldots, n;$$

in the general interval $I = [a,b]$ the $n+1$ Chebyshev–Gauss–Lobatto nodes are:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \widehat{x}_i \qquad i = 0, \ldots, n.$$

♦ **Example 3.11** We graphically highlight the $n+1$ Chebyshev–Gauss–Lobatto nodes $\{\widehat{x}_i\}_{i=0}^n$ in the reference interval $\widehat{I} = [-1,1]$ for $n = 4$ (left) and $n = 9$ (right).

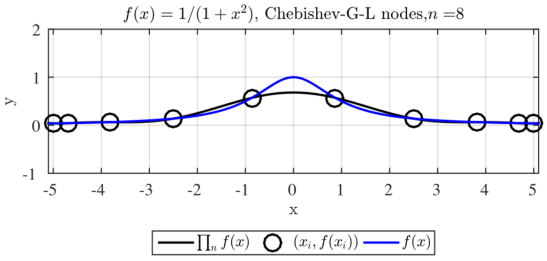



♦

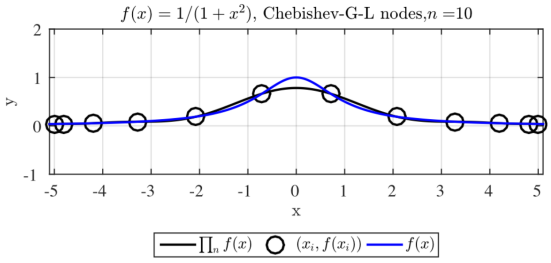
**Proposition 3.5** If $f \in C^1(I)$, with the interval $I = [a,b]$ and the $n+1$ Chebyshev–Gauss-Lobatto nodes are used in $I$, then $\lim_{n\to+\infty} \Pi_n f(x) = f(x)$ for all $x \in I$. Moreover, if $f \in C^\infty(I)$, then $\lim_{n\to+\infty} e_n(f) = 0$. Furthermore, the stability issues are mitigated.

♦ **Example 3.12** By recalling and using the same data of Examples 3.9 and 3.10, we show that the use of Chebyshev–Gauss–Lobatto (CGL) nodes avoids the insurgency of the Runge phenomenon and mitigates the stability issues of polynomial interpolation, respectively.

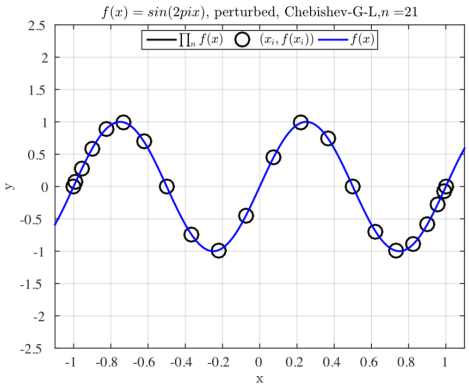
$n = 8$ with CGL nodes




$n = 10$ with CGL nodes




$\Pi_{21} f_\varepsilon(x)$ with CGL nodes




♦

As anticipated, the Lagrange interpolating polynomial $\Pi_n(x) \in \mathbb{P}_n$ uses the basis of Lagrange characteristic polynomials $\{\varphi_k(x)\}_{k=0}^n$ to determine the coefficients $\{a_i\}_{i=0}^n$ of this polynomial, i.e. $\Pi_n(x) = \sum_{k=0}^{n} y_k\, \varphi_k(x) = a_0 + a_1 x + \cdots + a_n x^n$. An alternative approach to Lagrange polynomial interpolation consists in computing directly the $n+1$ coefficients $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T \in \mathbb{R}^{n+1}$ by enforcing the $n+1$ interpolation constraints $\Pi_n(x_i) = y_i$ for all $i = 0, \ldots, n$; i.e., $\Pi_n(x_i) = a_0 + a_1 x_i + \cdots + a_n x_i^n = y_i$ for all $i = 0, \ldots, n$. The problem boils down to solve the following linear system:

$$B\,\mathbf{a} = \mathbf{y} \tag{3.4}$$

where $B \in \mathbb{R}^{(n+1)\times(n+1)}$ is the Vandermonde matrix, with $B_{ij} = (x_{i-1})^{j-1}$ for $i, j = 1, \ldots, n+1$, and $\mathbf{y} = (y_0, y_1, \ldots, y_n)^T \in \mathbb{R}^{n+1}$. The linear system (3.4) admits an unique solution if and only if $\det(B) \neq 0$, i.e. if and only if the $n+1$ nodes $\{x_i\}_{i=0}^n$ are distinct. We remark that interpolating polynomials computed by solving the linear system (3.4) may suffer stability issues already for relatively "small" values of $n$, due to the large conditioning numbers typically associated to the matrix $B$.

**Remark 3.5** The polynomial interpolation is in general not adequate to extrapolate information outside the interval $I$ containing the nodes (see e.g. Example 3.8).

### 3.2.2 Trigonometric interpolation

We consider the *trigonometric interpolation* which uses trigonometric basis functions and it is also referred as *discrete Fourier series*; such kind of interpolation is used for periodic signals and functions. With this aim, we consider a *periodic* function $f : [0,2\pi] \to \mathbb{C}$, i.e. such that $f(0) = f(2\pi)$; $\imath$ indicates the imaginary unit such that $\imath^2 = -1$.

**Definition 3.6** Given $n+1$ nodes $\{x_j\}_{j=0}^n$ such that $x_j = jh$ for $j = 0,\dots,n$ with $h = \dfrac{2\pi}{n}$, the trigonometric interpolant of the periodic function $f : [0,2\pi] \to \mathbb{C}$, say $I_t f(x)$, is:

$$I_t f(x) = \sum_{k=-(M+\mu)}^{M+\mu} \widetilde{c}_k e^{\imath k x},$$

where:

$$M = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (n-1)/2 & \text{if } n \text{ is odd} \end{cases}, \qquad \mu = \begin{cases} 0 & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases},$$

$$\widetilde{c}_k = \begin{cases} c_k & \text{for } k = -M,\dots,M \\ c_k/2 & \text{if } k = -(M+1) \text{ or } M+1 \end{cases},$$

and

$$c_k = \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) e^{-\imath k j h}.$$

The coefficients $\{c_k\}_{k=0}^n \in \mathbb{C}$ and $e^{\imath k x} = \cos(kx) + \imath \sin(kx)$. If $f(x)$ is a real valued function (i.e. $f(x) \in \mathbb{R}$ for all $x \in \mathbb{R}$), then also the trigonometric interpolant is real valued (i.e. $I_t f(x) \in \mathbb{R}$ for all $x \in \mathbb{R}$); indeed, in this case, $c_{-k} = \overline{c}_k$ for $k = 0,\dots,n$.

**Remark 3.6** The trigonometric interpolant $I_t f(x)$ interpolates $f(x)$ at the $n+1$ nodes $\{x_j\}_{j=0}^n$, indeed $I_t f(x_j) = f(x_j)$ for all $j = 0,\dots,n$.

The computation of the coefficients $\{c_k\}_{k=0}^n$ according to the above mentioned procedure requires $O(n^2)$ flops; instead, using the *Fast Fourier Transform* (FFT) only requires $O(n \log n)$ flops.

### 3.2.3 Piecewise polynomial interpolation

*Piecewise polynomial interpolation* is also known as *composite* interpolation and approximates a function $f(x)$ *locally* with polynomials. Piecewise polynomial interpolation is a good alternative to polynomial interpolation with equally spaced nodes to extract information inside an interval.

**Definition 3.7** Let us consider $n+1$ distinct nodes $\{x_i\}_{i=0}^n$ in the interval $I = [a,b]$ such that $a = x_0 < x_1 < \cdots < x_n = b$ for which $n$ subintervals $I_i = [x_i, x_{i+1}]$ are defined for $i = 0,\dots,n-1$; we indicate with $H := \max_{i=0,\dots,n-1} |I_i| = \max_{i=0,\dots,n-1} (x_{i+1} - x_i)$ the characteristic size of such subintervals. Given the set of data couples $\{(x_i, y_i)\}_{i=0}^n$, the *piecewise linear interpolating polynomial* $\Pi_1^H(x)$ of the data is a piecewise polynomial of degree 1 such that $\Pi_1^H(x) \in \mathbb{P}_1$ for all $x \in I_i$ and

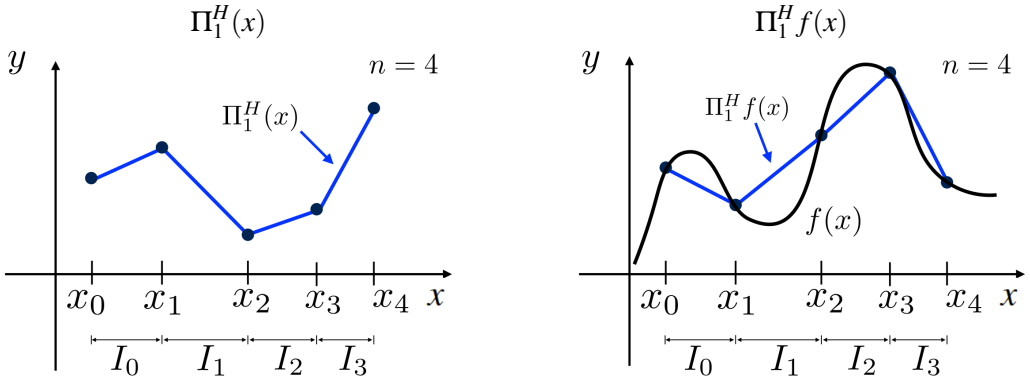
$i = 0, \ldots, n-1$ (i.e. $\Pi_1^H(x)|_{I_i} \in \mathbb{P}_1$ for all $i = 0, \ldots, n-1$), with:

$$\Pi_1^H(x)|_{I_i} = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \quad \text{for all } i = 0, \ldots, n-1.$$

If the function $f \in C^0(I)$ is known, then the *piecewise linear interpolating polynomial* $\Pi_1^H f(x)$ of the function $f(x)$ at the nodes is $\Pi_1^H f(x)|_{I_i} \in \mathbb{P}_1$ for all $i = 0, \ldots, n-1$, with:

$$\Pi_1^H f(x)|_{I_i} = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) \quad \text{for all } i = 0, \ldots, n-1.$$

♦ **Example 3.13** We report the piecewise linear interpolants of $n+1$ data couples, say $\Pi_1^H(x)$, and of a continuous function $f(x)$, say $\Pi_1^H f(x)$, in $n+1$ nodes in the interval $I$; specifically, $n = 4$ (i.e. $n+1 = 5$ nodes are used). The characteristic size of the subintervals $\{I_i\}_{i=0}^3$ is $H = \max_{i=0,1,2,3} |I_i|$.




♦

**Definition 3.8** If the function $f \in C^0(I)$ is known, we define the *error* associated to the piecewise linear interpolating polynomial $\Pi_1^H f(x)$ as $e_1^H(f) := \max_{x \in I} \left| f(x) - \Pi_1^H f(x) \right|$.

**Proposition 3.6** If $f \in C^2(I)$, then the error $e_1^H(f)$ associated to the piecewise linear interpolating polynomial $\Pi_1^H f(x)$ can be bounded by the error estimator $\widetilde{e}_1^H(f)$ as:

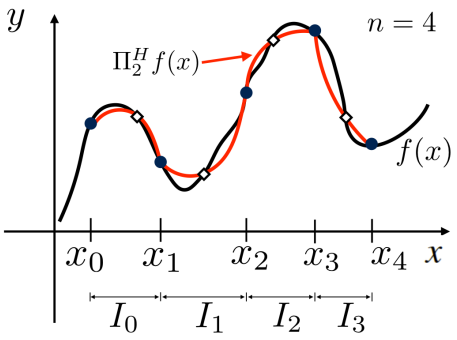
$$e_1^H(f) \leq \widetilde{e}_1^H(f) := \frac{H^2}{8} \max_{x \in I} \left| f''(x) \right|,$$

for which the error *converges* to zero with *order* 2 in $H$ (quadratically).

In analogy with $\Pi_1^H(x)$, one can define the *piecewise quadratic polynomial* $\Pi_2^H(x)$ as $\Pi_2^H(x)|_{I_i} \in \mathbb{P}_2$ for all the subintervals $I_i$ of $I$ from $i = 0, \ldots, n-1$; if $f \in C^0(I)$ is known, then we use the notation $\Pi_2^H f(x)$. Similarly, one can define the *piecewise interpolating polynomial of degree* $r \geq 1$ $\Pi_r^H(x)$ as $\Pi_r^H(x)|_{I_i} \in \mathbb{P}_r$ for all $i = 0, \ldots, n-1$ (or $\Pi_r^H f(x)$ if $f \in C^0(I)$ is known).

♦ **Example 3.14** We consider the piecewise quadratic interpolation of a continuous function $f(x)$, say $\Pi_2^H f(x)$, over $n+1$ nodes in the interval $I$; specifically, we set $n = 4$.

The piecewise quadratic interpolant $\Pi_2^H f(x)$ interpolates $f(x)$ at the $n+1=5$ nodes and at intermediate points internal to each subinterval of $I$ (as for example corresponding to the mid–points of the subintervals).




♦

**Proposition 3.7** If $f \in C^{r+1}(I)$, then the error $e_r^H(f) := \max_{x\in I} \left| f(x) - \Pi_r^H f(x) \right|$ associated to the piecewise interpolating polynomial of degree $r \geq 1$ $\Pi_r^H f(x)$ can be bounded by the error estimator $\widetilde{e}_r^H(f)$ as:

$$e_r^H(f) \leq \widetilde{e}_r^H(f) := C_r H^{r+1} \max_{x\in I} \left| f^{(r+1)}(x) \right|,$$

with $C_r$ a positive constant, for which the error *converges* to zero with *order* $r+1$ in $H$.

**Remark 3.7** Piecewise interpolating polynomials $\Pi_r^H f(x)$ of any degree $r \geq 1$ are only $C^0$–continuous across the subintervals (internal nodes); see e.g. Example 3.14.

### 3.2.4 Spline functions

Spline functions, or simply *splines*, are piecewise interpolating polynomials which are smoother than standard piecewise polynomials $\Pi_r^H f(x)$ across the subintervals. Splines and their generalizations (B–splines and NURBS) are widely used in Computer Graphics and industrial applications for which high regularity of the interpolants is necessary.

**Definition 3.9** A *cubic spline function*, say $s_3(x)$, is a piecewise interpolating polynomial of degree 3 which is $C^2$–continuous across the internal nodes of the interval $I$. Specifically, given $n+1$ nodes in $I = [x_0, x_n]$ with $x_0 < x_1 < \cdots < x_n$ and the subintervals $I_i = [x_i, x_{i+1}]$ for $i = 0, \ldots, n-1$, we have:

$s_3(x)|_{I_i} \in \mathbb{P}_3$ for all $i = 0, \ldots, n-1$ and $s_3^{(k)}(x_i^-) = s_3^{(k)}(x_i^+)$ for all $i = 1, \ldots, n-1$ and k = 0,1,2.

We can write $s_3(x)|_{I_i} = a_{0,i} + a_{1,i}x + a_{2,i}x^2 + a_{3,i}x^3$ for all $i = 0, \ldots, n-1$, with the $4n$ coefficients $\{a_{j,i}\}$ to be determined for $j = 0,1,2,3$ and $i = 0, \ldots, n-1$. In order to obtain the coefficients of $s_3(x)$ we impose the following $4n-2$ constraints according to Definition 3.9:

$$s_3(x_i) = y_i \quad (\text{or } s_3(x_i) = f(x_i) \text{ if } f \text{ is known}) \qquad \text{for all } i = 0, \ldots, n,$$

$$s_3(x_i^-) = s_3(x_i^+), \quad s_3'(x_i^-) = s_3'(x_i^+), \quad s_3''(x_i^-) = s_3''(x_i^+) \qquad \text{for all } i = 1, \ldots, n-1.$$

In order to fully determine $s_3(x)$ one needs to enforce 2 additional constraints, whose choice determine the type of cubic spline.

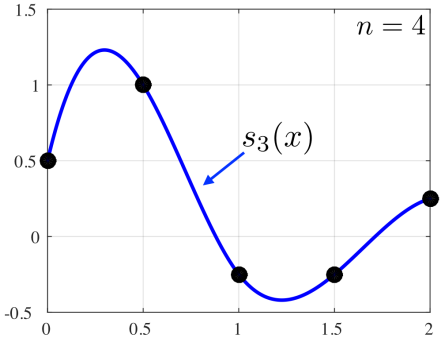
**Definition 3.10** If one sets $s_3''(x_0) = s_3''(x_n) = 0$, then $s_3(x)$ is a *natural interpolating cubic spline*.

If one sets $s_3'''(x_1^-) = s_3'''(x_1^+)$ and $s_3'''(x_{n-1}^-) = s_3'''(x_{n-1}^+)$, then $s_3(x)$ is a *not–a–knot interpolating cubic spline*.

The MATLAB command `spline` considers not–a–knot interpolating cubic splines.

♦ **Example 3.15** We consider the interpolation of a set of $n+1$ data by a not–a–knot interpolating cubic spline $s_3(x)$; specifically, $n = 4$.

The cubic spline $s_3(x)$ interpolates the data at the $n+1 = 5$ nodes $\{x_i\}_{i=0}^4$ and is $C^2$–continuous across each internal node $\{x_i\}_{i=1}^3$.




♦

**Proposition 3.8** Let us consider $n+1$ distinct nodes $\{x_i\}_{i=0}^n$ delimiting the interval $I = [x_0, x_n]$ for which $n$ subintervals $I_i = [x_i, x_{i+1}]$ are defined for $i = 0, \ldots, n-1$ and $H := \max_{i=0,\ldots,n-1} |I_i|$ is the characteristic size of such subintervals. Then, if $f \in C^4(I)$ and $s_3(x)$ is its *natural interpolating cubic spline* at the nodes, we have the following error estimates:

$$\max_{x\in I} \left| f^{(k)}(x) - s_3^{(k)}(x) \right| \leq C_k H^{4-k} \max_{x\in I} \left| f^{(4)}(x) \right| \qquad \text{for } k = 0, 1, 2$$

and

$$\max_{x\in I\setminus\{x_1,\ldots,x_{n-1}\}} \left| f'''(x) - s_3'''(x) \right| \leq C_3 \, H \, \max_{x\in I} \left| f^{(4)}(x) \right|,$$

with $C_k > 0$ positive constants, for which the *convergence order* of the error is $4-k$ in $H$ depending on the order of derivation $k = 0, 1, 2, 3$.

## 3.3 Least–Squares Method

The *least–squares approximation* is ideal to extract information from a large set of data, both with and without uncertainty and noise, as well as to make predictions outside the interval in which these data are available.

**Definition 3.11** Given the set of data couples $\{(x_i, y_i)\}_{i=0}^n$ (or $\{(x_i, f(x_i))\}_{i=0}^n$ if the function $f(x)$ is given) and an integer $m \geq 0$, we look for an approximating polynomial $\widetilde{f}_m(x)$ of degree $m$ such that:

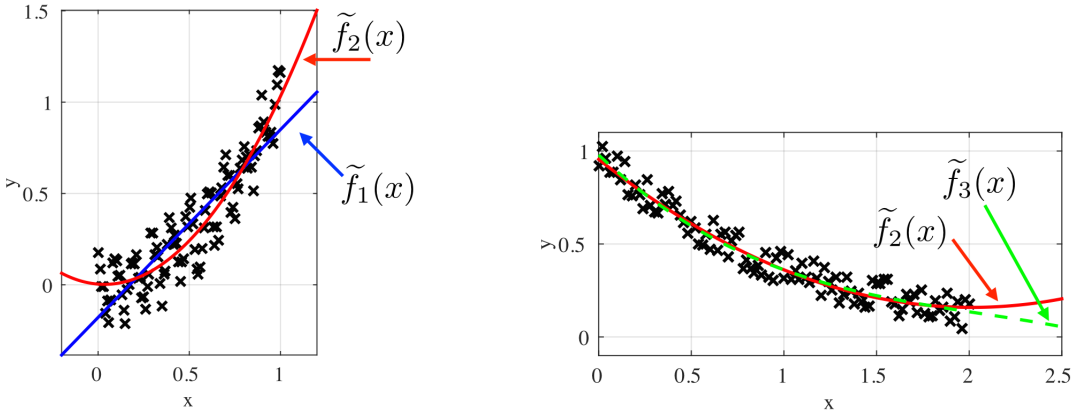
$$\sum_{i=0}^{n} \left( y_i - \widetilde{f}_m(x_i) \right)^2 \leq \sum_{i=0}^{n} \left( y_i - p_m(x_i) \right)^2 \qquad \text{for all } p_m \in \mathbb{P}_m.$$

If $\widetilde{f}_m \in \mathbb{P}_m$ exists, then it is called *least–squares approximating polynomial of degree m* of the data (or the function $f(x)$).

**Remark 3.8** By convention, we assume that the nodes $\{x_i\}_{i=0}^n$ are distinct and $0 \leq m \leq n$. In a typical scenario in which the least–squares method is used, one has $0 \leq m \ll n$.

**Remark 3.9** The least–squares approximating polynomial $\widetilde{f}_m(x)$ does not in general interpolate the data (or the function $f(x)$) at the nodes. Specifically, only if $m = n$ one can ensure that $\widetilde{f}_m(x_i) = y_i$ (or $\widetilde{f}_m(x_i) = f(x_i)$) for all $i = 0, \ldots, n$; indeed, in this case, $\widetilde{f}_m(x)$ coincides with the polynomial interpolant $\Pi_n(x)$ of degree $n$ (or $\Pi_n f(x)$).

♦ **Example 3.16** We graphically represent the least–squares approximating polynomials $\widetilde{f}_m(x)$ of degrees $m$ for two relatively large sets of data $\{(x_i, y_i)\}_{i=0}^n$, with $n = 100$. We consider $m = 1$ and 2 (left) and $m = 2$ and 3 (right).




♦

**Definition 3.12** Following Definition 3.11, the least–squares approximating polynomial $\widetilde{f}_1(x)$ of degree $m = 1$ is called *regression line* or least–squares straight line.

As for the polynomial interpolation, determining the least–squares approximating polynomial $\widetilde{f}_m(x)$ of degree $m$ consists in determining its $m+1$ coefficients $\{a_i\}_{i=0}^m$; indeed, $\widetilde{f}_m(x) = a_0 + a_1 x + \cdots + a_m x^m$. With this aim, we define the coefficients vector $\mathbf{a} = (a_0, a_1, \ldots, a_m)^T \in \mathbb{R}^{m+1}$ and the functional $\Phi : \mathbb{R}^{m+1} \to \mathbb{R}$ as:

$$\Phi(\mathbf{b}) = \sum_{i=0}^{n} [y_i - (b_0 + b_1 x_i + \cdots + b_m x_i^m)]^2,$$

which is associated to the set of data couples $\{(x_i, y_i)\}_{i=0}^n$ for the general coefficient vector $\mathbf{b} = (b_0, b_1, \ldots, b_m)^T \in \mathbb{R}^{m+1}$. Then, the *least–squares method* consists in determining the coefficients vector $\mathbf{a}$ of the polynomial $\widetilde{f}_m(x)$ of degree $m$ such that:

$$\Phi(\mathbf{a}) = \min_{\mathbf{b} \in \mathbb{R}^{m+1}} \Phi(\mathbf{b}).$$

Since $\Phi$ is differentiable, the previous minimization problem is equivalent to solve the following differential problem:

$$\text{find } \mathbf{a} \in \mathbb{R}^{m+1} \quad : \quad \frac{\partial \Phi}{\partial b_j}(\mathbf{a}) = 0 \qquad \text{for all } j = 0, \ldots, m. \tag{3.5}$$

In turn, such differential problem reduces into solving the following linear system:

$$A\,\mathbf{a} = \mathbf{q}, \tag{3.6}$$

where $A \in \mathbb{R}^{(m+1)\times(m+1)}$ and $\mathbf{q} \in \mathbb{R}^{m+1}$ read:

$$A = \begin{bmatrix} (n+1) & \sum_{i=0}^{n} x_i & \cdots & \sum_{i=0}^{n} x_i^m \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \cdots & \sum_{i=0}^{n} x_i^{m+1} \\ \vdots & \vdots & & \vdots \\ \sum_{i=0}^{n} x_i^m & \sum_{i=0}^{n} x_i^{m+1} & \cdots & \sum_{i=0}^{n} x_i^{2m} \end{bmatrix} \quad \text{and} \quad \mathbf{q} = \begin{bmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i y_i \\ \vdots \\ \sum_{i=0}^{n} x_i^m y_i \end{bmatrix},$$

respectively. By recalling the Vandermonde matrix $B \in \mathbb{R}^{(n+1)\times(m+1)}$, with $B_{ij} = (x_{i-1})^{j-1}$ for $i = 1,\ldots,n+1$ and $j = 1,\ldots,m+1$, and the data vector $\mathbf{y} = (y_0, y_1, \ldots, y_n)^T \in \mathbb{R}^{n+1}$, we observe that:

$$A = B^T B \qquad \text{and} \qquad \mathbf{q} = B^T \mathbf{y},$$

respectively.

**Remark 3.10** The linear system (3.6) is a generalization of the linear system (3.4) used for the polynomial interpolation. As a matter of fact, if the nodes are distinct and $m = n$, we obtain that $\widetilde{f}_m(x) = \Pi_n(x)$ and solving the linear systems (3.4) and (3.6) yields equivalent results.

♦ **Example 3.17** We illustrate the derivation of the linear system (3.6) for $m = 1$ (i.e. $\widetilde{f}_1(x)$ is the regression line) and $n \geq m$. In this case, the functional $\Phi(\mathbf{b})$ reads:

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

In order to formulate the problem as in Eq. (3.5), we compute the partial derivatives of $\Phi$:

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

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

Then, problem (3.5) can be written as the linear system (3.6) with $A \in \mathbb{R}^{2\times 2}$ and $\mathbf{q} \in \mathbb{R}^2$, being:

$$A = \begin{bmatrix} (n+1) & \sum_{i=0}^{n} x_i \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 \end{bmatrix} \quad \text{and} \quad \mathbf{q} = \begin{bmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i y_i \end{bmatrix},$$

respectively. Notice that, in this case, the Vandermonde matrix $B \in \mathbb{R}^{(n+1)\times 2}$ reads:

$$B = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}.$$

♦

# 4. Numerical Differentiation

We briefly consider the approximation of derivatives of functions, a topic commonly referred as *numerical derivation* or *numerical differentiation*.

## 4.1 Goal and Examples

Given a function $f \in C^1((a,b))$, we aim at approximating $f'(\bar{x})$ for some $\bar{x} \in (a,b) \subseteq \mathbb{R}$. Indeed, for a known function $f(x)$, one may prefer to not explicitly evaluate $f'(\bar{x})$ for some $\bar{x} \in (a,b)$ since this could be computationally expensive.

♦ **Example 4.1** Numerical derivation is necessary if only the set of data couples $\{(x_i, y_i)\}_{i=0}^n$ is provided and not the function $f(x)$ from which these are eventually obtained. In such case, one may still be interested in providing information on the first derivative of the unknown function $f(x)$ at one or all the nodes $\{x_i\}_{i=0}^n$. ♦

## 4.2 Finite Differences Schemes

We consider some *finite differences* schemes for the approximation of $f'(\bar{x})$ for some $\bar{x} \in (a,b)$.

### 4.2.1 Forward and backward finite differences

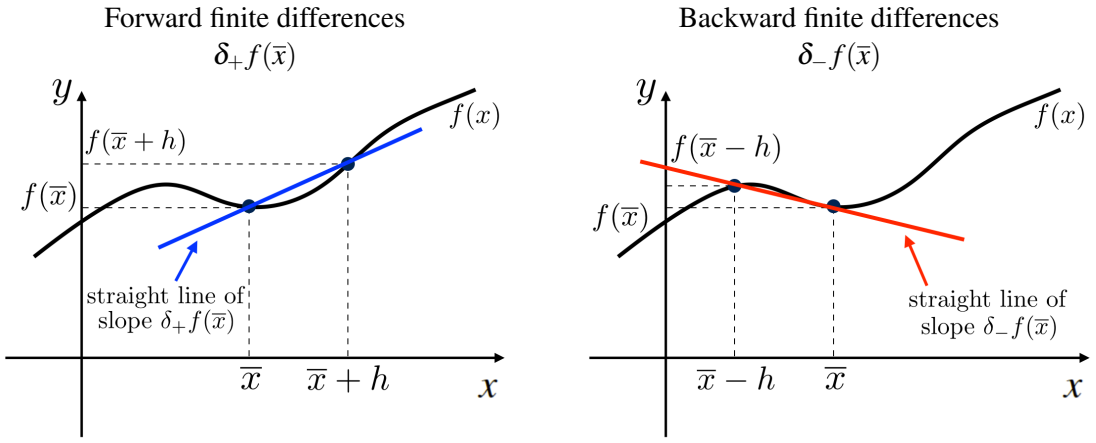
**Definition 4.1** Given a function $f(x)$ and the step size $h > 0$, the approximation of $f'(\bar{x})$ in some $\bar{x} \in (a,b) \subseteq \mathbb{R}$ by the *forward finite differences* scheme is defined as:

$$\delta_+ f(\bar{x}) := \frac{f(\bar{x}+h) - f(\bar{x})}{h},$$

while by the *backward finite differences* scheme as:

$$\delta_- f(\bar{x}) := \frac{f(\bar{x}) - f(\bar{x}-h)}{h}.$$

♦ **Example 4.2** We graphically illustrate the forward (left) and backward (right) finite differences schemes for the approximation of $f'(\bar{x})$ for some $h > 0$; with this aim, we plot the straight lines of slopes $\delta_+ f(\bar{x})$ and $\delta_- f(\bar{x})$, respectively.




♦

**Proposition 4.1** If $f \in C^2((a,b))$ and $\bar{x} \in (a,b)$, the *error* $E_+ f(\bar{x})$ associated to the *forward finite differences* scheme is:

$$E_+ f(\bar{x}) := f'(\bar{x}) - \delta_+ f(\bar{x}) = -\frac{1}{2} h f''(\xi_+) \qquad \text{for some } \xi_+ \in [\bar{x}, \bar{x}+h],$$

while the *error* $E_+ f(\bar{x})$ associated to the *backward finite differences* scheme reads:

$$E_- f(\bar{x}) := f'(\bar{x}) - \delta_- f(\bar{x}) = \frac{1}{2} h f''(\xi_-) \qquad \text{for some } \xi_- \in [\bar{x}-h, \bar{x}].$$

*Proof.* (Forward finite differences scheme). We consider the Taylor's expansion of $f(\bar{x}+h)$ around $\bar{x}$, obtaining $f(\bar{x}+h) = f(\bar{x}) + f'(\bar{x})h + \frac{1}{2} f''(\xi_+)h^2$ for some $\xi_+ \in [\bar{x}, \bar{x}+h]$; by applying the definition of the error $E_+ f(\bar{x})$, the result follows. The proof for backward finite differences follows in a similar manner. ∎

**Remark 4.1** The forward and backward finite differences schemes are methods of *order* 1; indeed, the errors $E_+ f(\bar{x})$ and $E_- f(\bar{x})$ *converge* to zero with *order* 1 in the step size $h$.

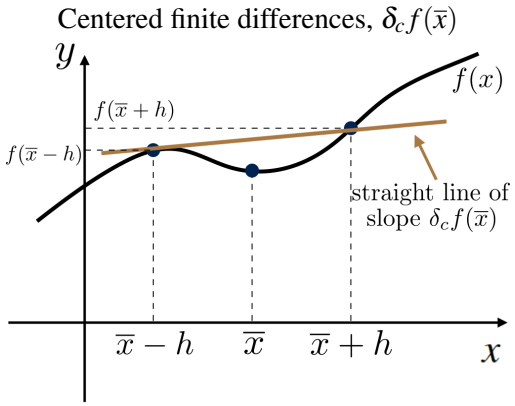
We observe that, if $f \in \mathbb{P}_1$, we have $\delta_+ f(\bar{x}) = \delta_- f(\bar{x}) = f'(\bar{x})$ for all $\bar{x} \in \mathbb{R}$.

### 4.2.2 Centered finite differences

**Definition 4.2** Given a function $f(x)$ and the step size $h > 0$, the approximation of $f'(\bar{x})$ in some $\bar{x} \in (a,b) \subseteq \mathbb{R}$ by the *centered finite differences* scheme is defined as:

$$\delta_c f(\bar{x}) := \frac{f(\bar{x}+h) - f(\bar{x}-h)}{2h}.$$

♦ **Example 4.3** We graphically illustrate the centered finite differences scheme for the approximation of $f'(\bar{x})$ for some $h > 0$; specifically, we plot the straight line of slope $\delta_c f(\bar{x})$.




♦

**Proposition 4.2** If $f \in C^3((a,b))$ and $\bar{x} \in (a,b)$, the *error* $E_c f(\bar{x})$ associated to the *centered finite differences* scheme is:

$$E_c f(\bar{x}) := f'(\bar{x}) - \delta_c f(\bar{x}) = -\frac{1}{12} h^2 \left[ f'''(\xi_+) + f'''(\xi_-) \right]$$

for some $\xi_+ \in [\bar{x}, \bar{x}+h]$ and $\xi_- \in [\bar{x}-h, \bar{x}]$.

*Proof.* We consider the Taylor's expansion of $f(\bar{x}+h)$ around $\bar{x}$, obtaining $f(\bar{x}+h) = f(\bar{x}) + f'(\bar{x})h + \frac{1}{2}f''(\bar{x})h^2 + \frac{1}{6}f'''(\xi_+)h^3$ for some $\xi_+ \in [\bar{x}, \bar{x}+h]$; similarly, the Taylor's expansion of $f(\bar{x}-h)$ around $\bar{x}$ is $f(\bar{x}-h) = f(\bar{x}) - f'(\bar{x})h + \frac{1}{2}f''(\bar{x})h^2 - \frac{1}{6}f'''(\xi_-)h^3$ for some $\xi_- \in [\bar{x}-h, \bar{x}]$. Then, by applying the definition of the error $E_c f(\bar{x})$, the result follows. ∎

**Remark 4.2** The centered finite differences scheme is a method of *order* 2; indeed, the error $E_c f(\bar{x})$ *converges* to zero with *order* 2 in the step size $h$.

We observe that, if $f \in \mathbb{P}_2$, we have $\delta_c f(\bar{x}) = f'(\bar{x})$ for all $\bar{x} \in \mathbb{R}$.

Let us now suppose to be interested in approximating the first derivative of a function $f(x)$ in *multiple* and equally spaced *nodes* in the interval $[a,b]$, that is $x_i = a + ih$ for all $i = 0, \ldots, n$, with $h = \frac{b-a}{n}$; $x_0 = a$ and $x_n = b$ are the nodes at the boundaries of the interval $[a,b]$. With this aim, we consider the centered finite differences scheme. However, this can only be used at the nodes internal to the interval $[a,b]$ as:

$$\delta_c f(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \qquad \text{for all } i = 1, \ldots, n-1,$$

since for $x_0$ and $x_n$ the nodes $x_{-1}$ and $x_{n+1}$ are not defined, respectively. In order to approximate $f'(x_0)$ and $f'(x_n)$, the forward and backward finite differences schemes can be used as $\delta_+ f(x_0) = \frac{f(x_1) - f(x_0)}{h}$ and $\delta_- f(x_n) = \frac{f(x_n) - f(x_{n-1})}{h}$, respectively. However, in this case, the approximation uses a method of order 2 in the internal nodes $\{x_i\}_{i=1}^{n-1}$ and of order 1 in the nodes $x_0$

and $x_n$. In order to overcome this issue and restore the full convergence order 2 for all the nodes $\{x_i\}_{i=0}^n$, one can use the following finite differences schemes in $x_0$ and $x_n$:

$$\delta_{c,0} f(x_0) = \frac{-3f(x_0) + 4f(x_1) - f(x_2)}{2h}$$

and

$$\delta_{c,n} f(x_n) = \frac{f(x_{n-2}) - 4f(x_{n-1}) + 3f(x_n)}{2h},$$

respectively, which yield methods of convergence order 2 in $h$. We observe that $\delta_{c,0} f(x_0) = \left(\Pi_{2,\{x_0,x_1,x_2\}} f\right)'(x_0)$ and $\delta_{c,n} f(x_n) = \left(\Pi_{2,\{x_{n-2},x_{n-1},x_n\}} f\right)'(x_n)$, where $\Pi_{2,\{x_0,x_1,x_2\}} f(x)$ and $\Pi_{2,\{x_{n-2},x_{n-1},x_n\}} f(x)$ are the polynomials of degree 2 interpolating the function $f(x)$ at the nodes $\{x_i\}_{i=0}^2$ and $\{x_i\}_{i=n-2}^n$, respectively.

# 5. Numerical Integration

We consider the numerical approximation of integrals of real valued functions by means of the so called *quadrature formulas* (*numerical integration*).

## 5.1 Goal and Classification of Quadrature Formulas

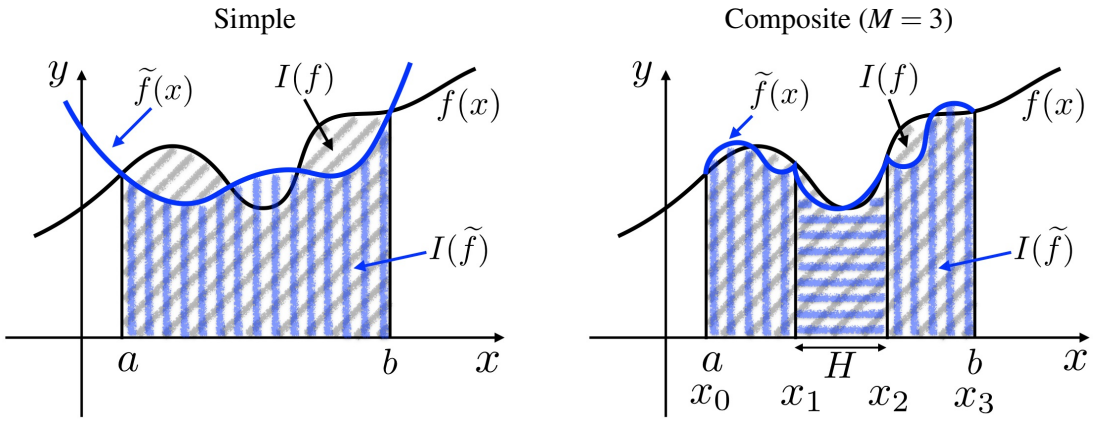
Given a function $f \in C^0([a,b])$, the goal consists in numerically approximating its integral in the interval $[a,b]$, say

$$I(f) = \int_a^b f(x)\,dx,$$

by means of suitable *quadrature formulas*, say $I_q(f)$ such that $I_q(f) \simeq I(f)$. Quadrature formulas can be classified into two main categories: *simple* and *composite* quadrature formulas.

- *Simple quadrature formulas* are based of the approximation of the function $f(x)$ globally in the interval $[a,b]$ with functions $\widetilde{f}(x)$ which are "easy" to integrate in $[a,b]$ (integrable in closed form); that is, $I_q(f) = I(\widetilde{f}) = \int_a^b \widetilde{f}(x)\,dx$, with $\widetilde{f}(x)$ an approximation of $f(x)$ for $x \in [a,b]$. Typically, for simple quadrature formulas, $\widetilde{f}(x)$ is a polynomial of degree $n$ interpolating $f(x)$ at $n+1$ nodes in $[a,b]$.
- *Composite quadrature formulas* are based on the subdivision of the interval $[a,b]$ into $M$ subintervals, eventually of the same size $H = \dfrac{b-a}{M}$, over which the function $f(x)$ is approximated by a *piecewise* function $\widetilde{f}(x)$. By indicating the $M+1$ nodes $\{x_k\}_{k=0}^M$ as $x_k = a + kH$ for $k = 0,\ldots,M$, with $x_0 = a$ and $x_M = b$, we recall that $I(f) = \sum_{k=1}^M \int_{x_{k-1}}^{x_k} f(x)\,dx$; then, the composite quadrature formula reads $I_q(f) = \sum_{k=1}^M \int_{x_{k-1}}^{x_k} \widetilde{f}_k(x)\,dx$, where $\widetilde{f}_k(x) = \widetilde{f}(x)|_{[x_{k-1},x_k]}$ for all $k = 1,\ldots,M$. Typically, for composite quadrature formulas, $\widetilde{f}(x)$ is a piecewise polynomial of degree $n$ interpolating $f(x)$ at $n+1$ nodes in each of the subintervals $\{[x_{k-1},x_k]\}_{k=1}^M$ of $[a,b]$.

♦ **Example 5.1** We graphically illustrate hypothetical simple (left) and composite (right) quadrature formulas for the approximation of the integral $I(f) = \int_a^b f(x)\,dx$ of a general function $f(x)$ in the interval $[a,b]$; $I(\widetilde{f})$ represents the approximated integral.




♦

The following definitions are used to characterize the quadrature formulas.

**Definition 5.1** The *degree of exactness* of a quadrature formula is the maximum integer $r \geq 0$ such that all the polynomials of degree less than or equal to $r$ are exactly integrated by the formula, i.e. for which $I_q(p) \equiv I(p)$ for all $p \in \mathbb{P}_r$.

**Definition 5.2** The *order of convergence* of a *composite* quadrature formula (also called *order of accuracy*) is the convergence order of the associated error in $H$, the size of the subintervals.

## 5.2 Mid–point Quadrature Formulas

**Definition 5.3** Let us consider a function $f(x) \in C^0([a,b])$, then the *simple mid–point quadrature formula* is defined as:

$$I_{mp}(f) := I(\Pi_0 f) = (b-a)\, f\left(\frac{a+b}{2}\right),$$

where $\Pi_0 f(x)$ is the polynomial of degree 0 interpolating $f(x)$ at the mid–point $\bar{x} = \dfrac{a+b}{2}$ of the interval $[a,b]$. The *composite mid–point quadrature formula* is defined as:

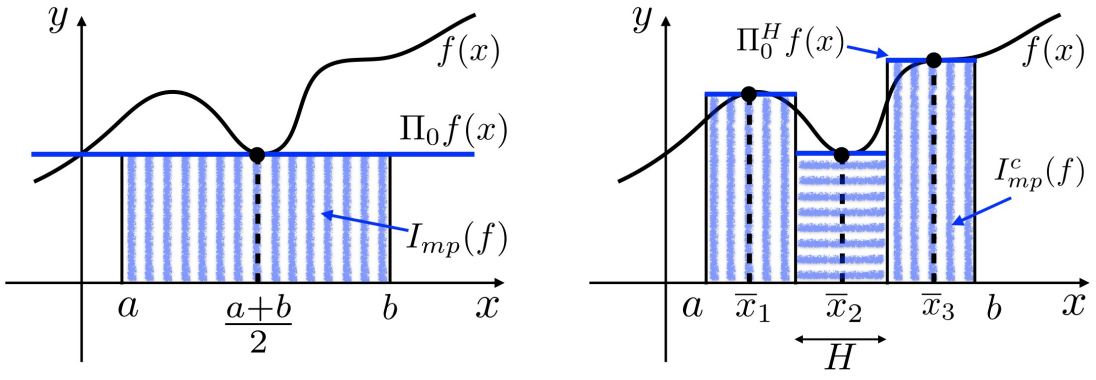
$$I^c_{mp}(f) := I\left(\Pi_0^H f\right) = H \sum_{k=1}^{M} f(\bar{x}_k), \tag{5.1}$$

where $\Pi_0^H f(x)$ is the piecewise polynomial of degree 0 interpolating $f(x)$ at the mid–points $\{\bar{x}_k\}_{k=1}^M$ of the $M$ subintervals of size $H = \dfrac{b-a}{M}$ in which $[a,b]$ is subdivided, i.e. $\bar{x}_k = \dfrac{x_{k-1}+x_k}{2}$ for all $k = 1,\ldots,M$.

The function $f(x)$ is approximated with $\widetilde{f}(x) = \Pi_0 f(x)$ or $\Pi_0^H f(x)$, for which $\int_a^b \widetilde{f}(x)\,dx$ yields the simple and composite quadrature formulas, respectively.

♦ **Example 5.2** We graphically illustrate simple (left) and composite (right) mid–point quadrature formulas for the approximation of the integral $I(f)$ of a general function $f(x)$.

Simple, $I_{mp}(f)$ Composite, $I^c_{mp}(f)$ $(M=3)$




♦

**Proposition 5.1** If $f \in C^2([a,b])$, the *error* $e_{mp}(f)$ associated to the *simple mid–point quadrature formula* reads:

$$e_{mp}(f) := I(f) - I_{mp}(f) = \frac{(b-a)^3}{24} f''(\xi) \qquad \text{for some } \xi \in [a,b],$$

while the *error* $e^c_{mp}(f)$ associated to the *composite mid–point quadrature formula* is:

$$e^c_{mp}(f) := I(f) - I^c_{mp}(f) = \frac{(b-a)}{24} H^2 f''(\xi) \qquad \text{for some } \xi \in [a,b].$$

*Proof.* (Simple mid–point formula). By indicating for simplicity the mid–point of $[a,b]$ as $\bar{x} = \dfrac{a+b}{2}$, we consider the Taylor's expansion of $f(x)$ around $\bar{x}$, which reads $f(x) = f(\bar{x}) + f'(\bar{x})(x-\bar{x}) + \dfrac{1}{2} f''(\eta(x))(x-\bar{x})^2$ for some $\eta(x) \in [a,b]$. We compute the integral of such expansion, obtaining $I(f) = I_{mp}(f) + f'(\bar{x}) \displaystyle\int_a^b (x-\bar{x})\,dx + \frac{1}{2} f''(\xi) \int_a^b (x-\bar{x})^2\,dx$, for some $\xi \in [a,b]$, in virtue of the mean value (Lagrange) theorem. Since $\displaystyle\int_a^b (x-\bar{x})\,dx = 0$ and $\displaystyle\int_a^b (x-\bar{x})^2\,dx = \frac{(b-a)^3}{12}$, the result follows. ∎

**Remark 5.1** The mid–point quadrature formulas have *degree of exactness* 1; indeed, the errors $e_{mp}(f)$ and $e^c_{mp}(f)$ are identically zero for all the polynomials of degree less than or equal to 1 (if $f \in \mathbb{P}_1$, $f''(\xi) = 0$ for all $\xi \in \mathbb{R}$).

**Remark 5.2** The composite mid–point quadrature formula has *convergence order* (*order of accuracy*) 2; indeed, the error $e^c_{mp}(f)$ depends on $H^2$.

## 5.3 Trapezoidal Quadrature Formulas

**Definition 5.4** Let us consider a function $f(x) \in C^0([a,b])$, then the *simple trapezoidal quadrature formula* is defined as:

$$I_t(f) := I(\Pi_1 f) = (b-a)\frac{f(a)+f(b)}{2}, \tag{5.2}$$

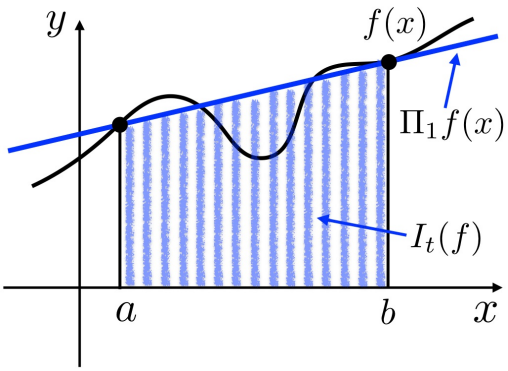
where $\Pi_1 f(x)$ is the polynomial of degree 1 interpolating $f(x)$ at the nodes $a$ and $b$. The *composite trapezoidal quadrature formula* is defined as:

$$I_t^c(f) := I\left(\Pi_1^H f\right) = \frac{H}{2}\sum_{k=1}^{M}[f(x_{k-1})+f(x_k)] = \frac{H}{2}[f(x_0)+f(x_M)]+H\sum_{k=1}^{M-1}f(x_k),$$

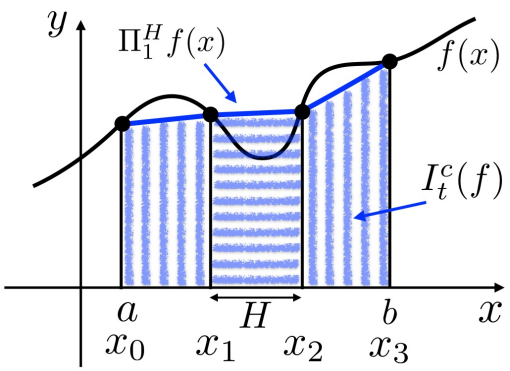
where $\Pi_1^H f(x)$ is the piecewise polynomial of degree 1 interpolating $f(x)$ at the nodes $\{x_k\}_{k=0}^M$ of the $M$ subintervals of size $H = \dfrac{b-a}{M}$ in which $[a,b]$ is subdivided.

♦ **Example 5.3** We graphically illustrate simple (left) and composite (right) trapezoidal quadrature formulas for the approximation of the integral $I(f)$ of a general function $f(x)$.

Simple, $I_t(f)$




Composite, $I_t^c(f)$ $(M=3)$




♦

**Proposition 5.2** If $f \in C^2([a,b])$, the *error* $e_t(f)$ associated to the *simple trapezoidal quadrature formula* reads:

$$e_t(f) := I(f) - I_t(f) = -\frac{(b-a)^3}{12}f''(\xi) \qquad \text{for some } \xi \in [a,b],$$

while the *error* $e_t^c(f)$ associated to the *composite trapezoidal quadrature formula* is:

$$e_t^c(f) := I(f) - I_t^c(f) = -\frac{(b-a)}{12}H^2 f''(\xi) \qquad \text{for some } \xi \in [a,b].$$

*Proof.* (Simple trapezoidal formula). Since $I_t(f) = I(\Pi_1 f)$, we have $e_t(f) = \displaystyle\int_a^b (f(x) - \Pi_1 f(x))\,dx$. By recalling the error function $E_1 f(x)$ of Eq. (3.1) for polynomial interpolation of degree 1 (Proposition 3.2), we have $E_1 f(x) = \dfrac{1}{2}f''(\eta(x))\,\omega_1(x)$ for some $\eta(x) \in [a,b]$, with $\omega_1(x) = (x-a)(x-b)$. By using the mean value (Lagrange) theorem, we have $e_t(f) = \dfrac{1}{2}\displaystyle\int_a^b f''(\eta(x))\,\omega_1(x)\,dx =$

$\frac{1}{2}f''(\xi)\int_a^b \omega_1(x)\,dx$, for some $\xi \in [a,b]$; then, since $\int_a^b \omega_1(x)\,dx = -\frac{(b-a)^3}{6}$, the result follows. ∎

**Remark 5.3** The trapezoidal quadrature formulas have *degree of exactness* 1; indeed, the errors $e_t(f)$ and $e_t^c(f)$ are identically zero for all the polynomials of degree less than or equal to 1 (if $f \in \mathbb{P}_1$, $f''(\xi) = 0$ for all $\xi \in \mathbb{R}$).

**Remark 5.4** The composite trapezoidal quadrature formula has *convergence order* (*order of accuracy*) 2; indeed, the error $e_t^c(f)$ depends on $H^2$.

**Remark 5.5** We consider the mid–point and trapezoidal quadrature formulas, for which the errors $e_{mp}(f)$, $e_{mp}^c(f)$, $e_t(f)$, and $e_t^c(f)$ are given in Propositions 5.1 and 5.3. We remark that, in general, one *cannot* ensure that $|e_{mp}(f)| = \frac{1}{2}|e_t(f)|$ and $\left|e_{mp}^c(f)\right| = \frac{1}{2}|e_t^c(f)|$ since $f''(\xi)$ may be evaluated in different values of $\xi \in [a,b]$ depending on the formula under consideration. However, if $f \in \mathbb{P}_2$, one has $f''(x) = C$, a constant value, for all $x \in \mathbb{R}$, for which we have $|e_{mp}(f)| = \frac{1}{2}|e_t(f)|$ and $\left|e_{mp}^c(f)\right| = \frac{1}{2}|e_t^c(f)|$; thus, in this specific case, the mid–point formulas are more accurate than the trapezoidal ones (for the same values of $H$).

## 5.4 Simpson Quadrature Formulas

**Definition 5.5** Let us consider a function $f(x) \in C^0([a,b])$, then the *simple Simpson quadrature formula* is defined as:

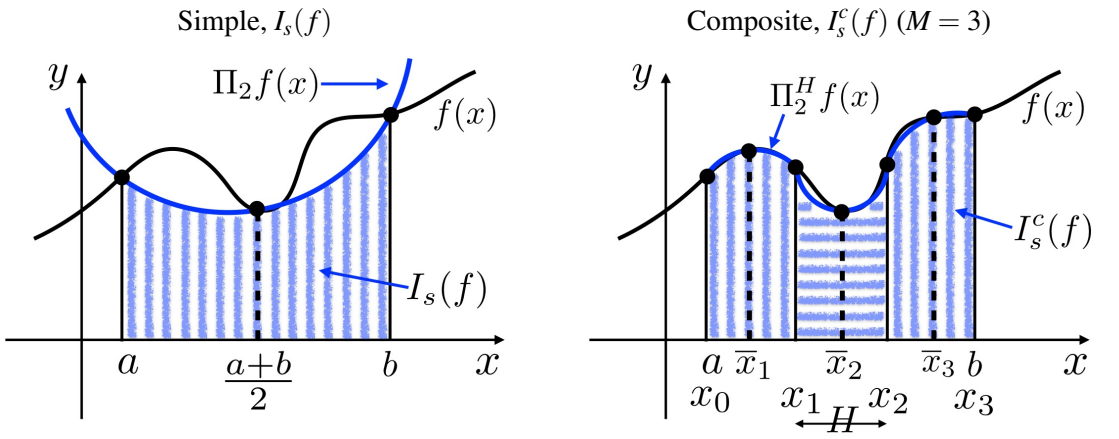
$$I_s(f) := I(\Pi_2 f) = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \tag{5.3}$$

with $\Pi_2 f(x)$ the polynomial of degree 2 interpolating $f(x)$ at the nodes $a$, $b$, and the mid–point $\frac{a+b}{2}$. The *composite Simpson quadrature formula* is defined as:

$$I_s^c(f) := I\left(\Pi_2^H f\right) = \frac{H}{6}\sum_{k=1}^{M}\left[f(x_{k-1}) + 4f(\bar{x}_k) + f(x_k)\right],$$

where $\Pi_2^H f(x)$ is the piecewise polynomial of degree 2 interpolating $f(x)$ at the nodes $\{x_k\}_{k=0}^M$ and mid–points $\{\bar{x}_k\}_{k=1}^M$ of the $M$ subintervals of size $H = \frac{b-a}{M}$ in which $[a,b]$ is subdivided; the subintervals mid–points are defined as $\bar{x}_k = \frac{x_{k-1}+x_k}{2}$ for all $k = 1,\ldots,M$.

♦ **Example 5.4** We illustrate simple (left) and composite (right) Simpson quadrature formulas for the approximation of the integral $I(f)$ of a general function $f(x)$.




♦

**Proposition 5.3** If $f \in C^4([a,b])$, the *error* $e_s(f)$ associated to the *simple Simpson quadrature formula* is:

$$e_s(f) := I(f) - I_s(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi) \qquad \text{for some } \xi \in [a,b],$$

while the *error* $e_s^c(f)$ associated to the *composite Simpson quadrature formula* reads:

$$e_s^c(f) := I(f) - I_s^c(f) = -\frac{(b-a)}{2880} H^4 f^{(4)}(\xi) \qquad \text{for some } \xi \in [a,b].$$

**Remark 5.6** The Simpson quadrature formulas possess *degree of exactness* 3; indeed, the errors $e_s(f)$ and $e_s^c(f)$ are identically zero for all the polynomials of degree less than or equal to 3 (if $f \in \mathbb{P}_3$, $f^{(4)}(\xi) = 0$ for all $\xi \in \mathbb{R}$).

**Remark 5.7** The composite Simpson quadrature formula has *convergence order* (*order of accuracy*) 4; indeed, the error $e_s^c(f)$ depends on $H^4$.

## 5.5 Interpolatory Quadrature Formulas

We consider the case of continuous functions $f(x)$ in the interval $[a,b]$ and specifically *simple* quadrature formulas with the aim of providing a generalization of the previous simple formulas.

**Definition 5.6** Let us consider a function $f(x) \in C^0([a,b])$, then a (simple) *interpolatory quadrature formula* is defined as:

$$\widetilde{I}(f) := I(\widetilde{f}) = \sum_{j=0}^{n} \alpha_j f(y_j), \tag{5.4}$$

where $\widetilde{f}(x)$ is a function interpolating $f(x)$ at the $n+1$ *quadrature nodes* $\{y_j\}_{j=0}^n$ in $[a,b]$ and $\{\alpha_j\}_{j=0}^n$ are the corresponding *quadrature weights*, with $n \geq 0$.

The interpolating function $\widetilde{f}(x)$ should be "easy" to integrate. We remark that there are different choices for such interpolating function $\widetilde{f}(x)$; its choice determines the method or the family of interpolatory quadrature formulas.

**Remark 5.8** If $\widetilde{f}(x) = \Pi_n f(x)$, the interpolating polynomial of degree $n = 0, 1, 2$ at $n+1$ equally spaced nodes in $[a,b]$, one obtains the simple mid–point, trapezoidal, and Simpson quadrature formulas, respectively. Indeed, by setting $\widetilde{f}(x) = \Pi_0 f(x)$ for $n = 0$, $\alpha_0 = b-a$, and $y_0 = \dfrac{a+b}{2}$ in Eq. (5.4), one obtains the simple mid–point quadrature formula (5.1). For $\widetilde{f}(x) = \Pi_1 f(x)$ with $n = 1$, $\alpha_0 = \alpha_1 = \dfrac{b-a}{2}$, $y_0 = a$, and $y_1 = b$ in Eq. (5.4), one gets the simple trapezoidal quadrature formula (5.2). Finally, for $\widetilde{f}(x) = \Pi_2 f(x)$ with $n = 2$, $\alpha_0 = \alpha_2 = \dfrac{b-a}{6}$, $\alpha_1 = \dfrac{2(b-a)}{3}$, $y_0 = a$, $y_1 = \dfrac{a+b}{2}$, and $y_2 = b$ in Eq. (5.4), one obtains the simple Simpson quadrature formula (5.3).

In general, if one chooses $\widetilde{f}(x) = \Pi_n f(x) = \displaystyle\sum_{k=0}^{n} f(x_k)\,\varphi_k(x)$, the Lagrange interpolating polynomial of degree $n \geq 0$ at $n+1$ nodes $\{x_k\}_{k=0}^n$ in $[a,b]$, with $\{\varphi_k(x)\}_{k=0}^n$ the corresponding Lagrange characteristic functions, then the interpolatory formula (5.4) is obtained for the quadrature nodes $y_j = x_j$ and weights $\alpha_j = \displaystyle\int_a^b \varphi_j(x)\,dx$ for all $j = 0, \ldots, n$. In such case, the degree of exactness of the formulas is $r \geq n$.

A minimum objective for interpolatory quadrature formulas (5.4) consists in exactly integrating constant functions $f(x) = C$ for any $n \geq 0$. Since $I(f) = I(C) = C\,(b-a)$, we set $\displaystyle\sum_{j=0}^{n} \alpha_j f\,(y_j) = \sum_{j=0}^{n} \alpha_j C = C\,(b-a)$, for which we obtain the following condition on the quadrature weigths:

$$\sum_{j=0}^{n} \alpha_j = b-a \qquad \text{for all } n \geq 0,$$

regardless of the position of the quadrature nodes.

Interpolatory quadrature formulas (5.4) are specified by $n$, the quadrature nodes $\{y_j\}_{j=0}^n$, and the quadrature weights $\{\alpha_j\}_{j=0}^n$. However, the quadrature weights and nodes depend on the interval $[a,b] \subset \mathbb{R}$ at hand. In order to provide general quadrature formulas that can be applied to functions $f(x)$ in any interval $[a,b]$, the quadrature nodes and weights are specified for the *reference interval* $[-1,1]$ and indicated as $\{\overline{y}_j\}_{j=0}^n$ and $\{\overline{\alpha}_j\}_{j=0}^n$, respectively. Then, quadrature nodes and weights for the general interval $[a,b]$ can be recovered as:

$$y_j = \frac{a+b}{2} + \frac{b-a}{2}\,\overline{y}_j \qquad \text{for } j = 0, \ldots, n,$$

and

$$\alpha_j = \frac{b-a}{2}\,\overline{\alpha}_j \qquad \text{for } j = 0, \ldots, n,$$

respectively.

As anticipated, interpolatory quadrature formulas based on Lagrange characteristic polynomials possess degree of exactness $r \geq n$; specifically, for some of these formulas the degree of exactness $r$ is only equal to $n$ (e.g. for the simple trapezoidal formula for which $n = 1$). The goal consists in finding, for a given $n \geq 0$, the optimal location of the quadrature nodes $\{\overline{y}_j\}_{j=0}^{n}$ in $[-1,1]$ and values of the corresponding quadrature weights $\{\overline{\alpha}_j\}_{j=0}^{n}$ such that the *degree of exactness* of the interpolatory quadrature formula is *maximized*.

### 5.5.1 Gauss–Legendre quadrature formulas

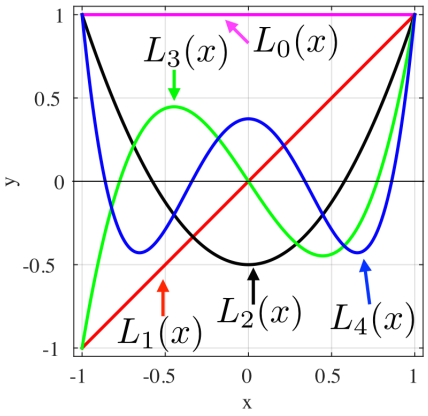
*Gauss–Legendre quadrature formulas* indicate a family of interpolatory quadrature formulas obtained by approximating the function $f(x)$ by means of *Legendre polynomials*. The Legendre polynomials $\{L_k(x)\}_{k=0}^{n+1}$ in the interval $[-1,1]$ are recursively defined as:

$$L_0(x) = 1, \quad L_1(x) = x, \quad \text{and} \quad L_{k+1}(x) = \frac{2k+1}{k+1} x L_k(x) - \frac{k}{k+1} L_{k-1}(x) \qquad \text{for } k = 1, \ldots, n;$$

such polynomials are orthogonal, i.e. $\displaystyle\int_{-1}^{1} L_{n+1}(x)\, L_k(x)\, dx = 0$ for all $k = 0, \ldots, n$.

♦ **Example 5.5** We consider the Legengre polynomials in $[-1,1]$ obtained for $n = 3$.

$$\begin{aligned}
L_0(x) &= 1, \\
L_1(x) &= x, \\
L_2(x) &= \frac{3}{2} x L_1(x) - \frac{1}{2} L_0(x), \\
L_3(x) &= \frac{5}{3} x L_2(x) - \frac{2}{3} L_1(x), \\
L_4(x) &= \frac{7}{4} x L_3(x) - \frac{3}{4} L_2(x).
\end{aligned}$$




♦

**Definition 5.7** Let us consider a function $f(x) \in C^0([a,b])$, then the *Gauss–Legendre quadrature formula* for $n \geq 0$ over the reference interval $[-1,1]$ is:

$$I_{GL,n} = \sum_{j=0}^{n} \overline{\alpha}_j^{GL} f\left(\overline{y}_j^{GL}\right),$$

where:

$$\overline{y}_j^{GL} := \text{zeros of } L_{n+1}(x) \qquad \text{for all } j = 0, \ldots, n,$$

$$\overline{\alpha}_j^{GL} := \frac{2}{\left[1 - \left(\overline{y}_j^{GL}\right)^2\right] \left[L'_{n+1}\left(\overline{y}_j^{GL}\right)\right]^2} \qquad \text{for all } j = 0, \ldots, n.$$

**Remark 5.9** The *degree of exactness* of the Gauss–Legendre quadrature formula is $r = 2n+1$ for all $n \geq 0$.

We report in the following table the quadrature nodes and weights of the Gauss–Legendre quadrature formulas over the interval $[-1,1]$ for $n=0,1$, and 2, as well as the corresponding degrees of exactness $r$. Such formula maximizes $r$ for any given $n \geq 0$.

| $n$ | $\{\bar{y}_j^{GL}\}_{j=0}^n$ | $\{\bar{\alpha}_j^{GL}\}_{j=0}^n$ | $r$ | |
|---|---|---|---|---|
| 0 | $0$ | $2$ | 1 | (mid–point formula) |
| 1 | $\left\{-\frac{1}{\sqrt{3}}, +\frac{1}{\sqrt{3}}\right\}$ | $\{1,1\}$ | 3 | |
| 2 | $\left\{-\frac{\sqrt{15}}{5}, 0, +\frac{\sqrt{15}}{5}\right\}$ | $\left\{\frac{5}{9}, \frac{8}{9}, \frac{5}{9}\right\}$ | 5 | |

We observe that the Gauss–Legendre quadrature formula for $n=0$ coincides with the simple mid–point one.

♦ **Example 5.6** We can verify that the Gauss–Legendre formula for $n=1$ has degree of exactness $r=3$, i.e. $I_{GL,1}(f) = I(f)$ for all $f \in \mathbb{P}_3$. By taking $f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$ for some $c_0$, $c_1$, $c_2$, and $c_3 \in \mathbb{R}$, for example over the reference interval $[-1,1]$, we have $I(f) = \int_{-1}^{1} f(x)\,dx = 2c_0 + \frac{2}{3}c_2$. By setting $n=1$, we have $I_{GL,1}(f) = \left(\bar{\alpha}_0^{GL} + \bar{\alpha}_1^{GL}\right) c_0 + \left(\bar{\alpha}_0^{GL}\bar{y}_0^{GL} + \bar{\alpha}_1^{GL}\bar{y}_1^{GL}\right) c_1 + \left(\bar{\alpha}_0^{GL}\left(\bar{y}_0^{GL}\right)^2 + \bar{\alpha}_1\left(\bar{y}_1^{GL}\right)^2\right) c_2 + \left(\bar{\alpha}_0\left(\bar{y}_0^{GL}\right)^3 + \bar{\alpha}_1\left(\bar{y}_1^{GL}\right)^3\right) c_3$. By enforcing the following constraints (i.e. by setting $I_{GL,1}(f) = I(f)$ for all $c_0$, $c_1$, $c_2$, and $c_3 \in \mathbb{R}$):

$$\begin{cases} \bar{\alpha}_0^{GL} + \bar{\alpha}_1^{GL} = 2, \\ \bar{\alpha}_0^{GL}\bar{y}_0^{GL} + \bar{\alpha}_1^{GL}\bar{y}_1^{GL} = 0, \\ \bar{\alpha}_0^{GL}\left(\bar{y}_0^{GL}\right)^2 + \bar{\alpha}_1\left(\bar{y}_1^{GL}\right)^2 = \frac{2}{3}, \\ \bar{\alpha}_0^{GL}\left(\bar{y}_0^{GL}\right)^3 + \bar{\alpha}_1^{GL}\left(\bar{y}_1^{GL}\right)^3 = 0, \end{cases}$$

we obtain the quadrature nodes $\bar{y}_0^{GL} = -\frac{1}{\sqrt{3}}$ and $\bar{y}_1^{GL} = +\frac{1}{\sqrt{3}}$ and the corresponding weights $\bar{\alpha}_0^{GL} = \bar{\alpha}_1^{GL} = 1$; we deduce that the Gauss–Legendre formula $I_{GL,1}(f)$ exactly integrates the polynomials of degree 3 regardless of their coefficients $c_0$, $c_1$, $c_2$, and $c_3 \in \mathbb{R}$, for which we verify that the formula has degree of exactness 3. ♦

### 5.5.2 Gauss–Legendre–Lobatto quadrature formulas

The Gauss–Legendre quadrature formulas maximize the degree of exactness $r$ for any given $n \geq 0$, but the resulting quadrature nodes are internal the reference interval $[-1,1]$. However, in some instances, one may want to include the boundaries of the interval $\{-1,1\}$ in the set of quadrature nodes. The *Gauss–Legendre–Lobatto quadrature formulas* extend the concept of maximizing the degree of exactness by including the boundaries of the interval as quadrature nodes.

**Definition 5.8** Let us consider a function $f(x) \in C^0([a,b])$, then the *Gauss–Legendre–Lobatto quadrature formula* for $n \geq 1$ over the reference interval $[-1,1]$ is:

$$I_{GLL,n} = \sum_{j=0}^{n} \bar{\alpha}_j^{GLL} f\left(\bar{y}_j^{GLL}\right),$$

where:

$$\bar{y}_0^{GLL} := -1, \quad \bar{y}_n^{GLL} := +1, \quad \text{and} \quad \bar{y}_j^{GLL} := \text{zeros of } L_n'(x) \qquad \text{for all } j = 1,\ldots,n-1,$$

$$\overline{\alpha}_j^{GLL} := \frac{2}{n(n+1)} \frac{1}{\left[L_n\left(\overline{y}_j^{GLL}\right)\right]^2} \qquad \text{for all } j = 0, \ldots, n.$$

**Remark 5.10** The *degree of exactness* of the Gauss–Legendre–Lobatto quadrature formula is $r = 2n-1$ for all $n \geq 1$.

We report in the following table the quadrature nodes and weights of the Gauss–Legendre–Lobatto quadrature formulas over the interval $[-1,1]$ for $n = 1$, 2, and 3, as well as the corresponding degrees of exactness $r$. Notice that the formula is not defined for $n = 0$.

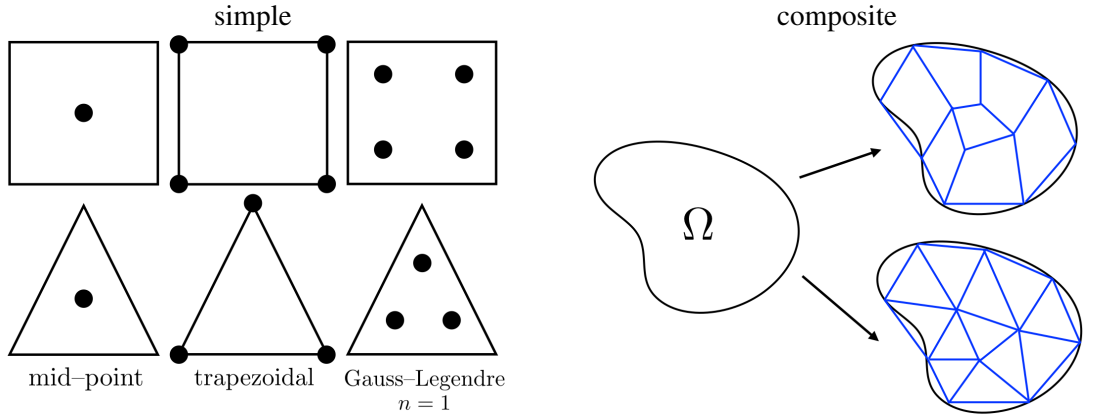
| $n$ | $\{\overline{y}_j^{GLL}\}_{j=0}^{n}$ | $\{\overline{\alpha}_j^{GLL}\}_{j=0}^{n}$ | $r$ | |
|---|---|---|---|---|
| 1 | $\{-1,+1\}$ | $\{1,1\}$ | 1 | (trapezoidal formula) |
| 2 | $\{-1,0,+1\}$ | $\left\{\frac{1}{3},\frac{4}{3},\frac{1}{3}\right\}$ | 3 | (Simpson formula) |
| 3 | $\left\{-1,-\frac{1}{\sqrt{5}},+\frac{1}{\sqrt{5}},+1\right\}$ | $\left\{\frac{1}{6},\frac{5}{6},\frac{5}{6},\frac{1}{6}\right\}$ | 5 | |

We observe that the Gauss–Legendre–Lobatto quadrature formulas for $n = 1$ and 2 coincide with the simple trapezoidal and Simpson ones, respectively.

## 5.6 Numerical Integration in Multiple Dimensions

Numerical integration in multiple dimensions, i.e. the integration of continuous functions $f : \Omega \to \mathbb{R}$, with $\Omega \subset \mathbb{R}^d$ for $d \geq 2$, is based on the generalization of quadrature formulas. Simple quadrature formulas are defined for specific domains, as e.g. trapezoids and triangles for $d = 2$ or tetrahedrons for $d = 3$. Numerical quadrature on complex domains is based on composite formulas.

♦ **Example 5.7** Schematic representations of numerical quadrature formulas for $d = 2$.




♦

# 6. Linear Systems

We consider the numerical solution of *linear systems* both by means of *direct* and *iterative methods*. For simplicity, we consider the case of real valued linear systems, even if several of the considerations and methods presented in this chapter can be straightforwardly used for linear systems involving complex numbers.

## 6.1 Motivations, Examples, and Classification of Methods

We briefly introduce the problem and a classification of methods for the numerical approximation of linear systems.

### 6.1.1 Goals, examples, and notation

Let us consider a square matrix $A \in \mathbb{R}^{n\times n}$ with $n \geq 1$, the vector $\mathbf{b} \in \mathbb{R}^n$, and the solution vector $\mathbf{x} \in \mathbb{R}^n$ of the following linear system:

$$A\mathbf{x} = \mathbf{b}. \tag{6.1}$$

The goal consists in *numerically approximating* the solution $\mathbf{x} \in \mathbb{R}^n$ of such linear system. We recall the following definition and proposition.

**Definition 6.1** The matrix $A \in \mathbb{R}^{n\times n}$ with $n \geq 1$ is *nonsingular* if and only if $\det(A) \neq 0$.

**Proposition 6.1** If $A \in \mathbb{R}^{n\times n}$ is nonsingular, then there exists an unique solution $\mathbf{x} \in \mathbb{R}^n$ of the linear system (6.1).

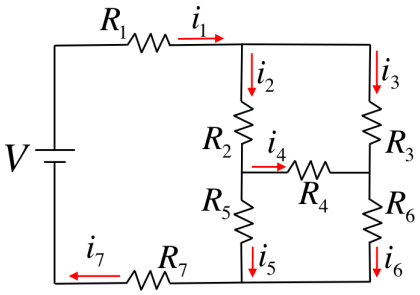
Regarding the linear system (6.1), we use the following notation to indicate the entries of the matrix $A \in \mathbb{R}^{n\times n}$, say $(A)_{ij} = a_{ij}$ for $i, j = 1, \ldots, n$, and the vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^n$, say $(\mathbf{x})_i = x_i$ and $(\mathbf{b})_i = b_i$ for $i = 1, \ldots, n$, respectively. We will also use the following notation to indicate the linear

system in terms of the entries of $A$, $\mathbf{b}$, and $\mathbf{x}$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad \vdots \quad \vdots &= \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases} \quad \text{or} \quad \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

We remark that linear systems may directly arise from physical problems, i.e. are themselves mathematical problems. However, in several instances, linear systems are obtained from the discretization of other mathematical problems. This is the case for differential problems as Partial Differential Equations (PDEs) or systems of Ordinary Differential Equations for which the numerical approximation of the problem generally leads to the solution of linear systems; an example is represented by the finite element method for the spatial approximation of PDEs. In such instances, the larger is the size $n$ of the linear system, the more accurate is the approximation of the mathematical problem which generated the linear system; for such problems, one can easily have $O(n) = 10^5$, $10^6$, or even $10^7$.

♦ **Example 6.1** We show in this example that the mathematical problem of the currents in an electric circuit corresponds to a linear system.




The problem consists in finding the currents $i_j$ for $j = 1,\ldots,n$ through the circuit, for $n = 7$, given the tension $V$ and the resistances $R_j$ of the circuit. The problem is defined by the balance of the tensions ($V = V_1 + V_2 + V_5 + V_7$, $V_3 = V_2 + V_4$, and $V_5 = V_4 + V_6$), the Kirchhoff laws ($i_1 = i_2 + i_3$, $i_2 = i_4 + i_5$, $i_3 + i_4 = i_6$, and $i_5 + i_6 = i_7$), and the constitutive equations ($V_j = R_j i_j$ for all $j = 1,\ldots,n$).

We obtain the following linear system, whose solution provides the distribution of the currents $\{i_j\}_{j=0}^{n}$ in the circuit:

$$\begin{bmatrix} R_1 & R_2 & 0 & 0 & R_5 & 0 & R_7 \\ 0 & R_2 & -R_3 & R_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & R_4 & -R_5 & R_6 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \\ i_7 \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

♦

### 6.1.2 Linear systems and complexity

We need to consider numerical methods and algorithms for the solution of linear systems that can be effectively implemented in software.

♦ **Example 6.2** We show that the solution of the linear system (6.1) by means of the Cramer rule may lead to excessive computational costs already for relatively small matrices $A$. According to the Cramer rule, the solution $\mathbf{x}$ of the linear system (6.1) is computed entry by entry as:

$$x_i = \frac{\det\left(A_i^{\mathbf{b}}\right)}{\det(A)} \quad \text{for all } i = 1,\ldots,n, \quad \text{with } A_i^{\mathbf{b}} := \begin{bmatrix} a_{11} & \cdots & a_{1(i-1)} & b_1 & a_{1(i+1)} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2(i-1)} & b_2 & a_{2(i+1)} & \cdots & a_{2n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & \cdots & a_{n(i-1)} & b_n & a_{n(i+1)} & \cdots & a_{nn} \end{bmatrix}.$$

The solution of the previous linear system with the above rule requires a number of operations $O(3(n+1)!)$. Therefore, by hypothetically assuming to use of calculators with CPU of different performances, we estimate the following computational times for relatively small sizes $n$ of the matrix $A$ ($1GHz = 10^9$ flops).

| $n$ | CPU $1\,GHz$ | CPU $10^3\,GHz$ | CPU $10^6\,GHz$ |
|---|---|---|---|
| 10 | $\sim 10^{-1}\,s$ | $\sim 10^{-4}\,s$ | $\sim 0\,s$ |
| 15 | $\sim 17\,h$ | $\sim 1\,m$ | $\sim 10^{-1}\,s$ |
| 20 | $\sim$ 5000 years | $\sim$ 5 years | $\sim$ 1.7 days |
| 25 | out of reach | out of reach | $\sim$ 40000 years |

Even if more efficient algorithms exist for the Cramer rule (e.g. reducing the number of operations to $O(n^{3.8})$), it is clear how inefficient methods may harm the possibility of effectively solving linear systems. ♦

We observe that by assuming that the matrix $A \in \mathbb{R}^{n\times n}$ is *full*, in principle, one needs at least $n^2$ operations to solve the linear system. Even if this is in general very optimistic, one needs to consider and develop methods for which the number of operations is as closer as possible to this ideal number. Different considerations hold for *sparse* matrices, i.e. matrices for which the number of nonzero entries is $O(n) \ll n^2$.

**Remark 6.1** Solving the linear system (6.1) as $\mathbf{x} = A^{-1}\mathbf{b}$, i.e. by computing the inverse of the matrix $A$, is a computationally inefficient procedure which should be avoided even for very small matrices.

### 6.1.3 Classification of methods for linear systems

The numerical methods for the solution of linear systems can be classified into two categories: *direct* and *iterative* methods.

**Definition 6.2** For a *direct method* the solution $\mathbf{x}$ of the linear system (6.1) is obtained in a *finite* number of steps. Conversely, for an *iterative method* the solution $\mathbf{x}$ is obtained, in principle, in an *infinite* number of steps.

The choice of a direct or iterative method to solve the linear system (6.1) depends on multiple factors as the nature, size, and sparsity of the matrix $A$, as well as the computational resources available (CPU and memory).

## 6.2 Direct Methods

We consider direct methods for the solution of the linear system $A\mathbf{x} = \mathbf{b}$ of Eq. (6.1) and we analyze their properties. The basic idea of this family of methods consists in redirecting the solution of the general linear system $A\mathbf{x} = \mathbf{b}$ to the solution of a "*simpler*" *linear system* by means of suitable manipulations of the matrix $A$.

### 6.2.1 "Simple" linear systems

We report in the following some examples of "simple" linear system for which their solution is straightforward. As anticipated, this depends on the matrix $A \in \mathbb{R}^{n\times n}$ at hand.

#### Diagonal matrix

We consider the case of a *diagonal matrix* $D \in \mathbb{R}^{n\times n}$, i.e. $(D)_{ii} = d_{ii}$ for $i = 1,\dots,n$ and $(D)_{ij} = 0$ for $i, j = 1,\dots,n$, but $j \neq i$. In this case, the diagonal matrix $D$ and the associated linear system

$D\mathbf{x}=\mathbf{b}$ read:

$$D=\begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{bmatrix} \qquad \text{and} \qquad \begin{cases} d_{11}x_1 &= b_1 \\ d_{22}x_2 &= b_2 \\ \vdots &= \vdots \\ d_{nn}x_n &= b_n, \end{cases}$$

respectively; in Eq. (6.1), $A=D$. The solution $\mathbf{x}\in\mathbb{R}^n$ of the diagonal system $D\mathbf{x}=\mathbf{b}$ is:

$$x_i=\frac{b_i}{d_{ii}} \qquad \text{for all } i=1,\dots,n,$$

which is obtained in $n$ operations (divisions).

**Remark 6.2** Since $D$ is a diagonal matrix, we have that its determinant is computed as $\det(D)=\prod_{i=1}^{n} d_{ii}$. It follows that $\det(D)\neq 0$ if and only if $d_{ii}\neq 0$ for all $i=1,\dots,n$.

### Lower triangular matrix: forward substitution algorithm

**Definition 6.3** $L\in\mathbb{R}^{n\times n}$ is a *lower triangular matrix* if and only its entries are such that $(L)_{ij}=l_{ij}\in\mathbb{R}$ for $i=1,\dots,n$, $j=1,\dots,i$ and otherwise $(L)_{ij}=0$ for $i=1,\dots,n$, $j=i+1,\dots,n$; the lower triangular matrix $L$ reads:

$$L=\begin{bmatrix} l_{11} & 0 & \cdots & & & 0 \\ l_{21} & l_{22} & 0 & \cdots & & 0 \\ l_{31} & l_{32} & l_{33} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & & l_{nn} \end{bmatrix}.$$

For $L\in\mathbb{R}^{n\times n}$ a lower triangular matrix, we consider the solution of the *lower triangular system*:

$$L\mathbf{y}=\mathbf{x}, \qquad \text{i.e.} \qquad \begin{cases} l_{11}y_1 &= x_1 \\ l_{21}y_1+l_{22}y_2 &= x_2 \\ l_{31}y_1+l_{32}y_2+l_{33}y_3 &= x_3 \\ \vdots \quad \vdots \quad \vdots \quad \ddots &= \vdots \\ l_{n1}y_1+l_{n2}y_2+l_{n3}y_3+\cdots+l_{nn}y_n &= x_n, \end{cases}$$

where, by referring to Eq. (6.1), we set $A=L$, $\mathbf{y}=\mathbf{x}$, and $\mathbf{b}=\mathbf{x}$. The lower triangular system $L\mathbf{y}=\mathbf{x}$ is solved by means of the *forward substitutions algorithm*, which reads:

$$\begin{aligned} y_1&=\frac{b_1}{l_{11}}, \\ y_i&=\frac{1}{l_{ii}}\left(b_i-\sum_{j=1}^{i-1} l_{ij}y_j\right) \qquad \text{for } i=2,\dots,n. \end{aligned} \tag{6.2}$$

The forward substitutions algorithm solves the lower triangular system $L\mathbf{x}=\mathbf{b}$ in $n^2$ operations, with $n$ the size of the matrix $L$; indeed, $n$ divisions, $\sum_{i=2}^{n}(i-1)$ subtractions, and $\sum_{i=2}^{n}(i-1)$ multiplications are performed by the algorithm yielding the operations count $n+2\sum_{i=2}^{n}(i-1)=n^2$.

**Remark 6.3** Since $L$ is a lower triangular matrix, we have $\det(L) = \prod_{i=1}^{n} l_{ii}$; hence, $\det(L) \neq 0$ if and only if $l_{ii} \neq 0$ for all $i = 1, \ldots, n$.

#### Upper triangular matrix: backward substitution algorithm

**Definition 6.4** $U \in \mathbb{R}^{n\times n}$ is a *upper triangular matrix* if and only its entries are such that $(U)_{ij} = u_{ij} \in \mathbb{R}$ for $i = 1, \ldots, n$, $j = i, \ldots, n$ and otherwise $(U)_{ij} = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, i-1$; the upper triangular matrix $U$ reads:

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & \cdots & 0 & u_{nn} \end{bmatrix}. \tag{6.3}$$

For $U \in \mathbb{R}^{n\times n}$ an upper triangular matrix, we consider the solution of the *upper triangular system*:

$$U\mathbf{x} = \mathbf{y}, \qquad \text{i.e.} \qquad \begin{cases} u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + \cdots + u_{1n}x_n &= y_1 \\ u_{22}x_2 + u_{23}x_3 + \cdots + u_{2n}x_n &= y_2 \\ u_{33}x_3 + \cdots + u_{3n}x_n &= y_3 \\ \ddots \quad \vdots &= \vdots \\ u_{nn}x_n &= y_n, \end{cases}$$

where, by referring to Eq. (6.1), we set $A = U$ and $\mathbf{b} = \mathbf{y}$. The upper triangular system $U\mathbf{x} = \mathbf{y}$ is solved by means of the *backward substitutions algorithm*, which reads:

$$\begin{aligned} x_n &= \frac{y_n}{u_{nn}}, \\ x_i &= \frac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \qquad \text{for } i = n-1, \ldots, 1. \end{aligned} \tag{6.4}$$

The backward substitutions algorithm solves the upper triangular system $U\mathbf{y} = \mathbf{x}$ in $n^2$ operations in analogy with the forward substitutions algorithm for lower triangular systems.

**Remark 6.4** Since $U$ is an upper triangular matrix, $\det(U) = \prod_{i=1}^{n} u_{ii}$, for which $\det(U) \neq 0$ if and only if $u_{ii} \neq 0$ for all $i = 1, \ldots, n$.

### 6.2.2 LU factorization method

**Definition 6.5** Let us consider the nonsingular matrix $A \in \mathbb{R}^{n\times n}$. Then, if it exists, the *LU factorization* (decomposition) of the matrix $A$ consists in determining a lower triangular matrix $L \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $U \in \mathbb{R}^{n\times n}$ such that:

$$A = LU.$$

If the LU factorization of the matrix $A$ exists ($A = LU$), then the linear system $A\mathbf{x} = \mathbf{b}$ can be solved as the sequential solution of the following lower and upper triangular linear systems:

$$L\mathbf{y} = \mathbf{b} \qquad \text{and} \qquad U\mathbf{x} = \mathbf{y};$$

indeed, since $A = LU$, we have $LU\mathbf{x} = \mathbf{b}$ for which, by introducing the auxiliary vector $\mathbf{y} = U\mathbf{x} \in \mathbb{R}^n$, we obtain the previous result.

**Definition 6.6** The *LU factorization method* for the solution of the linear system $A\mathbf{x} = \mathbf{b}$ consists in:

1. determining the *LU factorization* of the matrix $A$ ($A = LU$), if it exists;
2. solving the lower triangular system $L\mathbf{y} = \mathbf{b}$ with the *forward substitutions algorithm* (6.2);
3. solving the upper triangular system $U\mathbf{x} = \mathbf{y}$ with the *backward substitutions algorithm* (6.4).

Since the LU factorization method is based on the LU factorization of $A$, the matrices $L$ and $U$ must be determined, if these exist.

♦ **Example 6.3** We illustrate the LU factorization of a matrix $A \in \mathbb{R}^{n\times n}$ for $n = 2$, which reads:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}}_{U} \quad \text{or} \quad \begin{cases} l_{11}u_{11} &= a_{11} \\ l_{11}u_{12} &= a_{12} \\ l_{21}u_{11} &= a_{21} \\ l_{21}u_{12} + l_{22}u_{22} &= a_{22}. \end{cases}$$

We observe that the matrices $L$ and $U$ involve 6 unknown entries $l_{11}$, $l_{21}$, $l_{22}$, $u_{11}$, $u_{12}$, and $u_{22}$, respectively. However, only 4 constraints can be imposed to determine them. ♦

**Remark 6.5** Following the previous example, for the LU factorization of a general matrix $A \in \mathbb{R}^{n\times n}$ there are $n^2 + n$ unknown entries of the matrices $L$ and $U$, but only $n^2$ constraints to enforce; indeed, $a_{ij} = \sum_{r=1}^{\min\{i,j\}} l_{ir}u_{rj}$ for $i, j = 1,\dots,n$. In order to overcome this issue, by *convention*, the diagonal entries of the lower triangular $L$ obtained by the LU factorization of the matrix $A \in \mathbb{R}^{n\times n}$ are set equal to 1; i.e. $l_{ii} = 1$ for all $i = 1,\dots,n$:

$$L = \begin{bmatrix} 1 & 0 & \cdots & & & 0 \\ l_{21} & 1 & 0 & \cdots & & 0 \\ l_{31} & l_{32} & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & 1 \end{bmatrix}. \tag{6.5}$$

### Gauss elimination method (GEM)

The *Gauss elimination method* (GEM) is used to determine the LU factorization of the matrix $A \in \mathbb{R}^{n\times n}$. In order to illustrate the GEM algorithm, we introduce some notation; specifically, we define the matrix $\overline{A}^{(k)} \in \mathbb{R}^{n\times n}$ for some $k = 1,\dots,n$ as:

$$\overline{A}^{(k)} := \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & & \cdots & a_{2n}^{(2)} \\ 0 & 0 & \ddots & & & \vdots \\ 0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ 0 & \cdots & 0 & a_{k+1,k}^{(k)} & \cdots & a_{k+1,n}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & \cdots & a_{n,n}^{(k)} \end{bmatrix} \quad \text{for } k = 1,\dots,n, \tag{6.6}$$

or equivalently:

$$\left(\overline{A}^{(k)}\right)_{ij} = \begin{cases} a_{ij}^{(i)} & \text{for } i = 1,\dots,k-1,\ j = i,\dots,n \\ a_{ij}^{(k)} & \text{for } i,j = k,\dots,n \\ 0 & \text{otherwise.} \end{cases} \qquad \text{for } k = 1,\dots,n. \tag{6.7}$$

By convention we set $\overline{A}^{(1)} \equiv A$, i.e. $a_{ij}^{(1)} = a_{ij}$ for all $i,j = 1,\dots,n$.

**Definition 6.7** Given the index $k$, with $1 \le k \le n-1$, by referring to the corresponding matrix $\overline{A}^{(k)}$ of Eq. (6.6), its entry $a_{kk}^{(k)}$ is called *pivot element*.

The following GEM algorithm is used to determine the entries of the matrices $L \in \mathbb{R}^{n\times n}$ of Eq. (6.5) and $U \in \mathbb{R}^{n\times n}$ of Eq. (6.3) determining the LU factorization of $A \in \mathbb{R}^{n\times n}$; the $U$ matrix coincides with $\overline{A}^{(n)}$ obtained at the end of the GEM ($U = \overline{A}^{(n)}$).

**Algorithm 6.1:** Gauss elimination method (GEM)

set $\overline{A}^{(1)} = A$;
**for** $k = 1,\dots,n-1$ **do**
  **for** $i = k+1,\dots,n$ **do**
    $l_{ik} = \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$;
    **for** $j = k+1,\dots,n$ **do**
      $a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}\, a_{kj}^{(k)}$;
    **end**
  **end**
  fill $\overline{A}^{(k+1)}$ as in Eq. (6.7) ;
**end**
fill $L$ as in Eq. (6.5) and set $U = \overline{A}^{(n)}$;

The number of operations associated to the GEM for the LU factorization of $A$ is $O\left(\frac{2}{3}n^3\right)$.

**Remark 6.6** In order to perform the LU factorization of the matrix $A$ according to the GEM, all the pivot elements $a_{kk}^{(k)}$ associated to the matrices $\overline{A}^{(k)}$ must be non zero, i.e. $a_{kk}^{(k)} \neq 0$ for all $k = 1,\dots,n-1$.

♦ **Example 6.4** We provide the LU factorization of the matrix $A = \begin{bmatrix} 3 & 1 & -1 \\ 1 & 4 & 2 \\ -1 & -1 & 4 \end{bmatrix}$ using the GEM. We start by setting $\overline{A}^{(1)} = A$ and observing that $n = 3$; then, we obtain the LU factorization following the GEM Algorithm 6.1.

- $k = 1$: $\quad a_{11}^{(1)} = 3,$
  - $i = k+1 = 2$: $\quad l_{21} = \dfrac{a_{21}^{(1)}}{a_{11}^{(1)}} = \dfrac{1}{3},$

* $j = k+1 = 2$: $a_{22}^{(2)} = a_{22}^{(1)} - l_{21}\, a_{12}^{(1)} = \dfrac{11}{3}$,

* $j = n = 3$: $a_{23}^{(2)} = a_{23}^{(1)} - l_{21}\, a_{13}^{(1)} = \dfrac{7}{3}$;

– $i = n = 3$: $l_{31} = \dfrac{a_{31}^{(1)}}{a_{11}^{(1)}} = -\dfrac{1}{3}$,

* $j = k+1 = 2$: $a_{32}^{(2)} = a_{32}^{(1)} - l_{31}\, a_{12}^{(1)} = -\dfrac{2}{3}$,

* $j = n = 3$: $a_{33}^{(2)} = a_{33}^{(1)} - l_{31}\, a_{13}^{(1)} = \dfrac{11}{3}$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ -\frac{1}{3} & ? & 1 \end{bmatrix}, \qquad \overline{A}^{(2)} = \begin{bmatrix} 3 & 1 & -1 \\ 0 & \frac{11}{3} & \frac{7}{3} \\ 0 & -\frac{2}{3} & \frac{11}{3} \end{bmatrix}.$$

- $k = 2$: $a_{22}^{(2)} = \dfrac{11}{3}$,

– $i = k+1 = n = 3$: $l_{32} = \dfrac{a_{32}^{(2)}}{a_{22}^{(2)}} = -\dfrac{2}{11}$,

* $j = k+1 = n = 3$: $a_{33}^{(3)} = a_{33}^{(2)} - l_{32}\, a_{23}^{(2)} = \dfrac{45}{11}$.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ -\frac{1}{3} & -\frac{2}{11} & 1 \end{bmatrix}, \qquad U = \overline{A}^{(3)} = \begin{bmatrix} 3 & 1 & -1 \\ 0 & \frac{11}{3} & \frac{7}{3} \\ 0 & 0 & \frac{45}{11} \end{bmatrix}.$$

♦

**Remark 6.7** If the matrix $A$ admits the LU factorization then $\det(A) = \det(LU) = \det(L)\det(U) = \det(U)$, since $\det(L) = 1$. Therefore, the LU factorization can be used to compute the determinant of the matrix $A$ in a number of operations $O\left(\frac{2}{3}n^3\right)$.

Once the LU factorization of $A$ is computed, the matrices $L$ and $U$ can be stored in an unique matrix, eventually overwriting $A$.

#### Properties of the LU factorization and LU factorization method

The GEM provides the LU factorization of the matrix $A \in \mathbb{R}^{n\times n}$ required to solve the linear system $A\mathbf{x} = \mathbf{b}$ by means of the LU factorization method of Definition 6.6. The number of operations associated to the LU factorization method is $O\left(\frac{2}{3}n^3\right)$; indeed, $O\left(\frac{2}{3}n^3\right)$ operations are required by the GEM, while $n^2$ for both the forward and backward substitutions method.

**Remark 6.8** The LU factorization of the matrix $A \in \mathbb{R}^{n\times n}$ is independent of the vector $\mathbf{b} \in \mathbb{R}^n$ associated to the linear system $A\mathbf{x} = \mathbf{b}$. For this reason, the LU factorization method can be efficiently used for solving the linear system with different vectors $\mathbf{b}$ since the matrices $L$ and $U$ can be computed only once. Then, the computational costs for any new vector $\mathbf{b}$ are associated

only to the solution of the lower $L\mathbf{y} = \mathbf{b}$ and upper triangular $U\mathbf{x} = \mathbf{y}$ systems by means of the forward and backward substitutions algorithms, respectively,

We determine the cases for which the the LU factorization of a nonsingular matrix $A$ exists and is unique. With this, aim, we recall the following definitions and provide some propositions.

**Definition 6.8** The matrix $A \in \mathbb{R}^{n\times n}$ is:

- *symmetric* if and only if $A^T \equiv A$;
- *definite positive* if and only if $\mathbf{z}^T A\mathbf{z} > 0$ for all $\mathbf{z} \in \mathbb{R}^n$ with $\mathbf{z} \neq \mathbf{0}$;
- *diagonally dominant by row* if and only if $|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|$ for all $i = 1, \ldots, n$;
- *strictly diagonally dominant by row* if and only if $|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$ for all $i = 1, \ldots, n$;
- *diagonally dominant by column* if and only if $|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ji}|$ for all $i = 1, \ldots, n$;
- *strictly diagonally dominant by column* if and only if $|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ji}|$ for all $i = 1, \ldots, n$.

The *principal submatrix* of $A \in \mathbb{R}^{n\times n}$ of order $i$, with $1 \leq i \leq n$, is the matrix $A_i \in \mathbb{R}^{i\times i}$ such that $(A_i)_{lm} = (A)_{lm}$ for all $l, m = 1, \ldots, i$.

**Proposition 6.2 — Necessary and sufficient condition for LU factorization.** Given a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, its *LU factorization exists* and is *unique if and only if* $\det(A_i) \neq 0$ for all $i = 1, \ldots, n-1$ (i.e. all the principal submatrices of $A$ of order $i$, with $1 \leq i \leq n-1$, are nonsingular).

♦ **Example 6.5** The LU factorization of the nonsingular matrix $A = \begin{bmatrix} 1 & 1 & 4 \\ 2 & 2 & 3 \\ 4 & 6 & 7 \end{bmatrix}$ using the GEM does not exist. Indeed, using Proposition 6.2, we have $\det(A_1) = \det([1]) \neq 0$, but $\det(A_2) = \det\left(\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}\right) = 0$; specifically, the pivot element $a_{22}^{(2)} = 0$ in the GEM algorithm. ♦

**Proposition 6.3 — Sufficient conditions for LU factorization.** Given the matrix $A \in \mathbb{R}^{n\times n}$, if one of the following conditions holds:

- $A$ is *symmetric* and *definite positive*,
- $A$ is *strictly diagonally dominant by row*,
- or $A$ is *strictly diagonally dominant by column*,

*then* the *LU factorization* of $A$ *exists* and is *unique*.

♦ **Example 6.6** The LU factorization of the matrix $A = \begin{bmatrix} 4 & -2 & 1 \\ -2 & -5 & -1 \\ 1 & 3 & 9 \end{bmatrix}$ exists and is unique according to Proposition 6.3 since $A$ is strictly diagonally dominant by row; indeed, $|4| > |-2| + |1|$, $|-5| > |-2| + |-1|$, and $|9| > |1| + |3|$. ♦

♦ **Example 6.7** Let us consider the nonsingular matrix $A = \begin{bmatrix} 1 & -2 & 8 \\ -2 & 5 & -1 \\ 1 & 1 & 0 \end{bmatrix}$. None of the

sufficient conditions of Proposition 6.3 is satisfied for which one cannot infer the existence and uniqueness of the LU factorization of $A$ by using this result. In this case, the existence and uniqueness of the LU factorization of $A$ must be verified in terms of the necessary and sufficient condition of Proposition 6.2. We deduce that it exists and is unique since $\det(A_1) = \det([1]) \neq 0$ and $\det(A_2) = \det\left(\begin{bmatrix} 1 & -2 \\ -2 & 5 \end{bmatrix}\right) = 9 \neq 0$. ♦

### Pivoting technique

If the necessary and sufficient condition of Proposition 6.2 is not satisfied, the LU factorization of the matrix $A$ cannot be found by using the GEM Algorithm 6.1. Nevertheless, one should still be able to solve the linear system $A\mathbf{x} = \mathbf{b}$ for any nonsingular matrix $A$ by means of the LU factorization method. With this aim, it is possible to adopt the so called *pivoting technique* in combination with the GEM for the LU factorization of $A$.

**Definition 6.9** The *pivoting technique* consists in applying suitable *permutations* of the *rows* (or columns) of the nonsingular matrix $A$ in the presence of null pivot elements $a_{kk}^{(k)}$, for some $k = 1, \ldots, n-1$, encountered during the application of the GEM algorithm.

**Remark 6.9** The application of the *GEM with pivoting technique* ensures the existence and uniqueness of a LU factorization for any nonsingular matrix $A \in \mathbb{R}^{n\times n}$.

We specifically consider the *pivoting technique* with *permutation by rows* of the matrix $A$. Such permutation of the rows of the matrix $A \in \mathbb{R}^{n\times n}$ consists in pre- multiplying it for a permutation matrix $P \in \mathbb{R}^{n\times n}$ as $PA$. The permutation matrix $P$ is an orthogonal matrix, i.e. $P^T = P^{-1}$ ($P^T P = I$); if $P = I$, there are not permutations on the matrix $A$. In general, the permutation matrix $P$ is obtained simultaneously with the matrices $L$ and $U$ by applying the GEM with the pivoting technique to the nonsingular matrix $A$.

♦ **Example 6.8** The nonsingular matrix $A$ of Example 6.5 does not admit the LU factorization with the standard GEM method (without pivoting technique) since the pivot element $a_{22}^{(2)} = 0$. By introducing the permutation by row matrix $P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and by applying it to $A$, we obtain the matrix $\widetilde{A} = PA = \begin{bmatrix} 1 & 1 & 4 \\ 4 & 6 & 7 \\ 2 & 2 & 3 \end{bmatrix}$, for which the second and third rows are permuted. By applying the GEM to $\widetilde{A}$, we obtain the LU factorization with $L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$ and $U = \begin{bmatrix} 1 & 1 & 4 \\ 0 & 2 & -9 \\ 0 & 0 & -5 \end{bmatrix}$, where $\widetilde{A} = PA = LU$; the pivot elements are $\widetilde{a}_{11}^{(1)} = 1 \neq 0$ and $\widetilde{a}_{22}^{(2)} = 2 \neq 0$. ♦

In general, the pivoting technique is applied during the GEM even if the pivot elements are not necessarily zero. Indeed, the pivoting technique can be used also to reduce the propagation of the *round–off errors* involved the application of the GEM at the calculator. Specifically, at the general iterate $k = 1, \ldots, n-1$ of the GEM, the row $k$ is permuted with the $l$ row, where $l = \arg\max_{j=k,\ldots,n} \left|a_{jk}^{(k)}\right|$, with $\widetilde{a}_{kk}^{(k)} = a_{ll}^{(k)}$ the new pivot element.

If the pivoting technique is applied for the determination of the LU factorization of the nonsingular matrix $A$ through the permutation by row matrix $P$, the matrices $L$ and $U$ determine the LU factorization of the permuted matrix $PA$ as:

$$PA = LU.$$

Then, the linear system $A\mathbf{x} = \mathbf{b}$ can be solved as the sequential solution of the following lower and upper triangular linear systems:

$$L\mathbf{y} = P\mathbf{b} \qquad \text{and} \qquad U\mathbf{x} = \mathbf{y};$$

indeed, we have $PA\mathbf{x} = P\mathbf{b}$ and $LU\mathbf{x} = P\mathbf{b}$ for which, by introducing the vector $\mathbf{y} = U\mathbf{x}$, we obtain the previous result.

**Definition 6.10** The *LU factorization method with pivoting technique*, based on the permutation by row matrix $P$, for the solution of the linear system $A\mathbf{x} = \mathbf{b}$ consists in:

1. determining the *LU factorization* of the matrix $PA$ ($PA = LU$);
2. solving the lower triangular system $L\mathbf{y} = P\mathbf{b}$ with the *forward substitutions algorithm* (6.2);
3. solving the upper triangular system $U\mathbf{x} = \mathbf{y}$ with the *backward substitutions algorithm* (6.4).

### 6.2.3 Cholesky factorization method

If the matrix $A$ is symmetric and positive definite, the less computationally expensive *Cholesky factorization* can be used in place of the LU factorization.

**Definition 6.11** Let us consider the *symmetric and positive definite* matrix $A \in \mathbb{R}^{n\times n}$. Then, its *Cholesky factorization* consists in determining an upper triangular matrix $R \in \mathbb{R}^{n\times n}$ such that:

$$A = R^T R.$$

The general form of the upper triangular matrix $R \in \mathbb{R}^{n\times n}$ is:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ 0 & r_{22} & r_{23} & \cdots & r_{2n} \\ \vdots & \ddots & \ddots & \ldots & \vdots \\ 0 & & \cdots & 0 & r_{nn} \end{bmatrix},$$

which, for $A \in \mathbb{R}^{n\times n}$ symmetric and positive definite, is determined by means of the *Cholesky algorithm*.

**Algorithm 6.2:** Cholesky algorithm

$r_{11} = \sqrt{a_{11}}$;
**for** $i = 2, \ldots, n$ **do**
  **for** $j = 1, \ldots, i-1$ **do**
    $r_{ji} = \dfrac{1}{r_{jj}} \left( a_{ij} - \displaystyle\sum_{k=1}^{j-1} r_{ki} r_{kj} \right)$;
  **end**
  $r_{ii} = \sqrt{a_{ii} - \displaystyle\sum_{k=1}^{i-1} r_{ki}^2}$;
**end**

The Cholesky algorithm requires $O\left(\frac{1}{3}n^3\right)$ operations to determine the matrix $R$, a number about the half of that of the LU factorization; in addition, the memory storage is also inferior.

If $A$ is symmetric and positive definite, the Cholesky factorization exists ($A = R^T R$) and the linear system $A\mathbf{x} = \mathbf{b}$ can be solved as the sequential solution of the following lower and upper triangular linear systems:

$$R^T \mathbf{y} = \mathbf{b} \qquad \text{and} \qquad R\mathbf{x} = \mathbf{y},$$

since $R^T$ is a lower triangular matrix; indeed, being $A = R^T R$, we have $R^T R\mathbf{x} = \mathbf{b}$ from which the previous result follows by introducing the vector $\mathbf{y} = R\mathbf{x} \in \mathbb{R}^n$.

**Definition 6.12** The *Cholesky factorization method* for the solution of the linear system $A\mathbf{x} = \mathbf{b}$, with $A$ symmetric and positive definite, consists in:

1. determining the *Cholesky factorization* of the matrix $A$ ($A = R^T R$);
2. solving the lower triangular system $R^T \mathbf{y} = \mathbf{b}$ with the *forward substitutions algorithm* (6.2);
3. solving the upper triangular system $R\mathbf{x} = \mathbf{y}$ with the *backward substitutions algorithm* (6.4).

### 6.2.4 Thomas algorithm

Let us consider a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, with $n \geq 2$, which is *tridiagonal*, i.e. in the form:

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \cdots & & & 0 \\ e_2 & a_2 & c_2 & 0 & \cdots & & 0 \\ 0 & e_3 & a_3 & c_3 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & & \cdots & 0 & e_{n-2} & a_{n-2} & c_{n-2} & 0 \\ 0 & & & \cdots & 0 & e_{n-1} & a_{n-1} & c_{n-1} \\ 0 & & & & \cdots & 0 & e_n & a_n \end{bmatrix},$$

with some real valued entries $\{a_i\}_{i=1}^n$, $\{c_i\}_{i=1}^{n-1}$, and $\{e_i\}_{i=2}^n$. We assume that there exists an unique LU factorization of the matrix $A$ without pivoting; in the case of a tridiagonal matrix $A$, such LU factorization leads to the following lower $L$ and upper $U$ bidiagonal matrices:

$$L = \begin{bmatrix} 1 & 0 & \cdots & & & 0 \\ \beta_2 & 1 & 0 & \cdots & & 0 \\ 0 & \beta_3 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \\ 0 & & \cdots & 0 & \beta_{n-1} & 1 & 0 \\ 0 & & & \cdots & 0 & \beta_n & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} \alpha_1 & c_1 & 0 & \cdots & & & 0 \\ 0 & \alpha_2 & c_2 & 0 & \cdots & & 0 \\ 0 & 0 & \alpha_3 & c_3 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \\ 0 & & & \cdots & 0 & \alpha_{n-1} & c_{n-1} \\ 0 & & & & \cdots & 0 & \alpha_n \end{bmatrix},$$

with some real valued entries $\{\alpha_i\}_{i=1}^n$ and $\{\beta_i\}_{i=2}^n$ determined as:

$$\begin{aligned} &\alpha_1 = a_1, \\ &\beta_i = \frac{e_i}{\alpha_{i-1}} \quad \text{and} \quad \alpha_i = a_i - \beta_i c_{i-1} \qquad \text{for } i = 2,\ldots,n. \end{aligned} \tag{6.8}$$

By considering the linear system $A\mathbf{x} = \mathbf{b}$, with $A \in \mathbb{R}^{n\times n}$ the above mentioned tridiagonal matrix, we solve it by means of the LU factorization method. The LU factorization of $A$ is performed using Eq. (6.8); then, the linear system $L\mathbf{y} = \mathbf{b}$ is solved by means of the following forward substitutions algorithm adapted to the lower bidiagonal matrix $L$:

$$\begin{aligned} &y_1 = b_1, \\ &y_i = b_i - \beta_i y_{i-1} \qquad \text{for } i = 2,\ldots,n. \end{aligned} \tag{6.9}$$

Finally, the linear system $U\mathbf{x} = \mathbf{y}$ is solved by means of the following backward substitutions algorithm adapted to the upper bidiagonal matrix $U$:

$$\begin{aligned} x_n &= \frac{y_n}{\alpha_n}, \\ x_i &= \frac{y_i - c_i x_{i+1}}{\alpha_i} \qquad \text{for } i = n-1, \ldots, 1. \end{aligned} \tag{6.10}$$

**Definition 6.13** The *Thomas algorithm* for the solution of the linear system $A\mathbf{x} = \mathbf{b}$, with $A$ a nonsingular *tridiagonal* matrix which admits an unique LU factorization without pivoting, consists in:

1. determining the *LU factorization* of the matrix $A$ ($A = LU$) using the algorithm of Eq. (6.8);
2. solving the lower bidiagonal system $L\mathbf{y} = \mathbf{b}$ with the algorithm of Eq. (6.9);
3. solving the upper bidiagonal system $U\mathbf{x} = \mathbf{y}$ with the algorithm of Eq. (6.10).

The Thomas algorithm only requires $O(8n)$ operations (precisely $8n-7$) to solve the linear system associated to the tridiagonal matrix $A \in \mathbb{R}^{n\times n}$. We remark that the full LU factorization of $A$ would have required $O\left(\frac{2}{3}n^3\right)$ operations.

### 6.2.5 Accuracy of the numerical solution computed with direct methods

We assess the accuracy of the solution of the linear system $A\mathbf{x} = \mathbf{b}$ by means of direct methods; indeed, when using the calculator, the numerical solution can be affected by the propagation of *round–off errors*.

#### Preliminaries and definitions

We recall and provide some definitions in the context of linear algebra.

**Definition 6.14** For the vector $\mathbf{v} \in \mathbb{R}^n$, its *p–norm* is defined as:

$$\|\mathbf{v}\|_p := \left( \sum_{i=1} |v_i|^p \right)^{1/p} \qquad \text{for } 1 \le p \le +\infty.$$

**Remark 6.10** For the vector $\mathbf{v} \in \mathbb{R}^n$, we have $\|\mathbf{v}\|_2 = \sqrt{\mathbf{v}\cdot\mathbf{v}} = \sqrt{\sum_{i=1} |v_i|^2}$, $\|\mathbf{v}\|_1 = \sum_{i=1}^{n} |v_i|$, and $\|v\|_\infty = \max_{i=1,\ldots,n} |v_i|$. Typically, the norm 2 of the vector $\mathbf{v}$ is simply indicated as $\|\mathbf{v}\| \equiv \|\mathbf{v}\|_2$.

**Definition 6.15** Given the matrix $A \in \mathbb{C}^{n\times n}$, its *eigenvalues* $\{\lambda_i(A)\}_{i=1}^n \in \mathbb{C}$ and the corresponding *eigenvectors* $\{\mathbf{v}_i\}_{i=1}^n \in \mathbb{C}^n$ are such that $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$ for all $i = 1, \ldots, n$. The eigenvalues $\{\lambda_i(A)\}_{i=1}^n$ correspond to the zeros of the *characteristic polynomial* of the matrix $A$, say $p_A(\lambda) := \det(A - \lambda I)$.

**Definition 6.16** The *spectral radius* of the matrix $A \in \mathbb{C}^{n\times n}$, with eigenvalues $\{\lambda_i(A)\}_{i=1}^n \in \mathbb{C}$, is defined as:

$$\rho(A) := \max_{i=1,\ldots,n} |\lambda_i(A)|.$$

**Remark 6.11** For a matrix $A \in \mathbb{C}^{n\times n}$, we observe that:

- $\det(A) = \prod_{i=1}^{n} \lambda_i(A)$;
- $\lambda_i\left(A^{-1}\right) = 1/\lambda_{n+1-i}(A)$ for $i = 1,\dots,n$, if the inverse $A^{-1}$ of $A$ exists;
- $\rho(A) \geq 0$.

We focus now on a real valued matrix $A \in \mathbb{R}^{n\times n}$.

**Proposition 6.4** If the matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric*, then its eigenvalues are real, i.e. $\lambda_i(A) \in \mathbb{R}$ for all $i = 1,\dots,n$. It follows that, if $A \in \mathbb{R}^{n\times n}$ is symmetric, then it is also *positive definite* if and only if all its eigenvalues are strictly positive, i.e. $\lambda_i(A) > 0$ for all $i = 1,\dots,n$.

**Definition 6.17** For the matrix $A \in \mathbb{R}^{n\times n}$, its *p–norm* is defined as:

$$\|A\|_p := \sup_{\substack{\mathbf{v}\in\mathbb{R}^n,\\ \mathbf{v}\neq\mathbf{0}}} \frac{\|A\,\mathbf{v}\|_p}{\|\mathbf{v}\|_p} \qquad \text{for some } 1 \leq p \leq +\infty.$$

**Remark 6.12** For $A \in \mathbb{R}^{n\times n}$, we have:

- $\|A\|_1 = \max_{j=1,\dots,n} \left( \sum_{i=1}^{n} |a_{ij}| \right)$ and $\|A\|_\infty = \max_{i=1,\dots,n} \left( \sum_{j=1}^{n} |a_{ij}| \right)$;
- $\|A\|_2 = \sup_{\mathbf{v}\in\mathbb{R}^n,\ \mathbf{v}\neq\mathbf{0}} \frac{\|A\,\mathbf{v}\|}{\|\mathbf{v}\|} = \sqrt{\lambda_{max}(A^T A)}$, with $\lambda_{max}(A^T A)$ the maximum eigenvalue of $A^T A$;
- if $A$ is symmetric and positive definite, then $\|A\|_2 = \lambda_{max}(A)$ since $\lambda_{max}(A^T A) = (\lambda_{max}(A))^2$.

**Definition 6.18** The *conditioning number* in $p$–norm of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ is:

$$K_p(A) := \|A\|_p \, \|A^{-1}\|_p \qquad \text{for some } 1 \leq p \leq +\infty.$$

By convention, if $A$ is singular, $K_p(A) = +\infty$.

**Definition 6.19** The *spectral conditioning number* of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ is:

$$K(A) := \rho(A)\,\rho\left(A^{-1}\right) = \frac{\max_{i=1,\dots,n} |\lambda_i(A)|}{\min_{i=1,\dots,n} |\lambda_i(A)|},$$

where $\rho(A)$ and $\rho\left(A^{-1}\right)$ are the spectral radii of the matrices $A$ and $A^{-1}$, respectively.

**Remark 6.13** For the nonsingular matrix $A \in \mathbb{R}^{n\times n}$ we have:

- $K_p(A) \geq 1$ for all $1 \leq p \leq +\infty$;
- $K_2(A) = \|A\|_2 \, \|A^{-1}\|_2 = \sqrt{\frac{\lambda_{max}(A^T A)}{\lambda_{min}(A^T A)}}$;
- if the eigenvalues of $A$ are real and strictly positive, $K(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$, with $\lambda_{max}(A)$ and $\lambda_{min}(A)$ the maximum and minimum eigenvalues of $A$, respectively;

- if $A$ is symmetric and positive definite, then $K_2(A) \equiv K(A) = \dfrac{\lambda_{max}(A)}{\lambda_{min}(A)}$.

**Remark 6.14** The conditioning number of a matrix $A$ provides a measure of the sensitivity of the solution of the linear system $A\mathbf{x} = \mathbf{b}$ to perturbations on the data, i.e. $\mathbf{b}$ and $A$ itself. The system is said *well–conditioned* if $K_p(A)$ is relatively "small" and *ill–conditioned* if $K_p(A)$ is "very large" (e.g. $O\left(10^9\right)$ or larger ...).

### Accuracy of the numerical approximation

Solving *numerically* the linear system $A\mathbf{x} = \mathbf{b}$ is equivalent to solve in exact arithmetic the following *perturbed* linear system:

$$(A + \delta A)\,\widehat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}, \tag{6.11}$$

where $\widehat{\mathbf{x}} \in \mathbb{R}^n$ is the numerical solution, $\delta A \in \mathbb{R}^{n\times n}$ the perturbation matrix of $A$, and $\delta\mathbf{b} \in \mathbb{R}^n$ the perturbation vector of $\mathbf{b}$. In order to assess the accuracy of the numerical solution $\widehat{\mathbf{x}}$, we provide the following definitions and error estimates.

**Definition 6.20** For the linear system $A\mathbf{x} = \mathbf{b}$ we define:

- the (absolute) *error* $\mathbf{e} := \mathbf{x} - \widehat{\mathbf{x}}$, with $\mathbf{e} \in \mathbb{R}^n$;
- the *relative error* $e_{rel} := \dfrac{\|\mathbf{x} - \widehat{\mathbf{x}}\|}{\|\mathbf{x}\|}$, for $\mathbf{x} \neq 0$, with $e_{rel} \in \mathbb{R}$;
- the *residual* $\mathbf{r} := \mathbf{b} - A\widehat{\mathbf{x}}$, with $\mathbf{r} \in \mathbb{R}^n$;
- the *relative residual* $r_{rel} := \dfrac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$, for $\mathbf{b} \neq 0$, with $r_{rel} \in \mathbb{R}$.

**Remark 6.15** If $\delta A = 0$ in the perturbed linear system (6.11), then we have $A\widehat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$; it follows that $\delta\mathbf{b} = -\mathbf{r}$.

**Remark 6.16** In general, the *residual* $\mathbf{r}$ and the *relative residual* $r_{rel}$ are used as estimators (or indicators) of the error associated to the numerical solution $\widehat{\mathbf{x}}$; indeed, the exact solution $\mathbf{x}$ of the linear system $A\mathbf{x} = \mathbf{b}$ is generally unknown.

**Proposition 6.5** For the perturbed linear system (6.11), if $K_2(A)\,\dfrac{\|\delta A\|_2}{\|A\|_2} < 1$, then the relative error associated to the numerical solution $\widehat{\mathbf{x}}$ is bounded as:

$$e_{rel} \leq \frac{K_2(A)}{1 - K_2(A)\,\dfrac{\|\delta A\|_2}{\|A\|_2}} \left( \frac{\|\delta A\|_2}{\|A\|_2} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right).$$

It follows that, if $\dfrac{\|\delta A\|_2}{\|A\|_2} = 0$ or nearly zero (e.g. for perturbations $\delta A \simeq 0$), the relative error is bounded as:

$$e_{rel} \leq K_2(A)\, r_{rel} = K_2(A)\, \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}. \tag{6.12}$$

**Corollary 6.6** If $A$ and $\delta A$ are symmetric and positive definite matrices and $\dfrac{\lambda_{max}(\delta A)}{\lambda_{min}(A)} < 1$, then the relative error $e_{rel}$ is bounded as:

$$e_{rel} \leq \frac{K(A)}{1 - \dfrac{\lambda_{max}(\delta A)}{\lambda_{min}(A)}} \left( \frac{\lambda_{max}(\delta A)}{\lambda_{max}(A)} + \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right).$$

If $\dfrac{\lambda_{max}(\delta A)}{\lambda_{min}(A)} = 0$ or nearly zero (e.g. for perturbations $\delta A \simeq 0$), the relative error is bounded as:

$$e_{rel} \leq K(A)\, r_{rel} = K(A)\, \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}. \tag{6.13}$$

**Remark 6.17** The error estimators of Eqs. (6.12) and (6.13) can be fully computed once the numerical solution $\widehat{\mathbf{x}}$ is available.

**Remark 6.18** Based on the results (6.12) and (6.13), the *relative residual* $r_{rel}$ represents a satisfactory *criterion* to assess the *error* associated to the numerical solution $\widehat{\mathbf{x}}$ of the linear system with a direct method only if the conditioning number is "small", i.e. when the matrix $A$ is well–conditioned. Conversely, if the conditioning number of the matrix $A$ is "large", i.e. if $A$ is ill–conditioned, then the error associated to $\widehat{\mathbf{x}}$ may be very "large", even if $r_{rel}$ is "small" due to the propagation of *round–off errors* during the application of the direct method at the calculator.

## 6.3 Iterative Methods

We consider now *iterative methods* for the solution of the linear system $A\mathbf{x} = \mathbf{b}$. The goal consists in solving $A\mathbf{x} = \mathbf{b}$ in principle in an infinite number of steps as $\mathbf{x} = \lim_{k \to +\infty} \mathbf{x}^{(k)}$, where the iterates $\left\{ \mathbf{x}^{(k)} \right\}_{k=0}^{+\infty}$ represent a *sequence* of solution vectors, with $\mathbf{x}^{(0)}$ the *initial guess* (initial solution).

### 6.3.1 The general scheme

A general iterative method for the solution of $A\mathbf{x} = \mathbf{b}$, with $A \in \mathbb{R}^{n \times n}$ nonsingular and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ is:

$$\begin{aligned} &\text{given } \mathbf{x}^{(0)} \in \mathbb{R}^n, \\ &\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{g} \qquad \text{for } k = 0, 1, \ldots, \end{aligned} \tag{6.14}$$

where $B \in \mathbb{R}^{n \times n}$ is the *iteration matrix* and $\mathbf{g} \in \mathbb{R}^n$ is the *iteration vector*; $B$ and $\mathbf{g}$ depend on the matrix $A$, the vector $\mathbf{b}$, and the specific iterative method at hand. However, the iterative method must satisfy the *strong consistency condition* for which, if $\mathbf{x}$ is the solution of $A\mathbf{x} = \mathbf{b}$, one needs $\mathbf{x} = B\mathbf{x} + \mathbf{g}$; therefore, the iteration vector $\mathbf{g} = (I - B)\, A^{-1}\, \mathbf{b}$ since $\mathbf{x} = A^{-1}\, \mathbf{b}$.

**Definition 6.21** We define the *error* $\mathbf{e}^{(k)} \in \mathbb{R}^n$ associated to the iterate $\mathbf{x}^{(k)} \in \mathbb{R}^n$ of iterative method (6.14) as:

$$\mathbf{e}^{(k)} := \mathbf{x} - \mathbf{x}^{(k)} \qquad \text{for } k = 0, 1, \ldots,$$

while the *residual* $\mathbf{r}^{(k)} \in \mathbb{R}^n$ as:

$$\mathbf{r}^{(k)} := \mathbf{b} - A\mathbf{x}^{(k)} \qquad \text{for } k = 0, 1, \ldots.$$

From the definition of the error and the strong consistency condition we have $\mathbf{e}^{(k+1)} = \mathbf{x} - \mathbf{x}^{(k+1)} = (B\mathbf{x} + \mathbf{g}) - \left(B\mathbf{x}^{(k)} + \mathbf{g}\right) = B\left(\mathbf{x} - \mathbf{x}^{(k)}\right) = B\mathbf{e}^{(k)}$ for $k = 0, 1, \ldots$; then, by recursion, it follows that:

$$\mathbf{e}^{(k)} = B^k \mathbf{e}^{(0)} \qquad \text{for } k = 0, 1, \ldots,$$

from which we have the error estimate:

$$\left\|\mathbf{e}^{(k)}\right\| \leq \left\|B^k\right\|_2 \left\|\mathbf{e}^{(0)}\right\| \qquad \text{for } k = 0, 1, \ldots. \tag{6.15}$$

**Proposition 6.7** If the iteration matrix $B \in \mathbb{R}^{n \times n}$ of the iterative method (6.14) is *symmetric and definite positive*, we have:

$$\left\|\mathbf{e}^{(k)}\right\| \leq (\rho(B))^k \left\|\mathbf{e}^{(0)}\right\| \qquad \text{for } k = 0, 1, \ldots,$$

with $\rho(B)$ the spectral radius of $B$.

In general, $\lim_{k \to +\infty} \mathbf{e}^{(k)} = \mathbf{0}$ if and only if $\lim_{k \to +\infty} B^k = 0$, which occurs for $\rho(B) < 1$.

**Proposition 6.8 — Necessary and sufficient condition for convergence.** The iterative method (6.14) is *convergent* to the exact solution $\mathbf{x} \in \mathbb{R}^n$ of the linear system $A\mathbf{x} = \mathbf{b}$ *for all* the *initial guesses* $\mathbf{x}^{(0)} \in \mathbb{R}^n$ if and only if the spectral radius of the iteration matrix $B$ is strictly less than one, i.e. $\rho(B) < 1$. Moreover, the smaller is $\rho(B)$, the faster is the convergence.

### 6.3.2 Splitting methods

Splitting methods represent a family of iterative methods for which the iteration matrix $B$ is deduced following splitting operations on the matrix $A$. With this aim, a nonsingular matrix $P \in \mathbb{R}^{n \times n}$, called *preconditioning matrix* (or preconditioner), is introduced. By observing that $A = P - P + A$ and $A\mathbf{x} = \mathbf{b}$, we have $P\mathbf{x} = (P - A)\mathbf{x} + \mathbf{b}$, from which $\mathbf{x} = P^{-1}(P - A)\mathbf{x} + P^{-1}\mathbf{b}$; from the latter, in virtue of the strong consistency condition, we have the iteration matrix and vector:

$$B = I - P^{-1}A \tag{6.16}$$

and $\mathbf{g} = P^{-1}\mathbf{b}$, respectively. Then, the iterative method (6.14) can be written as $P\mathbf{x}^{(k+1)} = (P - A)\mathbf{x}^{(k)} + \mathbf{b}$, from which $P\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = \mathbf{r}^{(k)}$.

**Definition 6.22** The *preconditioned residual* $\mathbf{z}^{(k)} \in \mathbb{R}^n$ is the solution of the linear system:

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \qquad \text{for } k = 0, 1, \ldots,$$

with $P \in \mathbb{R}^{n \times n}$ the nonsingular preconditioning matrix.

It follows that the iterative method (6.14) can be written as:

$$\begin{aligned} &\text{given } \mathbf{x}^{(0)} \in \mathbb{R}^n, \\ &\text{solve } P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \text{ and set } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)} \qquad \text{for } k = 0, 1, \ldots. \end{aligned} \tag{6.17}$$

We observe that $\mathbf{r}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k)} - A\mathbf{z}^{(k)} = \mathbf{r}^{(k)} - A\mathbf{z}^{(k)}$. Then, from Eq. (6.17) we provide the following preconditioned iterative method.

**Algorithm 6.3:** Preconditioned iterative method

given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, set $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$;
**for** $k = 0, 1, \ldots$, *until a stopping criterion is satisfied* **do**
  solve the linear system $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$;
  set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{z}^{(k)}$;
  set $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - A\mathbf{z}^{(k)}$;
**end**

The iterative method should be stopped by a suitable *stopping criterion*. In particular, we can consider criterion based on the *residual* and the *relative residual* for which the iterations are stopped at the first $k \geq 0$ for which $\left\|\mathbf{r}^{(k)}\right\| < tol$ or $\dfrac{\left\|\mathbf{r}^{(k)}\right\|}{\left\|\mathbf{r}^{(0)}\right\|} < tol$, for some tolerance $tol$; in addition, the number of iterations should be limited by some integer $k_{max}$ "sufficiently" large".

**Remark 6.19** At each step of the iterative method (6.17) one needs to solve a linear system $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$. Therefore, the choice of the preconditioning matrix $P$ should ensure that $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ can be solved in a computationally efficient manner with a direct method (in "few" operations), i.e. it should represent a "simple" linear system. On the other side, the choice of $P$ should lead to a convergent iterative method, i.e. for which the associated iteration matrix $B = I - P^{-1}A$ has spectral radius strictly less than one ($\rho(B) < 1$); in addition, one would like to have $\rho(B) \ll 1$ to ensure fast convergence to the exact solution $\mathbf{x}$.

Let us set $P = I$ for which the linear system is the "simplest" to solve since $\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$; in this case, $B = I - A$ and, since $P$ has not any knowledge of $A$, one has very often $\rho(B) > 1$ or, if the method converges, this typically occurs in several iterations. Conversely, for $P = A$, the linear system $A\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ is as complex to be solved as the original one $A\mathbf{x} = \mathbf{b}$, but $B = 0$ and $\rho(B) = 0$ for which the convergence occurs in one iteration for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$. In this context, the *choice* of the *preconditioning matrix* $P$ is a tradeoff between the "simplicity" of solving the linear system $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ at each iteration and the need of ensure the (rapid) convergence of the iterative method (i.e. $\rho(B) < 1$ and eventually $\rho(B) \ll 1$).

### 6.3.3 Jacobi and Gauss–Seidel methods

We consider the Jacobi and Gauss–Seidel iterative methods for the solution of $A\mathbf{x} = \mathbf{b}$; these methods lay in the category of splitting methods; see Eq. (6.17).

#### Jacobi method

The *Jacobi method* can be used for a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ with *nonzero diagonal entries*, i.e. for $a_{ii} \neq 0$ for all $i = 1, \ldots, n$. The Jacobi method consists in setting the preconditioning matrix $P$ in the general scheme (6.17) as the *diagonal* matrix extracted from $A$. Specifically, by indicating with $P_J$ the preconditioning matrix $P$ for the Jacobi method, we have:

$$P_J = D,$$

where $D \in \mathbb{R}^{n\times n}$ is the diagonal matrix extracted from $A$, i.e. $D$ has nonzero entries $(D)_{ii} = a_{ii}$ for all $i = 1, \ldots, n$. We observe that $\det(P_J) \neq 0$ according to the hypothesis that the diagonal entries of $A$ are nonzero. The linear system $P_J\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ of Eq. (6.17) is "simple" to be solved by a direct

method since $P_J = D$ is a diagonal matrix. The iteration matrix associated to the Jacobi method, say $B_J$, reads:

$$B_J = I - P_J^{-1} A = I - D^{-1} A,$$

from which the convergence of the Jacobi method to $\mathbf{x}$ for all the initial guesses $\mathbf{x}^{(0)}$ depends on its spectral radius $\rho(B_J)$ according to Proposition 6.8.

In matrix form, the $k$th iterate of the Jacobi method can be written as:

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (A - D)\,\mathbf{x}^{(k)} \qquad \text{for } k = 0, 1, \ldots,$$

given some $\mathbf{x}^{(0)} \in \mathbb{R}^n$. This leads to the *Jacobi algorithm*:

$$\text{given } \mathbf{x}^{(0)} \in \mathbb{R}^n,$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1,\, j \neq i}^{n} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n, \quad \text{for } k = 0, 1, \ldots.$$

The Jacobi algorithm is also referred as *simultaneous update*.

#### Gauss–Seidel method

The *Gauss–Seidel method* can be used for a nonsingular matrix $A \in \mathbb{R}^{n \times n}$ with *nonzero diagonal entries*, i.e. for $a_{ii} \neq 0$ for all $i = 1, \ldots, n$. The method considers as preconditioning matrix $P$ in Eq. (6.17) the *lower triangular* matrix extracted from $A$. By convention, we indicate with $D$ the diagonal matrix extracted from $A$ and with $E \in \mathbb{R}^{n \times n}$ a lower triangular matrix (excluding the main diagonal) with nonzero entries $(E)_{ij} = -a_{ij}$ for $i = 2, \ldots, n$ and $j = 1, \ldots, i-1$; finally, $F \in \mathbb{R}^{n \times n}$ is an upper triangular matrix (excluding the main diagonal) with nonzero entries $(F)_{ij} = -a_{ij}$ for $i = 1, \ldots, n-1$ and $j = i+1, \ldots, n$. In this manner, we have $A = D - E - F$. By indicating with $P_{GS}$ the preconditioning matrix $P$ for the Gauss–Seidel method, we have:

$$P_{GS} = D - E,$$

where $\det(P_J) \neq 0$ according to the hypothesis that the diagonal entries of $A$ are nonzero. The linear system $P_{GS}\,\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ of Eq. (6.17) is "simple" to be solved by a direct method since $P_{GS} = D - E$ is a lower triangular matrix. Then, the iteration matrix associated to the Gauss–Seidel method, say $B_{GS}$, reads:

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

the convergence properties of the Gauss–Seidel method depend on its spectral radius $\rho(B_{GS})$ according to Proposition 6.8.

In matrix form, the $k$th iterate of the Gauss–Seidel method reads:

$$(D - E)\mathbf{x}^{(k+1)} = \mathbf{b} - (A - (D - E))\mathbf{x}^{(k)} \quad \text{for } k = 0, 1, \ldots,$$

given some $\mathbf{x}^{(0)} \in \mathbb{R}^n$, from which we deduce the *Gauss–Seidel algorithm*:

$$\text{given } \mathbf{x}^{(0)} \in \mathbb{R}^n,$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n, \quad \text{for } k = 0, 1, \ldots.$$

The Gauss–Seidel algorithm is also referred as *sequential update*.

### Sufficient conditions for the convergence of the Jacobi and Gauss–Seidel methods

The *necessary and sufficient* condition for the convergence of the Jacobi and Gauss–Seidel methods (for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$) is that the spectral radius of the corresponding iteration matrices is strictly less than one; see Proposition 6.8. However, in some instances, it is possible to establish the convergence of the methods simply by inspecting the matrix $A$ of the linear system $A\mathbf{x} = \mathbf{b}$ instead of assembling the iteration matrix $B$ and computing $\rho(B)$. We state the following *sufficient* conditions.

**Proposition 6.9** If $A$ is nonsingular and *strictly diagonally dominant by row*, then the *Jacobi* and *Gauss–Seidel* methods converge to $\mathbf{x}$ for all the initial guesses $\mathbf{x}^{(0)} \in \mathbb{R}^n$.

**Proposition 6.10** If $A$ is *symmetric* and *positive definite*, then the *Gauss–Seidel* method converges to $\mathbf{x}$ for all the initial guesses $\mathbf{x}^{(0)} \in \mathbb{R}^n$.

**Proposition 6.11** If $A$ is nonsingular and *tridiagonal* with all the diagonal entries nonzero, then the *Jacobi* and *Gauss–Seidel* methods are either both divergent or convergent to $\mathbf{x}$. In the latter case, the Gauss–Seidel method converges faster than the Jacobi method since $\rho(B_{GS}) = (\rho(B_J))^2$.

The previous conditions are only *sufficient*; that is, if these are not satisfied, the *necessary and sufficient* condition of Proposition 6.8 must be verified to determine the convergence of the iterative method.

♦ **Example 6.9** We consider the matrix $A = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$, which is nonsingular and strictly diagonally dominant by row. Since the sufficient conditions of Proposition 6.9 are satisfied, then both the Jacobi and Gauss–Seidel methods are convergent for any $\mathbf{x}^{(0)} \in \mathbb{R}^2$ to the solution $\mathbf{x} \in \mathbb{R}^2$ of a linear system $A\mathbf{x} = \mathbf{b}$, for some $\mathbf{b} \in \mathbb{R}^2$. We notice that also the hypotheses of Propositions 6.10 and 6.11 are satisfied. We verify the results by means of the necessary and sufficient condition of Proposition 6.8; i.e. we verify that the spectral radii of the iterations matrices of the Jacobi and Gauss–Seidel methods are strictly less than one. For the Jacobi method, we have $P_J = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$ and $B_J = I - P_J^{-1}A = \begin{bmatrix} 0 & -\frac{1}{3} \\ -\frac{1}{2} & 0 \end{bmatrix}$, from which $\rho(B_J) = \frac{1}{\sqrt{6}} < 1$. For the Gauss–Seidel method, we have $P_{GS} = \begin{bmatrix} 3 & 0 \\ 1 & 2 \end{bmatrix}$ and $B_{GS} = I - P_{GS}^{-1}A = \begin{bmatrix} 0 & -\frac{1}{3} \\ 0 & -\frac{1}{6} \end{bmatrix}$, from which $\rho(B_{GS}) = \frac{1}{6} < 1$. ♦

♦ **Example 6.10** Let us consider the nonsingular matrix $A = \begin{bmatrix} 1 & 0 & -1 \\ 3 & 2 & 0 \\ -1 & -1 & 2 \end{bmatrix}$. In this case, the hypotheses of Propositions 6.9, 6.10, and 6.11 are not satisfied, for which it is necessary to verify the necessary and sufficient condition of Proposition 6.8 to determine the convergence of the Jacobi and Gauss–Seidel methods for all the initial guesses. For the Jacobi method, we have $P_J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ and $B_J = I - P_J^{-1}A = \begin{bmatrix} 0 & 0 & 1 \\ -\frac{3}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$, from which $\rho(B_J) = \frac{109}{100} > 1$; we deduce that the Jacobi method does not converge for all $\mathbf{x}^{(0)} \in \mathbb{R}^3$ to the solution $\mathbf{x} \in \mathbb{R}^3$ of

the linear system $A\mathbf{x} = \mathbf{b}$, for some $\mathbf{b} \in \mathbb{R}^3$. Conversely, for the Gauss–Seidel method, we have

$$P_{GS} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 2 & 0 \\ -1 & -1 & 2 \end{bmatrix} \text{ and } B_{GS} = I - P_{GS}^{-1}A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -\frac{3}{2} \\ 0 & 0 & -\frac{1}{4} \end{bmatrix}, \text{ from which } \rho(B_{GS}) = \frac{1}{4} < 1;$$

hence, the Gauss–Seidel method converges to $\mathbf{x}$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^3$. ♦

### 6.3.4 Preconditioned Richardson methods

Let us introduce a sequence of real valued parameters $\{\alpha_k\}_{k=0}^{+\infty} \in \mathbb{R}$, then the *preconditioned Richardson method* represents a generalization of the iterative method (6.17), reading:

$$\begin{aligned} &\text{given } \mathbf{x}^{(0)} \in \mathbb{R}^n, \\ &\text{solve } P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \text{ and set } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \qquad \text{for } k = 0, 1, \ldots, \end{aligned} \tag{6.18}$$

for a nonsingular preconditioning matrix $P \in \mathbb{R}^{n\times n}$. If $\alpha_k = \alpha \in \mathbb{R}$ for all $k = 0, 1, \ldots$, the iterative method 6.18 is called *stationary* preconditioned Richardson method, while if $\alpha_k$ is not constant with the iteration number $k = 0, 1, \ldots$, it is called *dynamic* preconditioned Richardson method. We observe that for $\alpha_k = \alpha = 1$, we obtain the method of Eq. (6.17). Moreover, $\mathbf{r}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k)} - \alpha_k A\mathbf{z}^{(k)} = \mathbf{r}^{(k)} - \alpha_k A\mathbf{z}^{(k)}$. Then, from Eq. (6.17) we provide the following preconditioned Richardson method.

**Algorithm 6.4:** Dynamic preconditioned Richardson method

given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, set $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$;
**for** $k = 0, 1, \ldots$, *until a stopping criterion is satisfied* **do**
  solve the linear system $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$;
  choose $\alpha_k$;
  set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)}$;
  set $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\mathbf{z}^{(k)}$;
**end**

**Remark 6.20** For a dynamic preconditioned Richardson method, we have $\mathbf{x}^{(k+1)} = B_k\mathbf{x}^{(k)} + \mathbf{g}_k$ for $k = 0, 1, \ldots$, where the dynamic iteration matrix

$$B_k = I - \alpha_k P^{-1}A \qquad \text{for } k = 0, 1, \ldots$$

and the iteration vector $\mathbf{g}_k = \alpha_k P^{-1}A$ vary with the iteration number. Therefore, the convergence properties of the dynamics Richardson method changes with the iteration number $k$, since the parameter $\alpha_k$ also changes with $k$.

**Remark 6.21** For a stationary preconditioned Richardson method, the iteration matrix reads:

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

the convergence properties of the method depend on the spectral radius of $B_\alpha$, i.e. $\rho(\alpha) = \rho(B_\alpha)$.

We consider now conditions for the convergence of the stationary preconditioned Richardson methods to the solution $\mathbf{x} \in \mathbb{R}^n$ of a general linear system $A\mathbf{x} = \mathbf{b}$ for all the initial guesses $\mathbf{x}^{(0)} \in \mathbb{R}^n$.

**Proposition 6.12** If the matrices $A$ and $P \in \mathbb{R}^{n\times n}$ are nonsingular, then the *stationary* preconditioned Richardson method converges to $\mathbf{x} \in \mathbb{R}^n$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$ if and only if

$$\alpha \left|\lambda_i\left(P^{-1}A\right)\right|^2 < 2\mathscr{R}e\left\{\lambda_i\left(P^{-1}A\right)\right\} \qquad \text{for all } i = 1,\ldots,n,$$

with $\alpha \neq 0$, being $\left\{\lambda_i\left(P^{-1}A\right)\right\}_{i=1}^n$ the eigenvalues of $P^{-1}A$.

**Corollary 6.13** If the matrices $A$ and $P \in \mathbb{R}^{n\times n}$ are nonsingular with all the eigenvalues $\left\{\lambda_i\left(P^{-1}A\right)\right\}_{i=1}^n$ *real* (i.e. $\lambda_i\left(P^{-1}A\right) \equiv \mathscr{R}e\left\{\lambda_i\left(P^{-1}A\right)\right\}$ for all $i = 1,\ldots,n$), then the *stationary* preconditioned Richardson method converges to $\mathbf{x} \in \mathbb{R}^n$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$ if and only if

$$0 < \alpha\,\lambda_i\left(P^{-1}A\right) < 2 \qquad \text{for all } i = 1,\ldots,n.$$

**Definition 6.23** The *energy norm* of a vector $\mathbf{v} \in \mathbb{R}^n$ with respect to a symmetric and positive definite matrix $A \in \mathbb{R}^{n\times n}$ is defined as:

$$\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A\,\mathbf{v}}.$$

**Corollary 6.14** If the matrices $A$ and $P \in \mathbb{R}^{n\times n}$ are *symmetric* and *positive definite*, then the *stationary* preconditioned Richardson method converges to $\mathbf{x} \in \mathbb{R}^n$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$ if and only if

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

with $\lambda_{max}\left(P^{-1}A\right)$ the maximum of the eigenvalues of $P^{-1}A$. Moreover, the spectral radius of the iteration matrix $B_\alpha$ is *minimum* for $\alpha = \alpha_{opt}$, where

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

being $\lambda_{min}\left(P^{-1}A\right)$ the minimum of the eigenvalues of $P^{-1}A$; in this case (for $\alpha = \alpha_{opt}$), we have:

$$\left\|\mathbf{e}^{(k)}\right\|_A \leq d^k \left\|\mathbf{e}^{(0)}\right\|_A \qquad \text{for } k = 0,1,\ldots, \tag{6.19}$$

with $d := \dfrac{K\left(P^{-1}A\right)-1}{K\left(P^{-1}A\right)+1}$, being $K\left(P^{-1}A\right) = \dfrac{\lambda_{max}\left(P^{-1}A\right)}{\lambda_{min}\left(P^{-1}A\right)}$ the spectral conditioning number of $P^{-1}A$.

**Remark 6.22** Under the hypotheses of the Corollary 6.14, a choice for the optimal parameter for a stationary Richardson method is provided. However, the result (6.19) also indicates that the "closer" is the preconditioning matrix $P$ to the matrix $A$, the closer is the spectral conditioning number of the matrix $P^{-1}A$ to one and the faster is the convergence of the method; however, in this case, the linear system $P\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$ can be relatively complex to be solved. Specifically, for $P = A$, one has $\alpha_{opt} = 1$ and $d = 0$, for which convergence occurs in one iteration. Conversely, if

$P = I$, one has $\alpha_{opt} = \dfrac{2}{\lambda_{min}(A) + \lambda_{max}(A)}$ and $d = \dfrac{K(A)-1}{K(A)+1}$; in this case, the convergence of the iterative method can be slow if $K(A) \gg 1$, since $d \lesssim 1$. In general, the closer is $K\left(P^{-1}A\right)$ to one, the faster is the convergence of the method.

♦ **Example 6.11** Let us consider $A = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}$ and the preconditioning matrix $P = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$, which are both symmetric and definite positive. Therefore, to study the convergence properties of the *stationary* Richardson method, we can use the results of Corollary 6.14. Such convergence properties depend on the matrix $P^{-1}A = \begin{bmatrix} 1 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix}$ and its eigenvalues $\lambda_{min} = \lambda_{min}\left(P^{-1}A\right) = \dfrac{3}{4} - \dfrac{\sqrt{2}}{4}$ and $\lambda_{max} = \lambda_{max}\left(P^{-1}A\right) = \dfrac{3}{4} + \dfrac{\sqrt{2}}{4}$. In particular, the stationary Richardson method is convergent to the solution $\mathbf{x} \in \mathbb{R}^2$ of a linear system associated to $A$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^2$ if and only if $0 < \alpha < \dfrac{2}{\lambda_{max}} = \dfrac{8}{3+\sqrt{2}}$. Moreover, the optimal parameter $\alpha_{opt} = \dfrac{2}{\lambda_{min} + \lambda_{max}} = \dfrac{4}{3}$ yields the minimum spectral radius among the iteration matrices $B_\alpha$; specifically, $B_{\alpha_{opt}} = I - \alpha_{opt} P^{-1}A = \begin{bmatrix} -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{1}{3} \end{bmatrix}$ and $\rho\left(B_{\alpha_{opt}}\right) = \dfrac{\sqrt{2}}{3} < 1$. From Eq. (6.19), we have $d = \dfrac{K\left(P^{-1}A\right)-1}{K\left(P^{-1}A\right)+1} = \dfrac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \rho\left(B_{\alpha_{opt}}\right) = \dfrac{\sqrt{2}}{3}$; i.e. the error in energy norm $A$ is abated by a factor less than or equal to $\dfrac{\sqrt{2}}{3}$ at each iteration. ♦

In general, for a stationary preconditioned Richardson method, determining the parameter $\alpha$, and eventually $\alpha_{opt}$, can be computationally expensive since it is related to the eigenvalues of $P^{-1}A$. In order to overcome such computations to determine the parameter $\alpha$, dynamic preconditioned Richardson method can be conveniently used.

### 6.3.5 Gradient methods

The *preconditioned gradient* method is a dynamic preconditioned Richardson method (6.18) for which the parameters $\alpha_k$ are specifically chosen as:

$$\alpha_k = \frac{\left(\mathbf{z}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{z}^{(k)}\right)^T A\, \mathbf{z}^{(k)}} \qquad \text{for } k = 0, 1, \ldots,$$

with $P$ a nonsingular preconditioning matrix and $\mathbf{z}^{(k)}$ the preconditioned residual. Similarly, the *gradient* method is a dynamic Richardson method without preconditioning and with the parameters $\alpha_k$ chosen as:

$$\alpha_k = \frac{\left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{r}^{(k)}\right)^T A\, \mathbf{r}^{(k)}} \qquad \text{for } k = 0, 1, \ldots.$$

The gradient method is obtained from the preconditioned gradient method for $P = I$; indeed, in this case, $\mathbf{z}^{(k)} \equiv \mathbf{r}^{(k)}$ for all $k = 0, 1, \ldots$. For the gradient method, the residual vector $\mathbf{r}^{(k)}$ represents the descent direction for the error at the iterate $k = 0, 1, \ldots$ and, if $A$ is a symmetric and positive definite

matrix, the choice of $\alpha_k = \dfrac{\left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{r}^{(k)}\right)^T A\,\mathbf{r}^{(k)}}$ is the one which minimizes the error $\left\|\mathbf{e}^{(k+1)}\right\|_A$ along this direction $\mathbf{r}^{(k)}$. We report the gradient and preconditioned gradient algorithms in the following.

**Algorithm 6.5:** Gradient method

given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, set $\mathbf{r}^{(0)} = \mathbf{b} - A\,\mathbf{x}^{(0)}$;
**for** $k = 0, 1, \ldots$, *until a stopping criterion is satisfied* **do**
  set $\alpha_k = \dfrac{\left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{r}^{(k)}\right)^T A\,\mathbf{r}^{(k)}}$;
  set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\,\mathbf{r}^{(k)}$;
  set $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\,\mathbf{r}^{(k)}$;
**end**

**Algorithm 6.6:** Preconditioned gradient method

given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, set $\mathbf{r}^{(0)} = \mathbf{b} - A\,\mathbf{x}^{(0)}$;
**for** $k = 0, 1, \ldots$, *until a stopping criterion is satisfied* **do**
  solve $P\,\mathbf{z}^{(k)} = \mathbf{r}^{(k)}$;
  set $\alpha_k = \dfrac{\left(\mathbf{z}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{z}^{(k)}\right)^T A\,\mathbf{z}^{(k)}}$;
  set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\,\mathbf{z}^{(k)}$;
  set $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\,\mathbf{z}^{(k)}$;
**end**

**Proposition 6.15** If the matrices $A$ and $P \in \mathbb{R}^{n\times n}$ are *symmetric* and *positive definite*, then the preconditioned *gradient* method converges to the solution $\mathbf{x} \in \mathbb{R}^n$ for all the choices of $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and

$$\left\|\mathbf{e}^{(k)}\right\|_A \leq d^k \left\|\mathbf{e}^{(0)}\right\|_A \qquad \text{for } k = 0, 1, \ldots, \tag{6.20}$$

where $d := \dfrac{K\left(P^{-1}A\right) - 1}{K\left(P^{-1}A\right) + 1}$; $K\left(P^{-1}A\right) = \dfrac{\lambda_{max}\left(P^{-1}A\right)}{\lambda_{min}\left(P^{-1}A\right)}$ is the spectral conditioning number of $P^{-1}A$.

The previous result can be used for the gradient method (without preconditioning), simply by setting $P = I$. Moreover, the error estimate (6.20) can be used to predict a priori the number of iterations necessary for the preconditioned gradient method to converge to the solution up to a prescribed tolerance.

### 6.3.6 Conjugate gradient methods

Let us consider a symmetric and positive definite matrix $A \in \mathbb{R}^{n\times n}$, then the *conjugate gradient* method minimizes, at each iterate $k = 0, 1 \ldots$, the error $\left\|\mathbf{e}^{(k+1)}\right\|_A$ along a descent direction $\mathbf{p}^{(k)} \in \mathbb{R}^n$ which is *A–conjugate* with all the previously computed descent directions $\mathbf{p}^{(j)}$ for $j = 0, \ldots, k-1$

(i.e. $\left(\mathbf{p}^{(j)}\right)^T A\,\mathbf{p}^{(k)} = 0$ for all $j = 0,\ldots,k-1$, if $k \geq 1$). The conjugate gradient method does not fit in the family of dynamic Richardson methods; indeed, at each iteration two parameters $\alpha_k$ and $\beta_k$ need to be determined, the latter to compute the conjugate direction $\mathbf{p}^{(k)}$.

**Algorithm 6.7:** Conjugate gradient method

given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, set $\mathbf{r}^{(0)} = \mathbf{b} - A\,\mathbf{x}^{(0)}$ and $\mathbf{p}^{(0)} = \mathbf{r}^{(0)}$;
**for** $k = 0, 1, \ldots$, *until a stopping criterion is satisfied* **do**
  set $\alpha_k = \dfrac{\left(\mathbf{p}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{p}^{(k)}\right)^T A\,\mathbf{p}^{(k)}}$;
  set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\,\mathbf{p}^{(k)}$;
  set $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\,\mathbf{p}^{(k)}$;
  set $\beta_k = \dfrac{\left(\mathbf{p}^{(k)}\right)^T A\,\mathbf{r}^{(k+1)}}{\left(\mathbf{p}^{(k)}\right)^T A\,\mathbf{p}^{(k)}}$;
  set $\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} - \beta_k \mathbf{p}^{(k)}$;
**end**

**Proposition 6.16** If $A \in \mathbb{R}^{n\times n}$ is *symmetric* and *definite positive*, the *conjugate gradient* method converges to $\mathbf{x} \in \mathbb{R}^n$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$ in *at most n iterations* (in exact arithmetic) and

$$\left\|\mathbf{e}^{(k)}\right\|_A \leq \frac{2\,c^k}{1+c^{2k}} \left\|\mathbf{e}^{(0)}\right\|_A \qquad \text{for } k = 0, 1, \ldots, \tag{6.21}$$

where $c := \dfrac{\sqrt{K(A)}-1}{\sqrt{K(A)}+1}$, with $K(A)$ the spectral conditioning number of $A$.

**Remark 6.23** The conjugate gradient method can be interpreted as a direct method since convergence to $\mathbf{x} \in \mathbb{R}^n$ occurs in at most $n$ iterations in exact arithmetic. However, the algorithm is typically stopped before the $n$ iterations are reached.

**Remark 6.24** For $k$ sufficiently large, the term $\dfrac{2\,c^k}{1+c^{2k}}$ in the error estimate (6.21) behaves as $2\,c^k$. Therefore, for a symmetric and positive definite matrix $A$, the conjugate gradient method converges faster than the gradient method since $2\,c^k < d^k$ when the iteration number $k$ is "sufficiently" large; see Eqs. (6.20) and (6.21).

For $P \in \mathbb{R}^{n\times n}$ a nonsingular preconditioning matrix, one can define *preconditioned conjugate gradient* method by generalizing the conjugate gradient method. Even if we do not provide its algorithm, we highlight the following result.

**Proposition 6.17** If $A$ and $P \in \mathbb{R}^{n\times n}$ are *symmetric* and *definite positive* matrices, the *preconditioned conjugate gradient* method converges to $\mathbf{x} \in \mathbb{R}^n$ for all $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and the error $\left\|\mathbf{e}^{(k)}\right\|_A$ behaves as in Eq. (6.21), but with $c = \dfrac{\sqrt{K\left(P^{-1}A\right)}-1}{\sqrt{K\left(P^{-1}A\right)}+1}$, being $K\left(P^{-1}A\right)$ the spectral conditioning number of $P^{-1}A$.

### 6.3.7 Stopping criterion for iterative methods

As anticipated, iterative methods should be stopped by suitable *stopping criterion*; in addition, the number of iterations of the algorithm should be limited by some integer $k_{max}$ "sufficiently" large. The stopping criterion consists in terminating the algorithm at the iterate $k$ for which a suitable *error estimator* of the true error, say $\widetilde{e}^{(k)}$, is smaller than a prescribed tolerance $tol$, i.e. $\widetilde{e}^{(k)} < tol$. The following error estimators, and associated stopping criterion, can be considered:

- the (absolute) *residual*, for which $\widetilde{e}^{(k)} = \left\|\mathbf{r}^{(k)}\right\|$;
- the *relative residual*, for which $\widetilde{e}^{(k)}_{rel} = r^{(k)}_{rel} := \dfrac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|}$ is used to estimate the relative error $e^{(k)}_{rel} := \dfrac{\left\|\mathbf{x}-\mathbf{x}^{(k)}\right\|}{\|\mathbf{x}\|}$, for $\mathbf{x} \neq \mathbf{0}$;
- the *difference of successive iterates*, for which $\widetilde{e}^{(k)} = \left\|\boldsymbol{\delta}^{(k)}\right\|$, where $\boldsymbol{\delta}^{(k)} := \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ for $k \geq 0$.

Let us start by assessing the quality of the stopping criterion based on the *residual* (absolute and relative). With this aim, we recall the result of Proposition 6.5 and specifically Eq. (6.12), for which, by setting $\widehat{\mathbf{x}} = \mathbf{x}^{(k)}$ and $\mathbf{r} = \mathbf{r}^{(k)}$ for a general (preconditioned) iterative method, we have:

$$e^{(k)}_{rel} \leq K_2(A)\, r^{(k)}_{rel};$$

similarly, for the absolute error:

$$\left\|\mathbf{e}^{(k)}\right\| \leq K_2(A)\, \frac{\|\mathbf{x}\|}{\|\mathbf{b}\|} \left\|\mathbf{r}^{(k)}\right\|.$$

We deduce that the stopping criterion based on the (absolute and relative) residual is *satisfactory* if the conditioning number $K_2(A)$ of the matrix $A$ of the linear system to be solved is relatively "small", i.e. if $A$ is *well–conditioned*. Conversely, if the matrix $A$ is *ill–conditioned*, the stopping criterion based on the residual is *unsatisfactory* since the true error is *underestimated* by the error estimator (the residual).

We consider now the stopping criterion based on the *difference of successive iterates* for the general iterative method (6.17) (or the stationary preconditioned Richardson method). For simplicity, we assume that the iteration matrix $B$ is symmetric and positive definite, for which we can write:

$$\left\|\mathbf{e}^{(k)}\right\| \leq \frac{1}{1-\rho(B)} \left\|\boldsymbol{\delta}^{(k)}\right\| \qquad \text{for } k = 0, 1, \ldots;$$

indeed, using Proposition 6.7, we have $\left\|\mathbf{e}^{(k)}\right\| = \left\|\mathbf{x}-\mathbf{x}^{(k+1)}+\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| = \left\|\mathbf{e}^{(k+1)}+\boldsymbol{\delta}^{(k)}\right\| \leq \left\|\mathbf{e}^{(k+1)}\right\| + \left\|\boldsymbol{\delta}^{(k)}\right\| \leq \rho(B)\left\|\mathbf{e}^{(k)}\right\| + \left\|\boldsymbol{\delta}^{(k)}\right\|$, from which the above result follows. We deduce that the stopping criterion based on the difference of successive iterates is *satisfactory* if the spectral radius of the iteration matrix $B$ is much smaller than one, i.e. $\rho(B) \ll 1$. Conversely, the criterion is *unsatisfactory* if $\rho(B) \lesssim 1$, since the true error is *underestimated* by the error indicator $\left\|\boldsymbol{\delta}^{(k)}\right\|$. The previous considerations hold also for iteration matrices $B$ which are not symmetric nor definite positive.

## 6.4 A Brief Comparison of Direct and Iterative Methods

We present a brief overview of direct and iterative methods for the solution of the linear systems $A\mathbf{x} = \mathbf{b}$, with $A \in \mathbb{R}^{n\times n}$ a nonsingular matrix.

The golden standard to solve $A\mathbf{x} = \mathbf{b}$ with a *direct method* is represented by the *LU factorization* method; the Cholesky factorization method is used for a symmetric and positive definite matrix $A$. The Thomas algorithm is conveniently adopted for a tridiagonal matrix $A$; for matrices in diagonal form (e.g. pentadiagonal) and banded, analogous algorithms can be used.

**Remark 6.25** The MATLAB command \ solves a linear system by means of a direct method, which is suitably and automatically chosen based on on the properties of the matrix $A$; specifically, one obtains the numerical solution of the linear system with the MATLAB command:

```
» x = A \ b;
```

Regarding the *iterative methods*, preconditioned Richardson methods, both stationary and dynamic, constitute a family of methods with a single parameter and a preconditioning matrix $P$. The *preconditioned gradient* and *conjugate gradient* methods represent state of the art iterative methods for the solution of linear systems when the matrix $A$ and the preconditioning matrix $P$ are symmetric and positive definite. We observe that if $A$ is non singular, solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solve $A^T A\mathbf{x} = A^T\mathbf{b}$, for which the above methods can be conveniently used; however, the computational cost and memory storage for generating the matrix $A^T A$ and vector $A^T\mathbf{b}$ should be carefully taken into account for sizes $n$ large. In this respect, for a general matrix $A$, the *GMRES* (Generalized Minimum RESidual) method is very often used for practical problems.

Direct methods can be used to generate preconditioners $P$ for iterative methods. For example, *incomplete LU factorizations* (ILU) can be applied to the matrix $A$ to generate a couple of lower and upper triangular preconditioning matrices $\widetilde{L}$ and $\widetilde{U}$, respectively. Similarly, for $A$ symmetric and positive definite, *incomplete Cholesky factorizations* (IC) can be applied to generate suitable preconditioners.

As a general guideline, the *choice* to use a *direct* or an *iterative method* for the solution of the linear system $A\mathbf{x} = \mathbf{b}$ depends on the properties of the matrix $A$ itself and the computational resources available (CPU and memory). For example, if the size $n$ of the matrix $A$ is very large, iterative methods are preferred if $A$ is a full matrix, while direct methods can be conveniently chosen if $A$ is sparse and banded.

# 7. Approximation of Eigenvalues

We consider the numerical approximation of *eigenvalues* and *eigenvectors* of a matrix $A \in \mathbb{C}^{n\times n}$.

## 7.1 Definitions and Examples

We recall the following definitions since we are considering complex numbers; we indicate with $\imath$ the *imaginary unit* such that $\imath^2 = -1$.

**Definition 7.1** Let us consider the vector $\mathbf{v} \in \mathbb{C}^n$, with components $v_j = a_j + \imath\, b_j$, being $a_j$ and $b_j \in \mathbb{R}$ for all $j = 1,\dots,n$. The vector $\overline{\mathbf{v}}$ indicates the *complex conjugate* of $\mathbf{v}$, which has components $\overline{v}_j = a_j - \imath\, b_j$ for all $j = 1,\dots,n$. The vector $\mathbf{v}^H := (\overline{\mathbf{v}})^T$ is the *transpose complex conjugate* vector of $\mathbf{v}$.

**Definition 7.2** The matrix $A \in \mathbb{C}^{n\times n}$ is *Hermitian* if $A^H \equiv A$ (i.e. $\left(\overline{A}\right)^T \equiv A$).

Then, by recalling Definition 6.15, we provide the following ones.

**Definition 7.3** Given a matrix $A \in \mathbb{C}^{n\times n}$, the *eigenvalue problem* reads: find $\lambda \in \mathbb{C}$ and $\mathbf{x} \in \mathbb{C}^n$ such that $A\mathbf{x} = \lambda\,\mathbf{x}$, where $\lambda$ is an *eigenvalue* and $\mathbf{x}$ an *eigenvector*. The *characteristic polynomial* of the matrix $A$ is $p_A(\lambda) = \det(A - \lambda I)$; the $n$ eigenvalues $\{\lambda_i(A)\}_{i=1}^n$ of $A$ are the zeros of $p_A(\lambda)$.

The following definition provides a generalization of the eigenvalue concept.

**Definition 7.4** Given a matrix $A \in \mathbb{C}^{n\times n}$ and a nonsingular matrix $B \in \mathbb{C}^{n\times n}$, the *generalized eigenvalue problem* reads: find $\lambda \in \mathbb{C}$ and $\mathbf{x} \in \mathbb{C}^n$ such that $A\mathbf{x} = \lambda\, B\mathbf{x}$, where $\lambda$ is a *generalized eigenvalue* and $\mathbf{x}$ the corresponding *eigenvector*. The *characteristic polynomial* of the matrix $A$ with respect to $B$ is $p_{A,B}(\lambda) = \det(A - \lambda\, B)$; the $n$ eigenvalues $\{\lambda_i(A;B)\}_{i=1}^n$ of $A$ with respect to $B$ are the zeros of $p_{A,B}(\lambda)$.

We recall that for a matrix $A \in \mathbb{C}^{n\times n}$ of size $n$ there exist $n$ eigenvalues and $n$ corresponding eigenvectors.

**Definition 7.5** Let us consider a matrix $A \in \mathbb{C}^{n\times n}$ with eigenvalues $\{\lambda_i\}_{i=1}^n \in \mathbb{C}$ and the corresponding eigenvectors $\{\mathbf{x}_i\}_{i=1}^n \in \mathbb{C}^n$; then, the *Rayleigh quotient* is defined as:

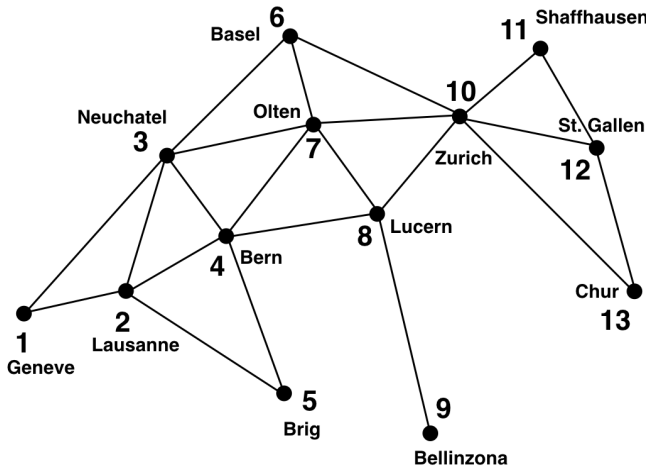
$$\lambda_i = \frac{\mathbf{x}_i^H A\,\mathbf{x}_i}{\mathbf{x}_i^H\,\mathbf{x}_i}, \tag{7.1}$$

for $\mathbf{x}_i \neq \mathbf{0}$.

The definition of Rayleigh quotient (7.1) can be used to determine the eigenvalues of a matrix $A$ once the eigenvectors are known. Conversely, if the eigenvalue $\lambda_i$ is known, the corresponding eigenvector $\mathbf{x}_i$ can be determined by solving $(A-\lambda_i I)\,\mathbf{x}_i = \mathbf{0}$; we notice that typically the eigenvectors are normalized, i.e. $\|\mathbf{x}_i\| = 1$ for $i = 1,\dots,n$.

♦ **Example 7.1** Let us consider the matrix $A = \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}$. The characteristic polynomial is $p_A(\lambda) = (3-\lambda)(1-\lambda) = \lambda^2 - 4\lambda + 3$. The eigenvalues of $A$ are $\lambda_1 = 3$ and $\lambda_2 = 1$, which correspond to the zeros of $p_A(\lambda)$ (i.e. $p_A(\lambda_i) = 0$ for $i = 1$ and 2). Now, we compute the eigenvectors $\mathbf{x}_1$ and $\mathbf{x}_2$. For $\lambda_1 = 3$, we set $(A-\lambda_1 I)\,\mathbf{x}_1 = \mathbf{0}$, yielding $\begin{bmatrix} 0 & -1 \\ 0 & -2 \end{bmatrix}\mathbf{x}_1 = \mathbf{0}$ and $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. For $\lambda_2 = 1$, we set $(A-\lambda_2 I)\,\mathbf{x}_2 = \mathbf{0}$, i.e. $\begin{bmatrix} 2 & -1 \\ 0 & 0 \end{bmatrix}\mathbf{x}_2 = \mathbf{0}$, thus obtaining $\mathbf{x}_2 = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 \\ 2 \end{bmatrix}$. ♦

♦ **Example 7.2** We assess the connectivity of some cities in Switzerland through the rail network.




With this aim, we assemble a matrix $A \in \mathbb{R}^{n\times n}$, with $n$ the number of cities connected by the rail network, which represent a connectivity matrix. The entries of $A$ are zero, except those entries $(A)_{ij} = 1$ for some $i, j = 1,\dots,n$ such that the cities $i$ and $j$ are directly connected; $(A)_{ii} = 0$ for all $i = 1,\dots,n$. We consider a matrix $A$ associated to a scheme of the Swiss rail network.

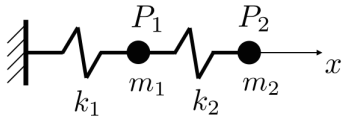
The largest eigenvalue $\lambda_1$ of the matrix $A$ is indicative of the overall connectivity of the network. The components of the associated eigenvector $\mathbf{x}_1$ quantify the relative connectivity of the corresponding cities to the rest of the network. For the rail network reported above, we have:

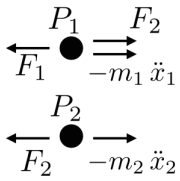
$$\mathbf{x}_1 \simeq (0.16, 0.28, 0.39, 0.39, 0.17, 0.29, 0.43, 0.31, 0.078, 0.36, 0.13, 0.15, 0.13)^T,$$

from which we deduce that Bellinzona ($\mathbf{x}_{1,9}$) is the least connected city in the network, while Olten ($\mathbf{x}_{1,7}$) the most connected one. ♦

♦ **Example 7.3** We consider the dynamics of two concentrated masses connected by elastic springs. Specifically, let us consider two particles $P_1$ and $P_2$ with masses $m_1$ and $m_2$, respectively, and connected through elastic springs with elastic coefficients $k_1$ and $k_2$; the particle $P_1$ is anchored

as highlighted in the following figure. The displacement of the particles are $x_1(t)$ and $x_2(t)$, with $t$ the time variable.




The internal forces read $F_1 = k_1 x_1$ and $F_2 = k_1(x_2 - x_1)$, while the inertial forces are $-m_1\ddot{x}_1$ and $-m_2\ddot{x}_2$ for the two particles, respectively.

The equilibrium of the forces, i.e. $-m_1\ddot{x}_1 = F_1 - F_2$ and $-m_2\ddot{x}_2 = F_2$, leads to the following linear system at each time $t$:

$$B\ddot{\mathbf{x}}(t) + A\,\mathbf{x}(t) = \mathbf{0},$$

where $\mathbf{x}(t) = (x_1(t), x_2(t))^T$, $B = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}$, and $A = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix}$. By assuming that $x_i(t) = a_i \sin(\omega t + \phi)$, for some $\omega$, $\phi$, and $a_i$ for $i = 1, 2$, the above system can be rewritten as:

$$\left(-\omega^2 B + A\right)\mathbf{a} = \mathbf{0},$$

where $\mathbf{a} = (a_1, a_2)^T$. The latter represents a generalized eigenvalue problem for which $\{\omega_i\}_{i=1}^2$ are the *natural frequencies* and $\{\mathbf{a}_i\}_{i=1}^2$ the corresponding *eigenmodes*. ♦

## 7.2 Power Method

Let us assume that for a matrix $A \in \mathbb{C}^{n\times n}$ its eigenvalues $\{\lambda_i\}_{i=1}^n$ are ordered as:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|,$$

with the two *largest* eigenvalues *distinct*, i.e. $\lambda_1 \neq \lambda_2$. Moreover, we assume that the eigenvectors of $A$ are linearly independent, i.e. $\det([\mathbf{x}_1, \ldots, \mathbf{x}_n]) \neq 0$. Under these hypotheses, the *power method* approximates the *largest eigenvalue* $\lambda_1$ of the matrix $A$ by means of the following algorithm.

**Algorithm 7.1:** Power method

set $\mathbf{x}^{(0)} \in \mathbb{C}^n$, with $\left\|\mathbf{x}^{(0)}\right\| \neq 0$;

$\mathbf{y}^{(0)} = \dfrac{\mathbf{x}^{(0)}}{\left\|\mathbf{x}^{(0)}\right\|}$;

**for** $k = 1, 2, \ldots$, *until a stopping criterion is satisfied* **do**

$\mathbf{x}^{(k)} = A\,\mathbf{y}^{(k-1)}$;

$\mathbf{y}^{(k)} = \dfrac{\mathbf{x}^{(k)}}{\left\|\mathbf{x}^{(k)}\right\|}$;

$\lambda^{(k)} = \left(\mathbf{y}^{(k)}\right)^H A\,\mathbf{y}^{(k)}$;

**end**

In the algorithm of the power method, $\lambda^{(k)}$ represents an approximation of the largest eigenvalue $\lambda_1$ of $A$ through the Rayleigh quotient (7.1), while $\mathbf{y}^{(k)}$ is an approximation of the corresponding eigenvector $\mathbf{x}_1$. As *stopping criterion* for the power method reported in Algorithm 7.1, the following one is used:

$$\frac{\left|\lambda^{(k)} - \lambda^{(k-1)}\right|}{\left|\lambda^{(k)}\right|} < tol \qquad \text{for } k = 1, 2, \ldots, \tag{7.2}$$

where *tol* is a prescribed tolerance.

At the general iterate $k \geq 2$ of the Algorithm 7.1, we have that $\mathbf{y}^{(k)} = \dfrac{\mathbf{x}^{(k)}}{\left\|\mathbf{x}^{(k)}\right\|} = \dfrac{1}{\left\|\mathbf{x}^{(k)}\right\|} A\mathbf{y}^{(k-1)} = \dfrac{1}{\left\|\mathbf{x}^{(k)}\right\| \left\|\mathbf{x}^{(k-1)}\right\|} A^2\mathbf{y}^{(k-2)}$; it follows that:

$$\mathbf{y}^{(k)} = \frac{1}{\prod_{j=1}^{k} \left\|\mathbf{x}^{(j)}\right\|} A^k \mathbf{y}^{(0)} \qquad \text{for } k = 1, 2, \ldots,$$

from which the denomination of the power method. For the set of linearly independent eigenvectors $\{\mathbf{x}_i\}_{i=1}^n \in \mathbb{C}^n$ of $A$, we can write any vector $\mathbf{v} \in \mathbb{C}^n$ as $\mathbf{v} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$, for some coefficients $\{\alpha_i\}_{i=1}^n \in \mathbb{C}$. Therefore, we have $\mathbf{x}^{(0)} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$ and $\mathbf{y}^{(0)} = \dfrac{1}{\left\|\mathbf{x}^{(0)}\right\|} \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$. Hence, $\mathbf{x}^{(1)} = A\mathbf{y}^{(0)} = \dfrac{1}{\left\|\mathbf{x}^{(0)}\right\|} \sum_{i=1}^{n} \alpha_i A\mathbf{x}_i = \dfrac{1}{\left\|\mathbf{x}^{(0)}\right\|} \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{x}_i$ and $\mathbf{y}^{(1)} = \dfrac{1}{\left\|\mathbf{x}^{(1)}\right\|} \mathbf{x}^{(1)} = \beta^{(1)} \sum_{i=1}^{n} \alpha_i \lambda_i \mathbf{x}_i$, with $\beta^{(1)} := \dfrac{1}{\left\|\mathbf{x}^{(1)}\right\| \left\|\mathbf{x}^{(0)}\right\|}$. By continuing in the same manner, for $k = 2, 3, \ldots$, we obtain:

$$\mathbf{y}^{(k)} = \beta^{(k)} \sum_{i=1}^{n} \alpha_i \lambda_i^k \mathbf{x}_i = \beta^{(k)} \lambda_1^k \left[ \alpha_1 \mathbf{x}_1 + \sum_{i=2}^{n} \alpha_i \left( \frac{\lambda_i}{\lambda_1} \right)^k \mathbf{x}_i \right] \qquad \text{for } k = 0, 1, \ldots$$

where $\beta^{(k)} := \dfrac{1}{\prod_{i=0}^{k} \left\|\mathbf{x}^{(i)}\right\|}$. From the previous result, we observe that, if $\alpha_1 \neq 0$, then as $k$ tends to $+\infty$ the iterates $\mathbf{y}^{(k)}$ tend to align along the same direction as the eigenvector $\mathbf{x}_1$, because $\lim_{k \to +\infty} \left( \dfrac{\lambda_i}{\lambda_1} \right)^k = 0$, being $|\lambda_1| > |\lambda_i|$ for all $i = 2, \ldots, n$, and $\lim_{k \to +\infty} \left| \beta^{(k)} \lambda_1^k \alpha_1 \right| = 1$.

For a general matrix $A \in \mathbb{C}^{n \times n}$, the absolute *error* on the eigenvalue $\lambda_1$ for an approximation $\lambda^{(k)}$ obtained by the power method (when applicable) reads:

$$e^{(k)} = \left| \lambda_1 - \lambda^{(k)} \right| \simeq \left| \frac{\lambda_2}{\lambda_1} \right|^k \qquad \text{for } k \text{ “sufficiently” large.}$$

If the matrix $A \in \mathbb{C}^{n \times n}$ is Hermitian, i.e., if $A^H \equiv A$, then we have:

$$e^{(k)} = \left| \lambda_1 - \lambda^{(k)} \right| \simeq \left| \frac{\lambda_2}{\lambda_1} \right|^{2k} \qquad \text{for } k \text{ “sufficiently” large.}$$

The power method allows the approximation of the largest eigenvalue $\lambda_1$ of $A$, under the hypotheses for which the method is applicable. If $A$ is nonsingular, then $\lambda_i(A^{-1}) = 1/\lambda_{n+1-i}(A)$ for $i = 1, \ldots, n$. Therefore, the power method can be used to approximate the largest eigenvalue of $A^{-1}$; then, the smallest eigenvalue of $A$ is obtained as $\lambda_n(A) = \dfrac{1}{\lambda_1(A^{-1})}$. However, this approach is in general computationally expensive if $A$ is “large” since its inverse $A^{-1}$ needs to be first computed.

## 7.3 Inverse Power Method

Let us assume that for a *nonsingular* matrix $A \in \mathbb{C}^{n \times n}$ its eigenvalues $\{\lambda\}_{i=1}^n$ are ordered as

$$|\lambda_1| \geq \cdots \geq |\lambda_{n-1}| \geq |\lambda_n|,$$

with the two *smallest* eigenvalues *distinct*, i.e. $\lambda_n \neq \lambda_{n-1}$. In addition, we assume that the eigenvectors of $A$ are linearly independent, i.e. $\det([\mathbf{x}_1, \ldots, \mathbf{x}_n]) \neq 0$. Under these hypotheses, the *inverse power method* approximates the *smallest eigenvalue* $\lambda_n$ of $A$ by means of the following algorithm.

**Algorithm 7.2:** Inverse power method

set $\mathbf{x}^{(0)} \in \mathbb{C}^n$, with $\|\mathbf{x}^{(0)}\| \neq 0$;

$\mathbf{y}^{(0)} = \dfrac{\mathbf{x}^{(0)}}{\|\mathbf{x}^{(0)}\|}$;

**for** $k = 1, 2, \ldots,$ *until a stopping criterion is satisfied* **do**

  solve $A\mathbf{x}^{(k)} = \mathbf{y}^{(k-1)}$;

  $\mathbf{y}^{(k)} = \dfrac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}$;

  $\mu^{(k)} = (\mathbf{y}^{(k)})^H A^{-1} \mathbf{y}^{(k)}$;

**end**

$\lambda = \dfrac{1}{\mu^{(k)}}$;

In the algorithm of the inverse power method, $\mu^{(k)}$ represents an approximation of $\dfrac{1}{\lambda_n}$, with $\lambda_n$ the smallest eigenvalue of $A$, i.e. $\lim_{k \to +\infty} \mu^{(k)} = \dfrac{1}{\lambda_n}$. As stopping criterion, the one already presented for the power method can be considered by replacing $\lambda^{(k)}$ and $\lambda^{(k-1)}$ with $\mu^{(k)}$ and $\mu^{(k-1)}$ in Eq. (7.2), respectively.

**Remark 7.1** Since the inverse power method involves the solution of the linear system $A\mathbf{x}^{(k)} = \mathbf{y}^{(k-1)}$ at each iteration of the Algorithm 7.2, it is convenient to use the LU factorization method; indeed, the LU factorization of the matrix A can be performed only once (at the first iteration).

## 7.4 Inverse Power Method with Shift

**Definition 7.6** Let us consider a matrix $A \in \mathbb{C}^{n \times n}$, then the *shift* is a complex number $s \in \mathbb{C}$ such that a matrix $A_s \in \mathbb{C}^{n \times n}$ is obtained as $A_s = A - sI$.

**Remark 7.2** Let $\{\lambda_i(A)\}_{i=1}^n$ be the eigenvalues of the matrix $A \in \mathbb{C}^{n \times n}$, then the eigenvalues of the matrix $A_s \in \mathbb{C}^{n \times n}$, with $s \in \mathbb{C}$ the shift, are $\lambda_j(A_s) = \lambda_i(A) - s$, for some $i, j = 1, \ldots, n$.

The shift is useful to approximate the eigenvalue of $A$ nearest to a given number $s \in \mathbb{C}$, that we denote by $\lambda_s(A)$. Indeed, we can compute the minimum eigenvalue of the matrix $A_s = A - sI$ with the inverse power method, and then we obtain $\lambda_s(A) = \lambda_n(A_s) + s$. In general, by suitably selecting the shift values $s_i$ such that $\lambda_n(A_{s_i}) = \lambda_i(A) - s_i$, one can use the inverse power method to approximate all the eigenvalues of $A$, i.e. $\lambda_i(A)$ for $i = 2, \ldots, n$. We remark that the so callled Gershgorin circles approximately locate the eigenvalues $\{\lambda_i\}_{i=1}^n$ of the matrix $A$ in the space $\mathbb{C}$, and can be used to suitably choose the corresponding shift values for each eigenvalue.

♦ **Example 7.4** Let us consider a matrix $A \in \mathbb{R}^{3 \times 3}$ such that its eigenvalues are $\lambda_1(A) = 5.3$, $\lambda_2(A) = 2.1$ and $\lambda_3(A) = 0.2$. The eigenvalues $\lambda_1(A)$ and $\lambda_3(A)$ are approximated with the power method and the inverse power method, respectively. Let us set, for example, the shift value to $s = 2.5$, for which the matrix $A_s = A - sI$ possesses eigenvalues $\lambda_1(A_s) = 2.8$, $\lambda_2(A_s) = -2.4$, and $\lambda_3(A_s) = -0.4$. By using the inverse power method for the matrix $A_s$, the eigenvalue $\lambda_3(A_s) = -0.4$ is approximated. Finally, the middle eigenvalue of $A$ is obtained as $\lambda_2(A) = \lambda_3(A_s) + s = 2.1$. ♦

# 8. Ordinary Differential Equations

We consider the numerical approximation of *Ordinary Differential Equations* (ODEs) and systems of ODEs.

## 8.1 Introduction and Examples

We aim at numerically approximating differential equations in the form:

$$F\left(y(t), \frac{dy}{dt}(t), \frac{d^2y}{dt^2}(t), \ldots, \frac{d^py}{dt^p}(t)\right) = 0,$$

with $t$ an independent variable, often associated with the time variable, $y(t)$ the solution of the differential problem, and $p$ the *order* of the differential equation. We mainly focus on first order problems for which $p = 1$, i.e.:

$$F\left(y(t), \frac{dy}{dt}(t)\right) = 0;$$

we remark that ODEs of order $p > 1$ can be recast in systems of ODEs of order $p = 1$.

### 8.1.1 The Cauchy problem

**Definition 8.1** Let us consider the interval $I = (t_0, t_f) \subset \mathbb{R}$, then the *Cauchy problem* reads:

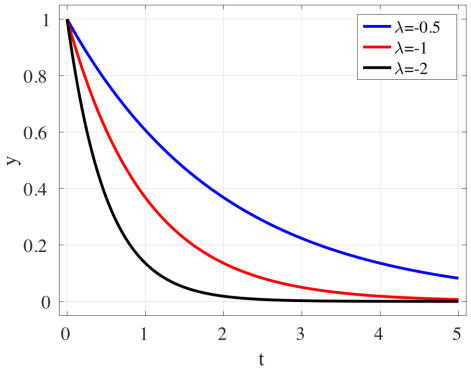
$$\text{find } y : I \subset \mathbb{R} \to \mathbb{R} \quad : \quad \begin{cases} \dfrac{dy}{dt}(t) = f(t, y(t)) & \text{for all } t \in I, \\ y(t_0) = y_0, \end{cases} \tag{8.1}$$

where $f(t,y) : I \times \mathbb{R} \to \mathbb{R}$ is a given function with two arguments and $y_0$ is the initial datum.

♦ **Example 8.1 Model problem**. The model problem is a Cauchy problem (8.1) for which $f(t,y) = \lambda\, y$ for some $\lambda \in \mathbb{R}$ and $\lambda < 0$. Such problem admits the solution:

$$y(t) = y_0\, e^{\lambda\,(t-t_0)} \qquad \text{for all } t \in [t_0, t_f]; \tag{8.2}$$

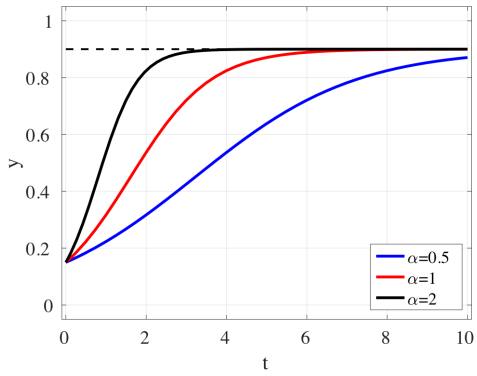
often the model problem is defined in the interval $I = (t_0, +\infty)$.




Solution of the model problem for $y_0 = 1$, $t_0 = 0$, $t_f = 5$, and different values of $\lambda$; $\lambda = -0.5$ (blue), $-1.0$ (red), and $-2.0$ (black).

♦

♦ **Example 8.2** Biology models often involve ODEs. A simple model to describe the evolution of the concentration of a bacteria in a solution, say $y(t)$, corresponds to the Cauchy problem (8.1) with $f(t,y) = \alpha y \left(1 - \frac{1}{\beta}\right)$, where $\alpha$ and $\beta > 0$, the latter representing the maximum concentration of the bacteria.




Solution of the problem for $y_0 = 0.15$, $\beta = 0.9$, $t_0 = 0$, $t_f = 10$, and different values of $\alpha$; $\alpha = 0.5$ (blue), 1.0 (red), and 2.0 (black).

♦

### 8.1.2 Well–posedness of the Cauchy problem

**Proposition 8.1** If the function $f(t,y) : I \times \mathbb{R} \to \mathbb{R}$ is:

1. *continuous* in both the arguments and
2. *Lipschitz continuous* with respect to the *second* argument $y$ for all $t \in I$, i.e. there exists a constant $L > 0$ such that $|f(t,y_1) - f(t,y_2)| \leq L\,|y_1 - y_2|$ for all $t \in I$ and for all $y_1, y_2 \in \mathbb{R}$,

then, there *exists* an *unique solution* $y(t) : I \to \mathbb{R}$ of the Cauchy problem (8.1) and $y \in C^1(I)$.

**Remark 8.1** If the function $f(t,y) : I \times \mathbb{R} \to \mathbb{R}$ is $C^1$–continuous in the second argument $y$, then it is also Lipschitz continuous in the second argument. Indeed, we have $|f(t,y_1) - f(t,y_2)| \leq \left(\max_{y\in\mathbb{R}, t\in I} \left|\frac{\partial f}{\partial y}(t,y)\right|\right) |y_1 - y_2|$, for which one can take $L = \max_{y\in\mathbb{R}, t\in I} \left|\frac{\partial f}{\partial y}(t,y)\right|$.

We observe that Proposition 8.1 allows to determine the existence and uniqueness of the solution $y(t)$, but not to determine it in closed form.

## 8.2 Numerical Approximation of Ordinary Differential Equations

We consider the partition of $\bar{I} = [t_0, t_f]$ into $N_h$ subintervals of equal size $h = \dfrac{t_f - t_0}{N_h}$ for which $t_n = t_0 + nh$ for $n = 0, 1, \ldots, N_h$ ($t_{N_h} \equiv t_f$).

**Remark 8.2** We indicate with $y_n = y(t_n)$ the *evaluation* of the (exact) solution $y(t)$ at $t = t_n$. Conversely, we indicate with $u_n$ the *approximation* of $y(t_n)$, i.e. the *numerical solution* at $t = t_n$.

### 8.2.1 Forward Euler method

The *forward Euler* method for the Cauchy problem (8.1) approximates $\dfrac{dy}{dt}(t_n)$ with forward finite differences for which $\dfrac{u_{n+1} - u_n}{h} = f(t_n, u_n)$ for $n = 0, 1, \ldots, N_h - 1$; the method reads:

$$\begin{cases} u_{n+1} = u_n + h f(t_n, u_n) & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases} \tag{8.3}$$

**Remark 8.3** The forward Euler method is an *explicit* method.

♦ **Example 8.3** For the *model problem* of Example 8.1, the forward Euler method reads:

$$\begin{cases} u_{n+1} = (1 + h\lambda)\, u_n & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

♦

♦ **Example 8.4** For the Cauchy problem (8.1) with $f(t, y) = e^{-t}\,(1 - y^{\alpha})$, for some $\alpha > 0$, the forward Euler method reads:

$$\begin{cases} u_{n+1} = u_n + h e^{-t_n}\,(1 - u_n^{\alpha}) & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

♦

### 8.2.2 Backward Euler method

The *backward Euler* method for the Cauchy problem (8.1) approximates $\dfrac{dy}{dt}(t_{n+1})$ with backward finite differences for which $\dfrac{u_{n+1} - u_n}{h} = f(t_{n+1}, u_{n+1})$ for $n = 0, 1, \ldots, N_h - 1$; the method reads:

$$\begin{cases} u_{n+1} = u_n + h f(t_{n+1}, u_{n+1}) & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases} \tag{8.4}$$

**Remark 8.4** The backward Euler method is an *implicit* method.

♦ **Example 8.5** For the *model problem* of Example 8.1, the backward Euler method reads:

$$\begin{cases} u_{n+1} = \dfrac{u_n}{1 - h\lambda} & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

♦

Since the backward Euler method is an implicit method, one needs, in principle, to solve a *nonlinear* equation for each $n = 0, 1, \ldots, N_h - 1$, i.e.:

$$\text{find } u_{n+1} \ : \ F_n^{BE}(u_{n+1}) = 0 \qquad \text{for all } n = 0, 1, \ldots, N_h - 1, \tag{8.5}$$

with $u_0 = y_0$, where $F_n^{BE}(y) := y - u_n - h f(t_{n+1}, y)$. Since Eq. (8.5) can not always be solved in closed form, an approximate solution can be determined e.g. by means of the Newton, fixed point iterations, or other methods for nonlinear equations. For example, by considering the Newton method (2.3) to solve Eq. (8.5), one should evaluate the first derivative of the nonlinear function $F_n^{BE}(y)$ in its argument $u$, which reads $\left(F_n^{BE}\right)'(y) = 1 - h \dfrac{\partial f}{\partial y}(t_{n+1}, y)$; then, the following algorithm can be used to solve an ODE with the backward Euler method in combination with the Newton method.

**Algorithm 8.1:** Backward Euler with Newton method

set $u_0 = y_0$;
**for** $n = 0, 1, \ldots, N_h - 1$ **do**
  set $u_{n+1}^{(0)} = u_n$;
  set $F_n^{BE}(y) = y - u_n - h f(t_{n+1}, y)$ and $\left(F_n^{BE}\right)'(y) = 1 - h \dfrac{\partial f}{\partial y}(t_{n+1}, y)$;
  **for** $k = 0, 1, \ldots$ *until a stopping criterion is satisfied* **do**
    $$u_{n+1}^{(k+1)} = u_{n+1}^{(k)} - \frac{F_n^{BE}\left(u_{n+1}^{(k)}\right)}{(F_n^{BE})'\left(u_{n+1}^{(k)}\right)};$$
  **end**
  set $u_{n+1} = u_{n+1}^{(k+1)}$;
**end**

♦ **Example 8.6** For the Cauchy problem (8.1) with $f(t, y) = e^{-t}(1 - y^{\alpha})$, for some $\alpha > 0$, we have from Eq. (8.5): $F_n^{BE}(y) = y - u_n - h e^{-t_{n+1}}(1 - y^{\alpha})$ and $\left(F_n^{BE}\right)'(y) = 1 + \alpha h e^{-t_{n+1}} y^{\alpha - 1}$. ♦

### 8.2.3 Crank–Nicolson method

The *Crank–Nicolson* method for the approximation of the Cauchy problem (8.1) considers the following approximation $\dfrac{u_{n+1} - u_n}{h} = \dfrac{1}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right]$ for $n = 0, 1, \ldots, N_h - 1$; the method reads:

$$\begin{cases} u_{n+1} = u_n + \dfrac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right] & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases} \tag{8.6}$$

**Remark 8.5** The Crank–Nicolson method is an *implicit* method.

Since the method is implicit, in principle, one needs to solve the following *nonlinear* equation for each $n = 0, 1, \ldots, N_h - 1$:

$$\text{find } u_{n+1} \ : \ F_n^{CN}(u_{n+1}) = 0 \qquad \text{for all } n = 0, 1, \ldots, N_h - 1,$$

with $u_0 = y_0$, where $F_n^{CN}(y) := y - u_n - \frac{h}{2}\left[f(t_n,u_n) + f(t_{n+1},y)\right]$. If the Newton method were to be used to solve such nonlinear problem, then one would need the first derivative of $F_n^{CN}(y)$, which reads $\left(F_n^{CN}\right)'(y) = 1 - \frac{h}{2}\frac{\partial f}{\partial y}(t_{n+1},y)$.

### 8.2.4 Heun method

The *Heun* method for the approximation of the Cauchy problem (8.1) reads:

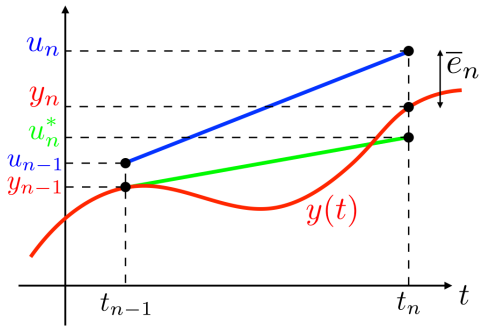
$$\begin{cases} u_{n+1}^* = u_n + h f(t_n,u_n) \\ u_{n+1} = u_n + \frac{h}{2}\left[f(t_n,u_n) + f(t_{n+1},u_{n+1}^*)\right] \\ u_0 = y_0. \end{cases} \qquad \text{for } n = 0,1,\ldots,N_h-1, \tag{8.7}$$

The Heun method can be interpreted as a modified Crank–Nicolson method with the implicit term $f(t_{n+1},u_{n+1})$ replaced by $f(t_{n+1},u_{n+1}^*)$, being $u_{n+1}^*$ the solution extrapolated by means of the forward Euler method.

**Remark 8.6** The Heun method is an *explicit* method.

### 8.2.5 Error analysis of the methods

By indicating with $\overline{e}_n = y_n - u_n$ the error at $t_n$, for some $n = 0,1,\ldots,N_h$, associated to a numerical method, we observe from the following Figure that this depends on two contributions.




In particular, we have $\overline{e}_n = y_n - u_n = (y_n - u_n^*) + (u_n^* - u_n)$. The component of the error $(y_n - u_n^*)$ is $(y_n - u_n^*) = h\,\tau_n(h)$, with $\tau_n(h)$ called the *local truncation error*, and it is associated to the consistency of the method. The solution $u_n^*$ indicates an extrapolated solution; for example, by considering the forward Euler method, one has $u_n^* = u_{n-1} + h f(t_{n-1},y_{n-1})$.

**Definition 8.2** The *error* associated to the numerical approximation of the Cauchy problem (8.1) at $t_n$ is $e_n := |y_n - u_n|$ for some $n = 0,1,\ldots,N_h$. If one has:

$$e_n \leq \widetilde{e}_n := C h^p,$$

with $\widetilde{e}_n$ the error estimator such that $C$ is a positive constant independent of $h$, then the method has *convergence order* $p > 0$ (order of accuracy of the method).

**Remark 8.7** If the solution of the Cauchy problem (8.1) is $y \in C^2(I)$, then the *forward* and *backward Euler* methods converge with *order* $p = 1$ in $h$.

**Remark 8.8** If the solution of the Cauchy problem (8.1) is $y \in C^3(I)$, then the *Crank–Nicolson* and *Heun* methods converge with *order* $p = 2$ in $h$.

In order to numerically determine the convergence order $p$ of an approximation method for ODEs, one can solve the Cauchy problem (8.1) for different values of $h = h_1, h_2, \ldots$ and, by knowing the

exact solution $y(t)$, compute the errors at $n = n_1, n_2, \ldots$ corresponding to the same $\bar{t} \in (t_0, t_f]$; in particular, $n_i = \dfrac{\bar{t} - t_0}{h_i}$ for some $i = 1, 2, \ldots$, if $n_i \in \mathbb{N}$. Then, one can estimate the convergence order as $p \simeq \log_{h_1/h_2} \dfrac{e_{n_1}}{e_{n_2}}$, for $h_1$ and $h_2$ "sufficiently" small.

### 8.2.6 Stability of the numerical methods: zero– and absolute stability

Regarding numerical methods for the approximation of ODEs, two concepts of stability should be considered, depending on the size of the interval $I$.

#### Zero–stability

Roughly speaking, the *zero–stability* is a property of a method in controlling the propagation of numerical perturbations for bounded intervals $I$ such that $|I| < +\infty$ or relatively small. In general, if $h$ is "sufficiently" small the zero–stability of a method is guaranteed.

**Definition 8.3** A numerical method for the approximation of ODEs is *zero–stable* if there exist $h_0 > 0$, $C > 0$ and, $\varepsilon_0 > 0$ such that, for all $h \in (0, h_0]$ and for all $\varepsilon \in (0, \varepsilon_0]$, if $|\rho_n| \leq \varepsilon$ for all $n = 0, 1, \ldots, N_h$, then $|z_n - u_n| \leq C\varepsilon$ for all $n = 0, 1, \ldots, N_h$; $\rho_n$ is the size of the perturbation introduced at the step $t_n$, $z_n$ is the solution than would be obtained by applying the numerical method to a perturbed ODE, $C$ is a constant independent of $h$, but dependent on $|I|$, while $\varepsilon$ the maximum size of the perturbation.

Based on the Lax–Richtmeyer equivalence Theorem 1.1, a consistent method for the approximation of ODEs is convergent if and only if zero–stable.

#### Absolute stability (stability on unbounded intervals)

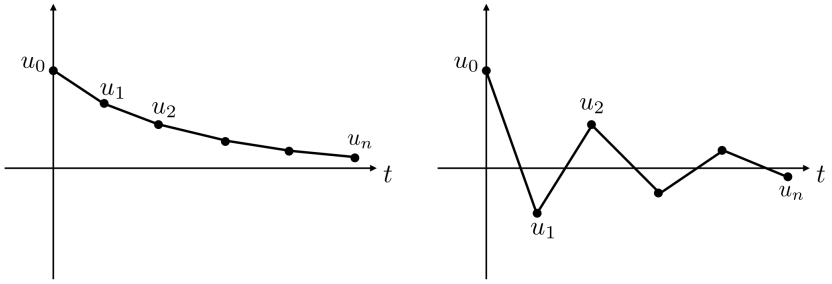
The *absolute stability* refers to unbounded intervals for which $t_f = +\infty$ or intervals for which $|I| < +\infty$, but very "large". In such instances, even if $h$ is fixed, then $\lim_{t_f \to +\infty} N_h = +\infty$; moreover, one can not choose $h$ as small as desired to ensure stability. Nevertheless, one may still be interested in controlling the propagation of numerical perturbations for $t_f$ tending to $+\infty$.

♦ **Example 8.7 Model problem in unbounded intervals**. Following Example 8.1, we consider the model problem on unbounded intervals, i.e. for which $t_f = +\infty$. In this case, from Eq. (8.2), it is straightforward to deduce that $\lim_{t \to +\infty} y(t) = 0$. ♦

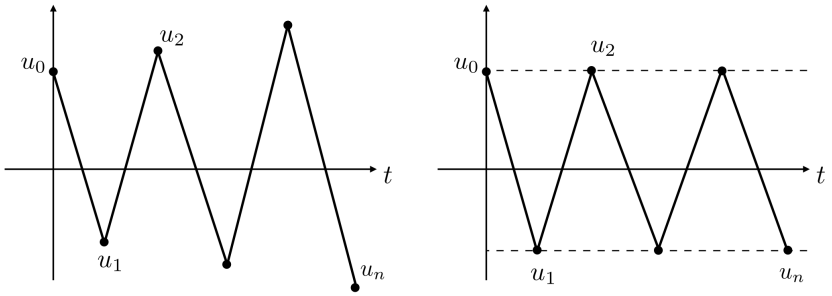
**Definition 8.4** The *absolute stability* is the property of a numerical method which yields $\lim_{n \to +\infty} u_n = 0$ for the *model problem* of Example 8.1 on unbounded intervals. The absolute stability is *unconditional* if $\lim_{n \to +\infty} u_n = 0$ for all $h > 0$, while *conditional* if $\lim_{n \to +\infty} u_n = 0$ for all $h > 0$ such that $h < h_{max}$, for some $h_{max} > 0$.

♦ **Example 8.8** Examples of absolutely stable and unstable numerical solutions.

Absolutely stable



Unstable




♦

**Definition 8.5** The *stability function* associated to a numerical method for the solution of the model problem of Example 8.1 is the complex function $R(z) : \mathbb{C} \to \mathbb{C}$ such that:

$$u_n = [R(h\lambda)]^n y_0 \qquad \text{for } n = 0, 1, \ldots \tag{8.8}$$

The stability function $R^{FE}(z) : \mathbb{C} \to \mathbb{C}$ for the *forward Euler* method is:

$$R^{FE}(z) = 1 + z;$$

indeed, from Eq. (8.3) applied to the model problem of Example 8.1, we have $u_{n+1} = (1 + h\lambda)u_n$ for $n = 0, 1, \ldots$, from which the result follows for $z = h\lambda$. The stability function $R^{BE}(z) : \mathbb{C} \to \mathbb{C}$ for the *backward Euler* method reads:

$$R^{BE}(z) = \frac{1}{1 - z},$$

since, from Eq. (8.4) applied to the model problem, we have $u_{n+1} = \dfrac{u_n}{1 - h\lambda}$ for $n = 0, 1, \ldots$. The stability function $R^{CN}(z) : \mathbb{C} \to \mathbb{C}$ for the *Crank–Nicolson* method is:

$$R^{CN}(z) = \frac{1 + z/2}{1 - z/2};$$

indeed, from Eq. (8.6) applied to the model problem, we have $u_{n+1} = \dfrac{1 + (h\lambda)/2}{1 - (h\lambda)/2} u_n$ for $n = 0, 1, \ldots$. Finally, the stability function $R^{H}(z) : \mathbb{C} \to \mathbb{C}$ for the *Heun* method reads:

$$R^{H}(z) = 1 + z + \frac{z^2}{2},$$

since for Eq. (8.7) applied to the model problem we have $u_{n+1} = \left[1 + h\lambda + \dfrac{(h\lambda)^2}{2}\right] u_n$ for $n = 0, 1, \ldots$.

From Eq. (8.8) it is straightforward to deduce that a method is *absolutely stable* if and only if $|R(z)| < 1$. More specifically, the method is *unconditionally absolutely stable* if:

$$|R(h\lambda)| < 1 \qquad \text{for all } h > 0;$$

instead, the method is *conditionally absolutely stable* if:

$$|R(h\lambda)| < 1 \qquad \text{for } 0 < h < h_{max}.$$

**Remark 8.9** The *forward Euler* method is *conditionally absolutely stable*, specifically for $0 < h < h_{max}$, with $h_{max} = \dfrac{2}{|\lambda|}$, as deduced by setting $\left|R^{FE}(h\lambda)\right| < 1$.

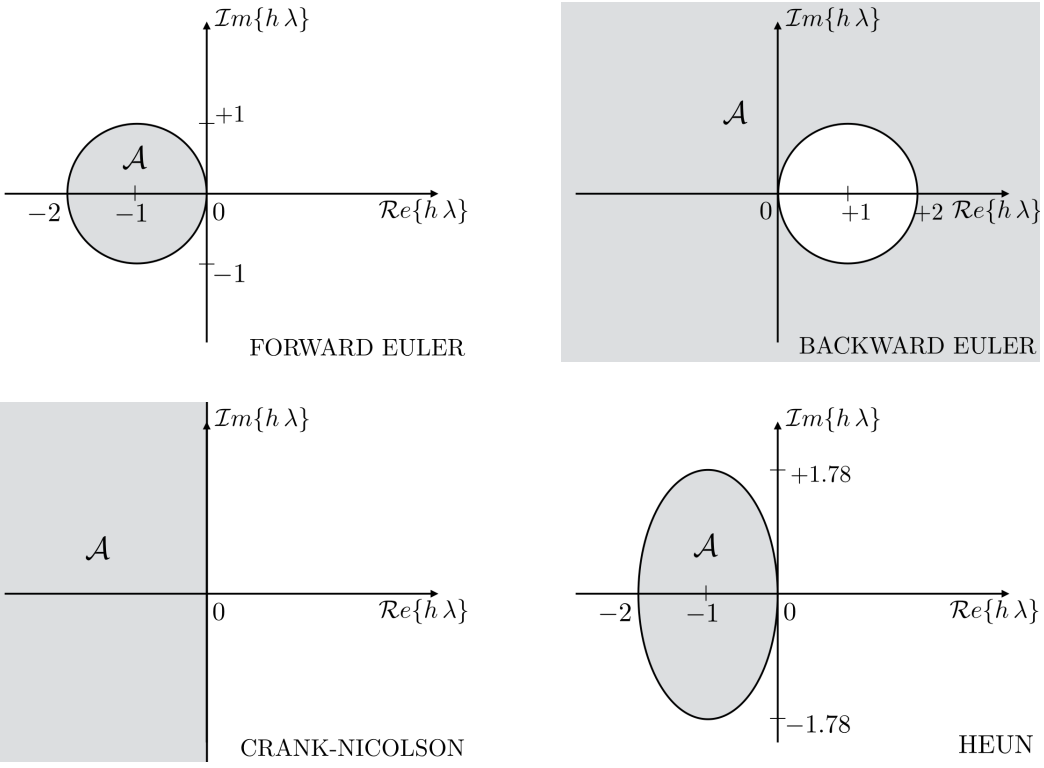
**Remark 8.10** The *backward Euler* method is *unconditionally absolutely stable*, as deduced by setting $\left|R^{BE}(h\lambda)\right| < 1$.

**Remark 8.11** The *Crank–Nicolson* method is *unconditionally absolutely stable*, as deduced by setting $\left|R^{CN}(h\lambda)\right| < 1$.

**Remark 8.12** The *Heun* method is *conditionally absolutely stable*, specifically for $0 < h < h_{max}$, with $h_{max} = \dfrac{2}{|\lambda|}$, as deduced by setting $\left|R^{H}(h\lambda)\right| < 1$.

**Definition 8.6** The *region of absolute stability* of a numerical method applied to the model problem of Example 8.1 is the set in the complex plane $\mathscr{A} := \{z \in \mathbb{C} : |R(z)| < 1\}$, where $R(z) : \mathbb{C} \to \mathbb{C}$ is the stability polynomial.

We highlight in the following Figure the regions $\mathscr{A}$ of absolute stability of some numerical methods.




**Definition 8.7** A numerical method is *$\mathscr{A}$–stable* if it is unconditionally absolutely stable for the model problem for all $\lambda \in \mathbb{C}$ such that $\mathscr{R}e\{\lambda\} < 0$.

**Remark 8.13** The *backward Euler* and *Crank-Nicolson* methods are $\mathscr{A}$-stable.

**Remark 8.14** Let us consider the general Cauchy problem (8.1) with $f$ continuously differentiable in the second argument, and such that $\frac{\partial f}{\partial y}(t,y(t)) < 0$ for all $t \in I$. In this case it does not make sense to require the method to be absolutely stable, because the exact solution $y(t)$ does not necessarily tend to zero as $t$ tends to infinity. However, if we consider $\widetilde{y}(t)$ the solution of the perturbed problem

$$\begin{cases} \frac{d\widetilde{y}}{dt}(t) = f(t,\widetilde{y}(t)) & \text{for all } t \in I, \\ \widetilde{y}(t_0) = y_0 + \rho_0, \end{cases}$$

the distance between the two solutions decreases as $t$ tends to infinity, $\lim_{t\to\infty} |y(t)-\widetilde{y}(t)| = 0$. Similarly, we want the numerical approximations $u_n$ of problem (8.1) and $\widetilde{u}_n$ of the perturbed problem, to satisfy $\lim_{n\to\infty} |u_n - \widetilde{u}_n| = 0$. This is always true for *backward Euler* and *Crank-Nicolson* methods, while for *forward Euler* and *Heun* methods a heuristic condition to obtain it is $0 < h < h_{max} = \frac{2}{\max_{t\in I} |\frac{\partial f}{\partial y}(t,y(t))|}$.

### 8.2.7 Runge-Kutta methods

*Runge-Kutta* methods indicate a family of one step methods for the numerical approximation of ODEs for which the approximate solution $u_{n+1}$ is determined by evaluating $f(t,y)$ at $s \geq 1$ *stages* in the interval $[t_n, t_{n+1}]$. The general Runge-Kutta method for the approximation of the Cauchy problem (8.1) reads:

$$\begin{cases} u_{n+1} = u_n + h\sum_{i=1}^{s} b_i K_i \quad \text{for } n = 0,1,\dots,N_h-1, \\ u_0 = y_0, \end{cases} \tag{8.9}$$
$$\text{where } K_i := f\left(t_n + c_i h, u_n + h\sum_{j=1}^{s} a_{ij}K_j\right) \quad \text{for } i = 1,\dots,s,$$

for some coefficients $\mathbf{c} = (c_1,\dots,c_s)^T \in \mathbb{R}^s, \mathbf{b} = (b_1,\dots,b_s)^T \in \mathbb{R}^s$, and $A \in \mathbb{R}^{s\times s}$, with $(A)_{ij} = a_{ij}$ for $i,j = 1,\dots,s$. These coefficients, which determine the Runge-Kutta method at hand, are stored in the so-called *Butcher's array* as:

| $\mathbf{c}$ | $A$ |
|---|---|
| | $\mathbf{b}^T$ |

We observe that if the matrix $A$ is stored from the bottom-left "corner", the Runge-Kutta method is *explicit* if $a_{ij} = 0$ for $j \geq i$ for all $i = 1,\dots,s$; otherwise, the Runge-Kutta method is *implicit*.

♦ **Example 8.9** We consider some examples of *explicit Runge-Kutta* methods with $s = 1,2$ and 4 stages.

- $s = 1, RK1$. In this case, Eq. (8.9) reads:
$$\begin{cases} u_{n+1} = u_n + hb_1K_1 \qquad \text{for } n = 0,1,\dots,N_h-1, \\ u_0 = y_0, \end{cases}$$
where $K_1 = f(t_n + c_1h, u_n + ha_{11}K_1)$; then, by setting $c_1 = 0, b_1 = 1$, and $a_{11} = 0$, i.e. with the following Butcher's array:

| 0 | 0 |
|---|---|
| | 1 |

we have the RK1 method, corresponding to the *forward Euler* method (8.3), being $K_1 = f(t_n,u_n)$.
- $s = 2, RK2$. In this case, Eq. (8.9) reads:
$$\begin{cases} u_{n+1} = u_n + hb_1K_1 + hb_2K_2 \qquad \text{for } n = 0,1,\dots,N_h-1, \\ u_0 = y_0, \end{cases}$$

where $K_1 = f(t_n + c_1 h, u_n + h a_{11} K_1 + h a_{12} K_2)$ and $K_2 = f(t_n + c_2 h, u_n + h a_{21} K_1 + h a_{22} K_2)$. Then, we consider the following Butcher's array:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

for which we obtain the RK2 method, i.e. the *Heun* method (8.7), being $K_1 = f(t_n, u_n)$ and $K_2 = f(t_{n+1}, u_n + h K_1)$.

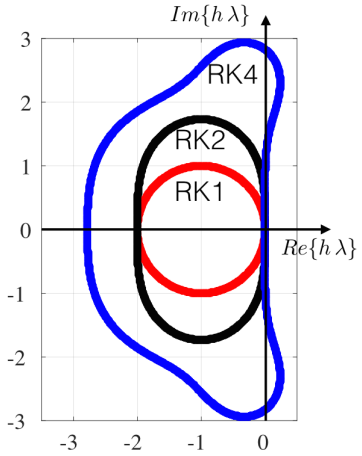
- $s = 4$, *RK4*. We consider the following Butcher's array:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

which yields:

$$\begin{cases} u_{n+1} = u_n + h\left(\dfrac{1}{6}K_1 + \dfrac{1}{3}K_2 + \dfrac{1}{3}K_3 + \dfrac{1}{6}K_4\right) & \text{for } n = 0, 1, \ldots, N_h - 1, \\ u_0 = y_0, \end{cases}$$

where $K_1 = f(t_n, u_n)$, $K_2 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2}K_1\right)$, $K_3 = f\left(t_n + \dfrac{h}{2}, u_n + \dfrac{h}{2}K_2\right)$, and $K_4 = f(t_{n+1}, u_n + h K_3)$, being $t_n + h \equiv t_{n+1}$. This RK4 method has order of accuracy 4.




Regions of absolute stability $\mathscr{A}$ for the *RK*1, *RK*2, and *RK*4 methods for the model problem (8.1). The regions $\mathscr{A}$ are inside the lines representing their boundaries.

♦

### 8.2.8 Multistep methods

*Multistep* methods indicate a family of methods for which the approximate solution $u_{n+1}$ is obtained by using $u_n, \ldots, u_{n-p}$ for some $p \geq 0$, being $p+1$ the number of *steps*. A multistep method for approximating the Cauchy problem (8.1) reads:

$$u_{n+1} = \sum_{j=0}^{p} a_j u_{n-j} + h \sum_{j=-1}^{p} b_j f(t_{n-j}, u_{n-j}) \quad \text{for } n = p, \ldots, N_h - 1, \tag{8.10}$$

given $u_0, \ldots, u_p$, for some coefficients $\{a_j\}_{j=0}^{p}$ and $\{b_j\}_{j=-1}^{p}$ which determine the multistep method. If $b_{-1} = 0$, the method is *explicit*, otherwise it is *implicit*. A multistep method is *consistent* if $\sum_{j=0}^{p} a_j = 1$ and $-\sum_{j=0}^{p} j a_j + \sum_{j=-1}^{p} b_j = 1$.

♦ **Example 8.10** We consider some one step methods, i.e. for $p = 0$, reading from Eq. (8.10):

$$u_{n+1} = a_0\, u_n + h\, b_{-1}\, f(t_{n+1}, u_{n+1}) + h\, b_0\, f(t_n, u_n) \quad \text{for } n = 0, 1, \ldots, N_h - 1,$$

given $u_0$. If $a_0 = 1$, $b_{-1} = 0$, and $b_0 = 1$, we obtain the *forward Euler* method (8.3); if $a_0 = 1$, $b_{-1} = 1$, and $b_0 = 0$, we have the *backward Euler* method (8.4). Finally, for $a_0 = 1$, $b_{-1} = \dfrac{1}{2}$, and $b_0 = \dfrac{1}{2}$, we obtain the *Crank–Nicolson* method (8.6). ♦

♦ **Example 8.11** We consider two common multistep methods.

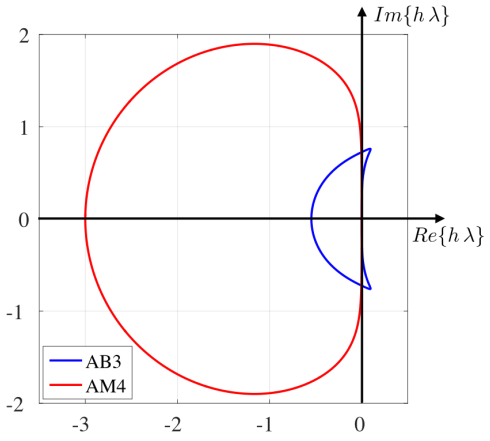
- *AB3* indicates the explicit *Adams–Bashforth* method with order of accuracy 3, which is a 3 steps method ($p = 2$). From Eq. (8.10), we have:

$$u_{n+1} = u_n + \frac{h}{12}\left[23\, f(t_n, u_n) - 16\, f(t_{n-1}, u_{n-1}) + 5\, f(t_{n-2}, u_{n-2})\right] \quad \text{for } n = 2, \ldots, N_h - 1,$$

  for which the coefficients are $a_0 = 1, a_1 = a_2 = 0, b_{-1} = 0, b_0 = \dfrac{23}{12}, b_1 = -\dfrac{16}{12}$, and $b_2 = \dfrac{5}{12}$.
- *AM4* indicates the implicit *Adams–Moulton* method with order of accuracy 4, which is a 3 steps method ($p = 2$). From Eq. (8.10), we have:

$$u_{n+1} = u_n + \frac{h}{24}\left[9\, f(t_{n+1}, u_{n+1}) + 19\, f(t_n, u_n) - 5\, f(t_{n-1}, u_{n-1}) + f(t_{n-2}, u_{n-2})\right],$$

  for $n = 2, \ldots, N_h - 1$, with $a_0 = 1, a_1 = a_2 = 0, b_{-1} = \dfrac{9}{24}, b_0 = \dfrac{19}{24}, b_1 = -\dfrac{5}{24}$, and $b_2 = \dfrac{1}{24}$.




Regions of absolute stability $\mathscr{A}$ for the *AB*3 and *AM*4 methods for the model problem (8.1). The regions $\mathscr{A}$ are inside the lines representing their boundaries.

♦

## 8.3 Numerical Approximation of Systems of Ordinary Differential Equations

We consider the numerical approximation of systems of ODEs. First, we define the Cauchy problem and then we consider the $\theta$–method for the approximation of these class of problems.

### 8.3.1 The Cauchy problem, examples, and definitions

**Definition 8.8** Let us consider the interval $I = (t_0, t_f) \subset \mathbb{R}$, then the vector–valued *Cauchy problem* reads:

$$\text{find } \mathbf{y} : I \subset \mathbb{R}^m \to \mathbb{R}^m \quad : \quad \begin{cases} \dfrac{d\mathbf{y}}{dt}(t) = \mathbf{f}(t, \mathbf{y}(t)) & \text{for all } t \in I, \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases} \tag{8.11}$$

where $m \geq 1$, $\mathbf{f}(t,\mathbf{y}) : I \times \mathbb{R}^m \to \mathbb{R}^m$, and $\mathbf{y}_0$ is the initial datum.

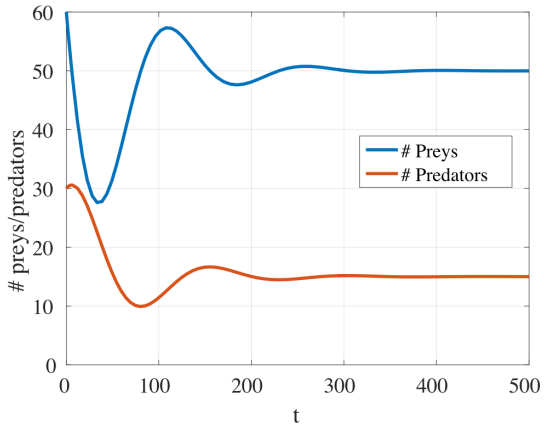
By referring to Eq. (8.11), we have $\mathbf{y}(t) = \left\{ \begin{array}{c} y_1(t) \\ \vdots \\ y_m(t) \end{array} \right\}$ and $\mathbf{f}(t,\mathbf{y}) = \left\{ \begin{array}{c} f_1(t,\mathbf{y}) \\ \vdots \\ f_m(t,\mathbf{y}) \end{array} \right\}$; moreover, we assume $\mathbf{f}(t,\mathbf{y})$ as continuous in both the arguments.

**Definition 8.9** If $\mathbf{f}(t,\mathbf{y}) : I \times \mathbb{R}^m \to \mathbb{R}^m$ is in the form:

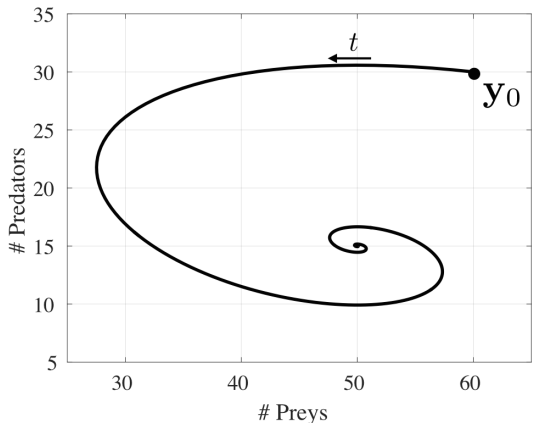
$$\mathbf{f}(t,\mathbf{y}) = A\mathbf{y} + \mathbf{g}(t),$$

with $A \in \mathbb{R}^{m \times m}$ and $\mathbf{g}(t) : I \to \mathbb{R}^m$, then, the system of ODEs is in *non homogeneous form* with *constant coefficients*, while in *homogeneous form* if $\mathbf{g}(t) = \mathbf{0}$ for all $t \in (t_0, t_f]$.

♦ **Example 8.12** *Lotka–Volterra* prey–predator model. By referring to Eq. (8.11), we consider $\mathbf{f}(t,\mathbf{y}) = A\mathbf{y} + \mathbf{g}(t)$, i.e. a system in non homogeneous form with constant coefficients; specifically, $m = 2$, $A = \begin{bmatrix} -b_1 C_1 & -d_2 C_1 \\ d_1 C_2 & b_2 C_2 \end{bmatrix}$, and $\mathbf{g} = (C_1, -C_2)^T$, with $b_1$, $b_2$, $d_1$, $d_2$, $C_1$, and $C_2$ suitable coefficients. The model can be used to determine the evolution in time $t$ of preys $y_1(t)$ and predators $y_2(t)$ in a closed environment. We set $b_1 = \frac{1}{100}$, $b_2 = 0$, $d_1 = \frac{1}{50}$, $d_2 = \frac{1}{30}$, $C_1 = 3$, $C_2 = 1$, $\mathbf{y}_0 = (60, 30)^T$, $t_0 = 0$, and $t_f = 500$.




Number of preys and predators vs. time




Trajectory in the *phases plane*

♦

## 8.3.2 $\theta$–method

Analogously to the case $m = 1$, we consider the partition of $\bar{I} = [t_0, t_f]$ into $N_h$ subintervals of equal size $h = \frac{t_f - t_0}{N_h}$ for which $t_n = t_0 + nh$ for $n = 0, 1, \ldots, N_h$. Then, we indicate with $\mathbf{y}_n = \mathbf{y}(t_n)$ the *evaluation* of $\mathbf{y}(t)$ at $t = t_n$, while with $\mathbf{u}_n$ the *numerical solution* at $t = t_n$.

Let us introduce the parameter $\theta \in \mathbb{R}$ such that $\theta \in [0, 1]$; then, the $\theta$–method for the approximation of the Cauchy problem (8.11) reads:

$$\left\{ \begin{array}{ll} \mathbf{u}_{n+1} = \mathbf{u}_n + h\left[(1-\theta)\mathbf{f}(t_n, \mathbf{u}_n) + \theta\,\mathbf{f}(t_{n+1}, \mathbf{u}_{n+1})\right] & \text{for } n = 0, 1, \ldots, N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{array} \right. \tag{8.12}$$

**Remark 8.15** The $\theta$–method is *explicit* for $\theta = 0$, while is an *implicit* method for $\theta \in (0,1]$.

**Remark 8.16** If the system is in *non homogeneous form* with *constant coefficients* according with Definition 8.9, then the $\theta$–method reads:

$$\begin{cases} (I - h\,\theta\,A)\,\mathbf{u}_{n+1} = [I + h(1-\theta)A]\,\mathbf{u}_n + h\,[(1-\theta)\,\mathbf{g}(t_n) + \theta\,\mathbf{g}(t_{n+1})] & \text{for } n = 0,1,\ldots,N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{cases}$$

We observe that we need to solve a linear system with the matrix $(I - h\,\theta\,A)$ for each $n = 0,1,\ldots,N_h - 1$, unless $\theta = 0$ (i.e. when the method is explicit).

For $\theta = 0$, the $\theta$–method coincides with the *forward Euler* method for systems of ODEs, reading:

$$\begin{cases} \mathbf{u}_{n+1} = \mathbf{u}_n + h\,\mathbf{f}(t_n, \mathbf{u}_n) & \text{for } n = 0,1,\ldots,N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{cases}$$

For $\theta = 1$, the $\theta$–method yields the *backward Euler* method, which reads:

$$\begin{cases} \mathbf{u}_{n+1} = \mathbf{u}_n + h\,\mathbf{f}(t_{n+1}, \mathbf{u}_{n+1}) & \text{for } n = 0,1,\ldots,N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{cases}$$

Similarly, for $\theta = \dfrac{1}{2}$ in Eq. (8.12), one obtains the *Crank–Nicolson* method:

$$\begin{cases} \mathbf{u}_{n+1} = \mathbf{u}_n + \dfrac{h}{2}\,[\mathbf{f}(t_n, \mathbf{u}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}_{n+1})] & \text{for } n = 0,1,\ldots,N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{cases}$$

**Definition 8.10** The *error* associated to the numerical approximation of the Cauchy problem (8.11) at $t = t_n$ is $e_n := \|\mathbf{y}_n - \mathbf{u}_n\|$ for some $n = 0,1,\ldots,N_h$. If one has:

$$e_n \leq \widetilde{e}_n := C\,h^p,$$

with $\widetilde{e}_n$ the error estimator, being $C$ a positive constant independent of $h$, then the method has *convergence order* $p > 0$ (order of accuracy of the method).

**Remark 8.17** If the solution of the Cauchy problem (8.11) is $\mathbf{y} \in C^2(I)$, then the *forward* ($\theta = 0$) and *backward* ($\theta = 1$) *Euler* methods converge with *order* $p = 1$ in $h$.

**Remark 8.18** If $\mathbf{y} \in C^3(I)$, the *Crank–Nicolson* ($\theta = \dfrac{1}{2}$) method converges with *order* $p = 2$.

**Remark 8.19** Even if it is not a $\theta$–method according to Eq. (8.12), one can deduce the *Heun* method for systems of ODEs by applying Eq. (8.7) to Eq. (8.11), thus reading:

$$\begin{cases} \mathbf{u}^*_{n+1} = \mathbf{u}_n + h\,\mathbf{f}(t_n, \mathbf{u}_n), \\ \mathbf{u}_{n+1} = \mathbf{u}_n + \dfrac{h}{2}\,\left[\mathbf{f}(t_n, \mathbf{u}_n) + \mathbf{f}(t_{n+1}, \mathbf{u}^*_{n+1})\right] & \text{for } n = 0,1,\ldots,N_h - 1, \\ \mathbf{u}_0 = \mathbf{y}_0. \end{cases}$$

Let us consider a system of ODEs in *non homogeneous form* with *constant coefficients* according with Definition 8.9, for which $\mathbf{f}(t,\mathbf{y}) = A\mathbf{y}+\mathbf{g}(\mathbf{t})$. By assuming that the eigenvalues of $A$ satisfy $\mathscr{R}e\{\lambda_i(A)\} < 0$ for all $i = 1,\dots,m$, then a numerical method is absolutely stable if the condition on the stability function is satisfied for all the eigenvalues, that is

$$|R(h\lambda_i(A))| < 1, \quad \text{for } i = 1,\dots,m,$$

where $R$ is the stability function in Definition 8.5. In particular, the *forward Euler* method is *conditionally absolutely stable* for $h > 0$ such that

$$|1 + h\lambda_i(A)| < 1, \quad \text{for } i = 1,\dots,m,$$

while *Heun* method is *conditionally absolutely stable* for $h > 0$ such that

$$\left|1 + h\lambda_i(A) + \frac{(h\lambda_i(A))^2}{2}\right| < 1, \quad \text{for } i = 1,\dots,m.$$

Backward Euler and Crank-Nicolson methods are *unconditionally absolutely stable*.

In the case a general function $\mathbf{f}(t,\mathbf{y}) : I \times \mathbb{R}^m \to \mathbb{R}^m$ is provided, the eigenvalues of $A$ are replaced by those of the *Jacobian matrix* $J(t) = \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}}(t,\mathbf{y}(t))$. Assuming that $\mathscr{R}e\{\lambda_i(J(t))\} < 0$ for all $i = 1,\dots,m$ and for all $t \in I$, a numerical method for the solution of the system of ODEs is absolutely stable if

$$\left|R\big(h\lambda_i(J(t))\big)\right| < 1, \quad \text{for } i = 1,\dots,m, \text{ and } t \in I,$$

where $R$ is again the stability function in Definition 8.5.

## 8.4 Numerical Approximation of High Order Ordinary Differential Equations

So far, we considered first order ODEs or systems of first order ODEs, namely the Cauchy problems (8.1) and (8.11), respectively. *High order* ODEs and systems of high order ODEs, which are often used as mathematical models for problems of physical interest, can be rewritten as systems of first order ODEs. Therefore, the numerical solution of high order ODEs can simply be obtained by rewriting such problems as systems of first order ODEs, which are then approximated by means of the numerical methods considered for this class of problems, as e.g. the $\theta$–method; see Sec. 8.3.2.

### 8.4.1 Second order ODEs

Let us consider the interval $I = (t_0, t_f) \subset \mathbb{R}$, then a *second order* ODE reads:

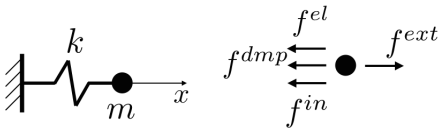
$$\text{find } y : I \subset \mathbb{R} \to \mathbb{R} : \begin{cases} \dfrac{d^2y}{dt^2}(t) = f\left(t, y(t), \dfrac{dy}{dt}(t)\right) & \text{for all } t \in I, \\ \dfrac{dy}{dt}(t_0) = y_{1,0}, \\ y(t_0) = y_{0,0}, \end{cases} \tag{8.13}$$

where $f(t,y,w_2) : I \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a given function with three arguments, with $y_{0,0}$ and $y_{1,0}$ the initial data. We introduce now the auxiliary variable $w_2(t) : I \to \mathbb{R}$ such that $w_2(t) = \frac{dy}{dt}(t)$ for all $t \in I$. Then, problem (8.13) can be rewritten as the following system of ODEs:

$$\text{find } y, w_2 : I \subset \mathbb{R} \to \mathbb{R} : \begin{cases} \dfrac{dw_2}{dt}(t) = f(t, y(t), w_2(t)) & \text{for all } t \in I, \\ \dfrac{dy}{dt}(t) = w_2(t) & \text{for all } t \in I, \\ w_2(t_0) = y_{1,0}, \\ y(t_0) = y_{0,0}, \end{cases}$$

and, in general, as the Cauchy problem (8.11) for $m = 2$, $\mathbf{y}(t) = \begin{Bmatrix} w_2(t) \\ y(t) \end{Bmatrix}$, $\mathbf{f}(t,\mathbf{y}) = \begin{Bmatrix} f(t,y,w_2) \\ w_2(t) \end{Bmatrix}$, and $\mathbf{y}_0 = \begin{Bmatrix} y_{1,0} \\ y_{0,0} \end{Bmatrix}$; we notice that $\mathbf{y} : I \to \mathbb{R}^2$ and $\mathbf{f}(t,\mathbf{y}) : I \times \mathbb{R}^2 \to \mathbb{R}^2$.

♦ **Example 8.13** Let us consider the problem of the dynamics of a concentrated mass anchored through a spring.



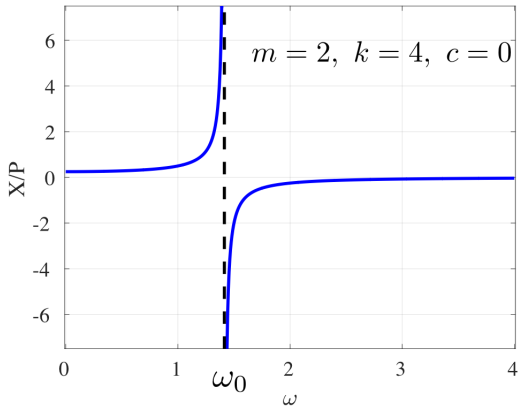

We consider the external force $f^{ext}(t)$, the inertial force $f^{in}(t) = m\ddot{x}(t)$, with $m$ the mass, the elastic force $f^{el}(t) = kx(t)$, with $k$ the elastic constant of the spring, and the damping force $f^{dmp}(t) = c\dot{x}(t)$, with $c \geq 0$; $x(t)$ represents the position of the concentrated mass over time $t$.

The problem corresponds to the following second order ODE in the time interval $I = (t_0, t_f)$:

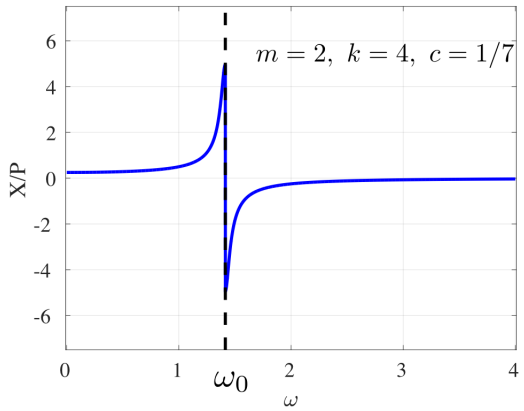
$$\text{find } x : I \subset \mathbb{R} \to \mathbb{R} \quad : \quad \begin{cases} m\ddot{x}(t) + c\dot{x}(t) + kx(t) = f^{ext}(t) & \text{for all } t \in I, \\ \dot{x}(t_0) = v_0, \\ x(t_0) = x_0, \end{cases}$$

where $v_0$ is the initial velocity of the mass, while $x_0$ its initial position. This second order ODE can be recast in a system of first order ODEs, i.e. in the form of the Cauchy problem (8.11) with two variables, by introducing the (auxiliary) velocity variable $v : I \to \mathbb{R}$, i.e. for $\mathbf{y}(t) = \begin{Bmatrix} v(t) \\ x(t) \end{Bmatrix}$, $\mathbf{f}(t, \mathbf{y}) = \begin{Bmatrix} -\frac{c}{m}v - \frac{k}{m}x + \frac{1}{m}f^{ext}(t) \\ v \end{Bmatrix}$, and $\mathbf{y}_0 = \begin{Bmatrix} v_0 \\ x_0 \end{Bmatrix}$. We observe that this system of ODEs is in non homogeneous form with constant coefficients according to Definition 8.9, i.e. $\mathbf{f}(t, \mathbf{y}) = A\mathbf{y} + \mathbf{g}(t)$, with $A = \begin{bmatrix} -\frac{c}{m} & -\frac{k}{m} \\ 1 & 0 \end{bmatrix}$ and $\mathbf{g}(t) = \begin{Bmatrix} \frac{1}{m}f^{ext}(t) \\ 0 \end{Bmatrix}$.

Let us now assume that $f^{ext}(t) = P\sin(\omega t)$ for some $P > 0$ and $\omega > 0$. Then, one can show that, after an initial transitory, the solution $x(t)$ assumes the form $\widetilde{x}(t) = X\sin(\omega t - \phi)$, where $\tan(\phi) = \dfrac{c\,\omega}{k - \omega^2 m}$ and $X = \dfrac{P}{(k - \omega^2 m)\cos(\phi) + c\,\omega\sin(\phi)}$. If $c = 0$, we have $\phi = 0$ and $\widetilde{x}(t) = X\sin(\omega t)$, with $X = \dfrac{P}{(k - \omega^2 m)}$; $\omega_0 = \sqrt{\dfrac{k}{m}}$ is called *natural angular frequency* of the system, while $f_0 = \dfrac{\omega_0}{2\pi}$ is the natural frequency. If $\omega = \omega_0$ (or nearly), then the system is in *resonance* conditions.




Ratio $\dfrac{X}{P}$ vs. $\omega$ for $m = 2$, $k = 4$, and $c = 0$.




Ratio $\dfrac{X}{P}$ vs. $\omega$ for $m = 2$, $k = 4$, and $c = \dfrac{1}{7}$.

♦

### 8.4.2 General high order ODEs

We consider again the interval $I = (t_0, t_f) \subset \mathbb{R}$, then a *high order* ODE, with order $m \geq 2$, reads:

$$\text{find } y : I \subset \mathbb{R} \to \mathbb{R} \quad : \quad \begin{cases} \dfrac{d^m y}{dt^m}(t) = f\left(t, y(t), \dfrac{dy}{dt}(t), \ldots, \dfrac{d^{m-1}y}{dt^{m-1}}(t)\right) & \text{for all } t \in I, \\ \dfrac{d^{m-1}y}{dt^{m-1}}(t_0) = y_{m-1,0}, & \\ \vdots & \\ y(t_0) = y_{0,0}, & \end{cases} \tag{8.14}$$

where $f(t, y, w_2, \ldots, w_m) : I \times \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R} \to \mathbb{R}$ is a given function with $m+1$ arguments, with $\{y_{k,0}\}_{k=0}^{m-1}$ the initial data. Now, we introduce the auxiliary variables $w_k(t) : I \to \mathbb{R}$ such that $w_k(t) = \dfrac{d^{k-1}y}{dt^{k-1}}(t)$ for all $t \in I$ and for $k = 2, \ldots, m$. In this manner, problem (8.14) can be rewritten as the following system of ODEs:

$$\text{find } y, w_2, \ldots, w_m : I \subset \mathbb{R} \to \mathbb{R} \quad : \quad \begin{cases} \dfrac{dw_m}{dt}(t) = f(t, y(t), w_2(t), \ldots, w_m(t)) & \text{for all } t \in I, \\ \dfrac{dw_{m-1}}{dt}(t) = w_m(t) & \text{for all } t \in I, \\ \vdots & \\ \dfrac{dy}{dt}(t) = w_2(t) & \text{for all } t \in I, \\ w_m(t_0) = y_{m-1,0}, & \\ \vdots & \\ w_2(t_0) = y_{1,0}, & \\ y(t_0) = y_{0,0}. & \end{cases}$$

The previous system can be recast in the general form of the Cauchy problem (8.11) for $\mathbf{y}(t) = \begin{Bmatrix} w_m(t) \\ \vdots \\ w_2(t) \\ y(t) \end{Bmatrix}$, $\mathbf{f}(t, \mathbf{y}) = \begin{Bmatrix} f(t, y, w_2, \ldots, w_m) \\ w_m \\ \vdots \\ w_2 \end{Bmatrix}$, and $\mathbf{y}_0 = \begin{Bmatrix} y_{m-1,0} \\ \vdots \\ y_{0,0} \end{Bmatrix}$; we observe that $\mathbf{y} : I \to \mathbb{R}^m$ and $\mathbf{f}(t, \mathbf{y}) : I \times \mathbb{R}^m \to \mathbb{R}^m$.

**Remark 8.20** Systems of high order ODEs can be rewritten into systems of first order ODEs, following the above procedure. If $n$ is the size of the system of $p$–order ODEs, then the corresponding system of first order ODEs (8.11) has dimension $m = pn$.

# Bibliography

[1] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, 2nd ed. Texts in Applied Mathematics 37, Springer, Berlin and Heidelberg, 2007.

[2] Alfio Quarteroni, Fausto Saleri, and Paola Gervasio. *Scientific Computing with MATLAB and Octave*, 4th ed. Texts in Computational Science and Engineering 2, Springer–Verlag, Berlin and Heidelberg, 2014.