Class notes

Numerical Analysis

Prof. Annalisa Buffa

2021-2022

Contents

Chapter 1

Representation of numbers

The aim of this section is to understand how numbers are represented in computers. Indeed, the set of real numbers is infinite, but computers can only represent and work with a finite subset. Therefore, we need to understand which numbers are representable and how the operations are performed in this set of representable real numbers.

1.1 Representation of real numbers

When representing numbers, the so-called positional notation is usually used. This means that in the string that identifies a number, the position of the digits matter as much as their value in order to define the number.

Example 1.1. *The string* 129.45 *defines the number*

$$129.45 = 1\cdot 10^2 + 2\cdot 10^1 + 9\cdot 10^0 + 4\cdot 10^{-1} + 5\cdot 10^{-2}.$$

A number in positional notation is written with respect to a base β. If not specified, as in the example above, it is usually taken equal to 10, but it could also be chosen differently. When the base is different from 10, it will be specified by putting the number between brackets and by writing the base in the lower right corner.

Example 1.2. *Take* $\beta = 3$*; in this case, the digits used are only* 0, 1 *and* 2, *and a number can be written for instance as follows:*

$$(102.1)_3 = \underbrace{1\cdot 3^2 + 0\cdot 3^1 + 2\cdot 3^0 + 1\cdot 3^{-1}}_{\textit{in base } 10} = \frac{34}{3} = \left(\frac{34}{3}\right)_{10}.$$

Computers also use a positional notation, but with $\beta = 2$. That is, each representable number should be written as a finite serie of 0's and 1's.

Let $\alpha_n, \alpha_{n-1}, \ldots, \alpha_0\, \alpha_{-1}, \ldots$ be digits. Multiplication by β moves the "decimal" dot by one position to the right:

$$(\pm\alpha_n\alpha_{n-1}\cdots\alpha_1\alpha_0\underset{\uparrow}{.}\alpha_{-1}\alpha_{-2}\cdots)_\beta\cdot\beta=(\pm\alpha_n\alpha_{n-1}\cdots\alpha_1\alpha_0\alpha_{-1}\underset{\uparrow}{.}\alpha_{-2}\cdots)_\beta,$$

while division by β moves it to the left. From this observation, one can write any number in the so-called normalized representation of the real numbers:

Definition 1.1 (Normalized representation)**.** *The normalized representation of a real number consists in multiplying a number by the correct power of β so that the first non-zero digit appears right after the dot:*

$$(\pm\alpha_n\alpha_{n-1}\cdots\alpha_1\alpha_0\underset{\uparrow}{.}\alpha_{-1}\alpha_{-2}\cdots)_\beta=(\pm 0\underset{\uparrow}{.}\alpha_n\alpha_{n-1}\cdots\alpha_1\alpha_0\alpha_{-1}\alpha_{-2}\cdots)_\beta\cdot\beta^{n+1},$$

with $\alpha_n\neq 0$.

Example 1.3.

$$\pi=3.1415\ldots=0.31415\ldots\cdot 10^1;\qquad 245=0.245\cdot 10^3;\qquad 0.012=0.12\cdot 10^{-1}.$$

Theorem 1.1. *A rational number written as an irreducible fraction $\frac{p}{q}$ has a finite representation in base β if and only if the prime factors of the denominator q divide β.*

Example 1.4.

- *In the base $\beta=10$, a rational number has a finite representation if and only if it can be written as $\pm\frac{p}{2^n5^m}$ for some $p,n,m\in\mathbb{N}$. This corresponds to the known fact that in base 10, if the divider has a factor which is neither 2 nor 5, then the Euclidean division does not finish and becomes periodic.*
- *In the base $\beta=2$ instead, to have a finite representation, a number needs to be written as $\pm\frac{p}{2^n}$ for some $p,n\in\mathbb{N}$. For instance, $\frac{1}{5}=(0.2)_{10}$ does not have a finite representation in base 2. This can be checked by directly computing the Euclidean division in base 2: since $5=(101)_2$, then,*

$$\begin{array}{r|l} 1 & 101 \\ \cline{2-2} & \end{array} = \begin{array}{r|l} 1000 & 101 \\ \cline{2-2} & 0.00 \end{array} = \begin{array}{r|l} 1000 & 101 \\ \cline{2-2} 101 & 0.0011 \\ \cline{1-1} 110 & \\ 101 & \\ \cline{1-1} 1 & \end{array}$$

and therefore, $\frac{1}{5}=(0.00110011\ldots)_2=\left(0.00\overline{11}\right)_2$.

1.2 Representation of numbers in computers

In a computer, the numbers that can be represented and that are used to perform any operation have to belong to a finite set. For the results produced by the computer to be, within certain limits, the same as the ones given by the exact arithmetic in $\mathbb{R}$, the set of representable numbers has to be chosen carefully.

Floating-point representation

Let β be a base, $t \in \mathbb{N} \setminus \{0\}$ be a number of digits, and $L \leq U$ be two integers (usually, $L < 0$ and $U > 0$). We call $\mathscr{F}(\beta, t, L, U)$ the finite set of rational numbers which are written in the following normalized form, called floating-point representation:

$$(-1)^s\,(0.\alpha_1\alpha_2\ldots\alpha_t)_\beta\,\beta^e,$$

where

- s is the sign, $s \in \{0, 1\}$,
- $\alpha_1\alpha_2\ldots\alpha_t$ is the mantissa with $\alpha_1 \neq 0$, $\alpha_i \in \left\{0, 1, \ldots, \beta - 1\right\}$ for $i = 1, \ldots, t$ (since β is the base),
- e is the exponent with $L \leq e \leq U$.

The number zero is added apart since it does not admit such a representation.
It should be clear that it is commonly accepted that the "value" of a number corresponds to its representation in base 10. So when we "compute" $(-1)^s\,(0.\alpha_1\alpha_2\ldots\alpha_t)\,\beta^e$, we are converting the number in base 10.

Example 1.5. *When one writes* $(-1)^0(0.1011)_2 \cdot 2^3$*, it is the floating point representation in base* 2 *of the number whose "value" (in base* 10*) is equal to* $1 \cdot 2^{-1+3} + 1 \cdot 2^{-3+3} + 1 \cdot 2^{-4+3} = 5.5$*. This number belongs for example to* $\mathscr{F}(2, 4, -1, 4)$*, but not to* $\mathscr{F}(2, 3, -1, 4)$*, nor to* $\mathscr{F}(2, 4, -2, 2)$.

Example 1.6. *Let us consider the set* $\mathscr{F} = \mathscr{F}(10, 3, -2, 2)$*. In this case, we have* $\beta = 10$*,* $t = 3$*, and* $-2 \leq e \leq 2$*. Let us see which of the following numbers belong to* $\mathscr{F}$*:*

$$\begin{aligned}
23.4 &= (-1)^0\,(0.234)\,10^2 = 23.4 && \in \mathscr{F} \\
-53.8 &= (-1)^1\,(0.538)\,10^2 && \in \mathscr{F} \\
3.141 &= (-1)^0\,(0.3141)\,10^1 && \notin \mathscr{F} \text{ since there are 4 significant digits, not 3} \\
3.1 &= (-1)^0\,(0.310)\,10^1 && \in \mathscr{F}
\end{aligned}$$

Therefore, a set $\mathscr{F}(\beta, t, L, U)$ only contains some rational numbers (not all of them) that have a finite representation in base β. From Theorem 1.1, a rational number written as an irreducible fraction $\frac{p}{q}$ belongs to $\mathscr{F}(\beta, t, L, U)$ only if the prime factors of q are divisors of β. Note that this is a necessary but not sufficient condition.

In modern computers, double-precision typically corresponds to numbers in the set

$$\mathscr{F}(2, 53, -1021, 1024),$$

and it uses memory as follows:

- 1 bit for the sign;
- 52 bits to represent the mantissa: indeed, α_1 is always equal to 1 since it has to be non-zero;
- 11 bits for the exponent:
    - 1 bit for its sign;
    - 10 bits for its value.

Consequently, the total number of bits used is $1+52+11=64$ bits, which corresponds to 8 bytes. Thus, every number needs 8 bytes to be represented.

Note that from Theorem 1.1, all the numbers in $\mathscr{F}(2,53,-1021,1024)$, are of the form $\frac{p}{2^n}$ with p in a bounded subset of $\mathbb{N}$. Consequently, only a few rational numbers are exactly represented in a computer.

Example 1.7.
$$\frac{1}{10}=\frac{1}{2\cdot 5}\neq\frac{p}{2^n}\notin\mathscr{F}(2,53,-1021,1024).$$

Example 1.8. *Let us run the following computation in MATLAB:*

```
>> 0.3 - 0.2 - 0.1
     ans =
          -2.7756e-17
```

We would have expected the result to be zero, but it is not. Indeed, computers are in base $\beta=2$. Therefore, from Theorem 1.1, the only numbers that can be represented by $\mathscr{F}(2,t,L,U)$ are numbers that can be written $\frac{p}{2^n}$ with p odd. Thus,

$$(0.1)_{10}=\frac{1}{2\cdot 5}\text{ is not in }\mathscr{F}(2,t,L,U),$$

no matter which t, L and U are chosen. This is also true for $(0.2)_{10}$ and $(0.3)_{10}$. Consequently, since none of the three numbers of the computation are exactly represented in the computer, we cannot expect the MATLAB result to be equal to the one obtained by exact arithmetic.

Example 1.9 (No associativity)**.** *The operation computed in Example 1.8 is performed in the following order:*

```
>> (0.3 - 0.2) - 0.1
     ans =
          -2.7756e-17
```

If the terms are grouped in an other way, the results changes:

```
>> 0.3 - (0.2 + 0.1)
     ans =
          -5.5511e-17
```

Therefore, not only the result is not exact, but the associative property is also not verified.

Let us now study into details the structure of $\mathcal{F} := \mathcal{F}(\beta, t, L, U)$, what is in $\mathcal{F}$ and what is not. First of all, $\mathcal{F}$ is symmetric with respect to the origin, it is therefore enough to study its structure for positive numbers only.

The smallest and the largest (positive) value in $\mathcal{F}$ are:

$$x_{\min} = (0.10\ldots0)_\beta \cdot \beta^L = \beta^{L-1}$$
$$x_{\max} = \Big(0.\underbrace{\beta-1\,\beta-1\ldots\beta-1}_{t \text{ times}}\Big)_\beta \cdot \beta^U$$

The smallest and the largest (positive) double-precision numbers in a modern computer consequently are $x_{\min} = 2^{-1022} \approx 2\cdot 10^{-308}$ and $x_{\max} = (0.11\ldots1)_2 \cdot 2^{1024} \approx 1.8\cdot 10^{308}$.

Example 1.10 (Large numbers)**.** *Let us try to compute* 10^{50} *with* *MATLAB:*

```
>> format rat
>> 10^50
    ans =
        100000000000000007629769841091887003294964970946560
```

Thus 10^{50} *is not exactly represented. Indeed, let us see for which* $n \in \mathbb{N}$, $10^n \in \mathcal{F}(2, 53, -1021, 1024)$. *Since* $10^n = 5^n \cdot 2^n$, *and since the considered base is* 2, *then* 5^n *plays the role of the mantissa. So for* 10^n *to be in* $\mathcal{F}(2, 53, -1021, 1024)$, *we need*

$$5^n \le (\underbrace{11\ldots11}_{53\ \textit{times}})_2 = \textit{ maximum mantissa } = 2^{53} - 1,$$

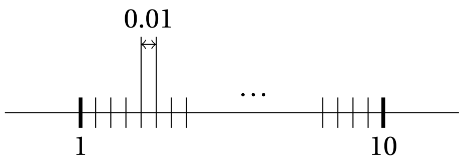
that is n should be at most equal to 22*. Therefore, the largest power of* 10 *exactly represented in a modern computer in double-precision is* 10^{22}.

Spacing
Spacing stands for the distance between two consecutive numbers represented in $\mathcal{F}$.

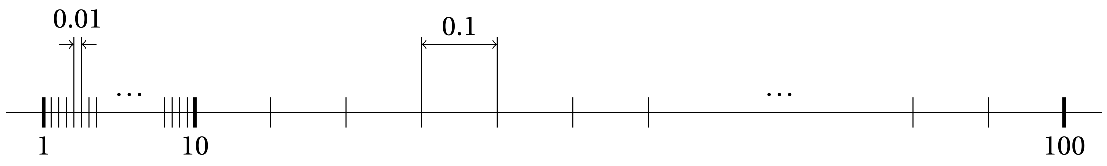
To get an intuition on it, let us first look at which are the representable numbers close to 1 when $\beta = 10$ and $t = 3$. First of all, $1 = 0.100\cdot 10^1 \in \mathcal{F}$. Moreover, the representable number coming right after 1 in $\mathcal{F}$ is obtained by adding 1 to the last digit in the mantissa. Thus, it is $0.101\cdot 10^1 = 1.01$. Its distance to 1 is

$$1.01 - 1 = 0.01 = \frac{1}{100}.$$

The successive numbers are $1.02, 1.03, \ldots$, until $9.99 = 0.999\cdot 10^1$. The representable number that comes after $0.999\cdot 10^1$ is obtained by adding 1 to the exponent and change the mantissa to 100 to obtain $0.100\cdot 10^2 = 10$. Consequently, the numbers in $\mathcal{F}$ ranging from 1 to 10 are evenly spaced with spacing 0.01.



A similar situation arises in the interval $[10,100]$: the number coming right after $10 = 0.100 \cdot 10^2$ in $\mathscr{F}$ is $0.101 \cdot 10^2 = 10.1$, and its distance to 10 is 0.1. With the same process as before, we recursively increase by 1 the mantissa until the number $0.999 \cdot 10^2 = 99.9$; then we need to change the mantissa to 100 and add 1 to the exponent in order to get the next number, $0.100 \cdot 10^3 = 100$. Therefore, in the interval $[10,100]$, the numbers are still evenly spaced, but the spacing is now equal to 0.1 instead of 0.01.



Consequently, the entire structure of the floating point numbers in $\mathscr{F}$ from 1 to 10 is repeated but expanded by a factor 10 in the interval $[10,100]$, and the same effect can be observed in the next interval $[100,1000]$. It can also be easily seen that the same thing happens for the numbers that are smaller than 1: the interval $[0.1,1]$ is a shrunk copy of the interval $[1,10]$, and so on.

More generally, the set $\mathscr{F}$ has the following structure: let us take $p \in \mathbb{N}$ such that $L < p < U$. Then $\beta^p, \beta^{p+1} \in \mathscr{F}$ and

- intervals between two consecutive powers of the base, $[\beta^p, \beta^{p+1}]$, are called segments;
- the number coming in $\mathscr{F}$ right after $\beta^p = (-1)^0\,(0.10\ldots00)_\beta \cdot \beta^{p+1}$ is $(-1)^0\,(0.10\ldots01)_\beta \cdot \beta^{p+1}$. Therefore, the distance between β^p and the successive number is

$$(-1)^0\,(0.10\ldots01)\,\beta^{p+1} - (-1)^0\,(0.10\ldots0)\,\beta^{p+1} = \beta^{p+1-t}; \tag{1.1}$$

- in each segment, the representable numbers are evenly spaced, and every segment contains the same quantity of numbers equal to $\beta^t - 2$ (the extrema of each segment are not counted here);
- from $[\beta^p, \beta^{p+1}]$ to $[\beta^{p+1}, \beta^{p+2}]$, the numbers are obtained by expanding everything by a factor β. When the exponent attains U, it is called overflow;
- from $[\beta^p, \beta^{p+1}]$ to $[\beta^{p-1}, \beta^p]$, the numbers are obtained by shrinking everything by a factor β. When the exponent attains L, it is called underflow;

Remark 1.1. *In $[\beta^p, \beta^{p+1}]$, the spacing is equal to β^{p+1-t}. Thus the distance between two consecutive points varies according to their value.*

Example 1.11. *Consider $\beta = 10$ and $t = 4$. Then:*

- *in $[1,10]$, $p = 0$ and the spacing is equal to $\beta^{1-t} = 10^{-3}$;*
- *in $[10,100]$, $p = 1$ and the spacing is equal to $\beta^{2-t} = 10^{-2}$;*
- *in $[100,1000]$, $p = 2$ and the spacing is equal to $\beta^{3-t} = 10^{-1}$.*

Remark 1.2. *Consider $\mathscr{F}(2,53,-1021,1024)$, as in double-precision computers. For some value of p, the spacing is equal to 1. Indeed, $\beta^{p+1-t} = 1$ whenever $p = t-1$. Thus $[2^{52}, 2^{53}]$ is the segment where only the integers are represented.*
This can be checked in MATLAB as follows. Let us consider the interval $[2^{52}, 2^{52}+2]$. Since we expect the computer to see only integers in this interval, we expect to only obtain the integers 2^{52}, $2^{52}+1$ and $2^{52}+2$ (possibly repeated), even if we try to divide the interval into 10 evenly spaced parts. Let us see if it is true:

```
>> x = linspace(2^52, 2^52+2, 10)'
    x =
        4503599627370496
        4503599627370496
        4503599627370496
        4503599627370497
        4503599627370497
        4503599627370497
        4503599627370497
        4503599627370498
        4503599627370498
        4503599627370498
```

It indeed seems to be true, but to be sure that this is not only an effect of the format in which numbers are displayed in MATLAB (here the format is `rat`*), we compute the difference between two successive entries of x:*

```
>> diff(x)'
    ans =
        0       0       1       0       0       0       1       0       0
```

This confirms the fact that there are only the integers in $[2^{52}, 2^{53}]$. Similarly, it can be seen that in the segment $[2^{53}, 2^{54}]$, there are only even numbers, and so forth.

Every real number $x \in \mathbb{R}$ can be represented with an infinite mantissa:

$$x = (-1)^s (0.\alpha_1 \ldots \alpha_t \alpha_{t+1} \ldots)_\beta \cdot \beta^e.$$

Let $\mathrm{fl}(x)$ (float of x) be the number in $\mathscr{F}$ that has a minimal distance to x, that is we define the function fl as:

$$\begin{aligned} \mathrm{fl} : \mathbb{R} &\to \mathscr{F} \\ x &\mapsto \mathrm{fl}(x) = (-1)^s\,(0.\alpha_1 \ldots \tilde{\alpha}_t)_\beta \cdot \beta^e, \end{aligned}$$

where $\tilde{\alpha}_t$ has to be chosen between 0 and $\beta - 1$. This type of approximation is called round-off. Since $\mathscr{F}$ is symmetric with respect to the origin, let us only consider the case in which $s = 0$, without loss of generality. Since $\alpha_1 \neq 0$, then $x, \mathrm{fl}(x) \in \left[\beta^{e-1}, \beta^e\right]$.

Remark 1.3. *In a computer, when one writes for example* `x=0.3`*, what is actually encoded in the variable* `x` *is the round-off approximation of* 0.3*, that is* $\underline{fl}(0.3)$.

Let us estimate the error between x and its floating-point approximation. From equation (1.1), by taking $p = e - 1$, the spacing in the segment $\left[\beta^{e-1}, \beta^e\right]$ is equal to β^{e-t}. We can then compute the absolute and relative errors between x and $\mathrm{fl}(x)$.

Absolute error

$$|x - \mathrm{fl}\,(x)| \leq \frac{1}{2} \cdot \text{spacing} \leq \frac{1}{2}\beta^{e-t}.$$

Relative error

$$\begin{aligned} \frac{|x - \mathrm{fl}\,(x)|}{|x|} &\leq \frac{1}{2}\beta^{e-t} \cdot \beta^{1-e} \qquad \text{since } \frac{1}{|x|} \leq \beta^{1-e} \\ &\leq \frac{1}{2}\beta^{1-t}. \end{aligned}$$

Note that the absolute error increases when x becomes large (that is, when the exponent e increases), whereas the relative error does not depend on the interval in which x is located. It actually does not even depend on x itself.

Definition 1.2 (Round-off unit)**.**

$$u := \frac{1}{2}\beta^{1-t}$$

is called the round-off unit. In $\mathscr{F}(2, 53, -1021, 1024)$*, that is in double-precision modern computers, we have* $u = 2^{-53} \approx 1.11 \cdot 10^{-16}$.

Errors made when performing arithmetic operations will be measured with respect to u.

Theorem 1.2. *Let* $x \in \mathbb{R}$*. There exists a* $\underline{fl}(x) \in \mathscr{F}$ *such as :*

$$\underline{fl}(x) = x\,(1 + \epsilon_x), \qquad \textit{with } |\epsilon_x| \leq u.$$

ϵ_x *depends on* x *but is still always bounded by* u*.*

Proof. Let ϵ_x be the signed relative error, that is

$$\epsilon_x = \frac{\mathrm{fl}(x) - x}{x}. \tag{1.2}$$

Then, $|\epsilon_x| \leq u$, and if we multiply (1.2) by x and simplify, we obtain $\mathrm{fl}(x) = x(1+\epsilon_x)$. □

1.3 Operations in $\mathscr{F}$ and round-off errors

First of all, let us remark that $\mathscr{F}$ is not a field with respect to addition and multiplication. In particular, this means that the sum of two numbers in $\mathscr{F}$ may not belong to $\mathscr{F}$ anymore. As a consequence, given two real numbers $x, y \in \mathscr{F}$, the approximation of their sum/subtraction is $\mathrm{fl}(\mathrm{fl}(x) \pm \mathrm{fl}(y))$ as, in general, $\mathrm{fl}(x) \pm \mathrm{fl}(y)$ does not belong to $\mathscr{F}$.

Example 1.12. *Consider $\mathscr{F}(10,3,L,U)$. Then $x := 0.123 \cdot 10^0$ and $y := 0.456 \cdot 10^3$ belong to $\mathscr{F}$, but*

$$x + y = 456.123 = \underbrace{0.456123 \cdot 10^3}_{>3 \textit{ significant digits}} \notin \mathscr{F}.$$

The round-off approximation of $x+y$ in $\mathscr{F}(10,3,L,U)$ is $\underline{fl}(x+y) = 0.456 \cdot 10^4 \in \mathscr{F}$.

Therefore, more generally, the computer does the following:

$$\begin{cases} x + y \to \mathrm{fl}\left(\mathrm{fl}(x) + \mathrm{fl}\left(y\right)\right) = \left(x(1+\epsilon_x) + y\left(1+\epsilon_y\right)\right)(1+\epsilon_s), \\ x \cdot y \to \mathrm{fl}\left(\mathrm{fl}(x) \cdot \mathrm{fl}\left(y\right)\right) = \left(x(1+\epsilon_x) \cdot y\left(1+\epsilon_y\right)\right)\left(1+\epsilon_p\right), \end{cases}$$

where $\epsilon_x, \epsilon_y, \epsilon_s$ and ϵ_p are respectively the relative errors in the approximation of x, y, the sum and the product as in Theorem 1.2. The only common operation for which $\mathscr{F}$ is closed is the change of sign, that is $x \in \mathscr{F}$ if and only if $-x \in \mathscr{F}$.

We are interested to know if and when the rounding of numbers can generate large errors. This depends on the so-called conditioning of a problem.

Definition 1.3 (Stability). *Let $G(x) = y$ be a problem to solve. We say that the problem is stable or well conditioned if a small disturbance δx of the data x implies a small disturbance δy of the result y. Mathematically, it means that if there is $\eta = \eta(x) > 0$ (independent from y) such that $\|\delta x\| \leq \eta$ and $G(x+\delta x) = y + \delta y$, then there exists $\epsilon = \epsilon(x) > 0$ (independent from y) such that $\|\delta y\| \leq \epsilon \|\delta x\|$.*

Definition 1.4 (Absolute condition number). *The absolute condition number of problem G given the data x is the quantity*

$$K_{abs}(x) := \sup_{\delta x : \|\delta x\| \neq 0} \frac{\|\delta y\|}{\|\delta x\|}.$$

Definition 1.5 (Relative condition number). *The relative condition number of problem G given the data x is the quantity*

$$K_{rel}(x) := \sup_{\delta x : \|\delta x\| \neq 0} \frac{\|\delta y\| / \|y\|}{\|\delta x\| / \|x\|}.$$

If K_{rel} is small, the problem is well conditioned, or stable. The problem is ill-conditioned if K_{rel} is large. If we round x by $\mathrm{fl}(x)$, the resulting error made on y can be estimated thanks to the condition number: in this case, the disturbance δx of the data x corresponds to the error made by round-off, and thanks to Theorem 1.2,

$$\delta x = \mathrm{fl}(x) - x = x(1+\epsilon_x) - x = x\epsilon_x, \qquad \text{with } |\epsilon_x| \le u.$$

Then, as previously seen, the relative error in the data is bounded by the round-off unit:

$$\frac{\|\delta x\|}{\|x\|} = \frac{\|x\epsilon_x\|}{\|x\|} = |\epsilon_x| \le u,$$

and consequently, when we have $y = G(x)$ and $y + \delta y = \mathrm{fl}\Big(G\big(\mathrm{fl}(x)\big)\Big)$, then

$$\begin{aligned}\frac{\|\delta y\|}{\|y\|} &= \left(\frac{\|\delta y\|}{\|y\|}\frac{\|x\|}{\|\delta x\|}\right)\frac{\|\delta x\|}{\|x\|}\\ &\le K_{\mathrm{rel}}(x)u.\end{aligned}$$

We can apply this very same reasoning to the evaluation of the round-off error made when computing a sum on a computer:

$$y = G(\mathbf{x}) = x_1 + x_2, \quad y + \delta y = \mathrm{fl}\Big(G\big(\mathrm{fl}(\mathbf{x})\big)\Big) = \mathrm{fl}\big(\mathrm{fl}(x_1) + \mathrm{fl}(x_2)\big).$$

Let us look at the relative error on the sum y, thanks to Theorem 1.2:

$$\begin{aligned}\frac{\|\delta y\|}{\|y\|} &= \frac{\big|\mathrm{fl}\big(\mathrm{fl}(x_1)+\mathrm{fl}(x_2)\big) - (x_1 - x_2)\big|}{|y|}\\ &= \frac{|(x_1(1+\epsilon_1) + x_2(1+\epsilon_2))(1+\epsilon_s) - (x_1+x_2)|}{|x_1+x_2|}\\ &= \frac{|x_1\epsilon_1 + x_2\epsilon_2 + (x_1(1+\epsilon_1) + x_2(1+\epsilon_2))\epsilon_s|}{|x_1+x_2|},\end{aligned}$$

where ϵ_1, ϵ_2 and ϵ_s are respectively the relative errors in the approximation of x_1, x_2 and the sum. However, $|\epsilon_1|$, $|\epsilon_2|$, $|\epsilon_s| \le u$ and $u = 2^{-53}$. Thus $|\epsilon_1\epsilon_s|$, $|\epsilon_2\epsilon_s| \le u^2$ are negligible compared to the other terms. Consequently,

$$\begin{aligned}\frac{\|\delta y\|}{\|y\|} &\approx \frac{|x_1\epsilon_1 + x_2\epsilon_2 + (x_1+x_2)\epsilon_s|}{|x_1+x_2|}\\ &= \left|\frac{x_1\epsilon_1 + x_2\epsilon_2}{x_1+x_2} + 1\right|\\ &\le \left(\frac{|x_1|}{|x_1+x_2|} + \frac{|x_2|}{|x_1+x_2|} + 1\right)u = K_{\mathrm{rel}}(x)u.\end{aligned}$$

So since u is a constant, if x_1 and x_2 have opposite signs, $|x_1 + x_2|$ is very small, so the value of K_{rel} explodes: the problem is ill-conditioned. This is in the case of a subtraction of two very close numbers. The resulting error can be really big, such operations therefore have to be avoided to obtain accurate solutions. Instead, if x_1 and x_2 have the same sign, $|x_1 + x_2|$ remains of the same order of magnitude as x_1 and x_2; the problem is well-conditioned.

The same reasoning can be performed to check the stability of other operations such as the multiplication and the division.

Remark 1.4. *Often, there are many different ways to do a computation. The conditioning number of a problem may depend on this choice.*
For example, imagine that one wants to compute $\frac{1}{x(1+x)}$ using a computer, with x large. Take for instance $x = 10^6$. This operation can be performed in at least two different ways:

$$\begin{tikzcd}
 & x & & \\
1)\ x \arrow[ur] \arrow[dr] & & x(1+x) \arrow[r] & \frac{1}{x(1+x)} \text{ is stable;} \\
 & 1+x \arrow[ur] & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & \frac{1}{x} \arrow[r] & 1+x \arrow[dr] & \\
2)\ x \arrow[ur] \arrow[dr] & & & \frac{1}{x} - \frac{1}{(1+x)} \text{ is unstable.} \\
 & 1+x \arrow[r] & \frac{1}{1+x} \arrow[ur] &
\end{tikzcd}$$

In the general case of more complex problems of the type $G(x) = y$ that involve many different operations, the following general rule applies:

1) If the problem to solve is ill-conditioned, the numerical method is affected.

2) If the numerical method itself is ill-conditioned (wrong order of the operations for example), errors are generated.

Chapter 2

Numerical integration

The goal of this section is to construct algorithms in order to evaluate definite integrals, i.e.

$$\int_a^b f(x)\,\mathrm{d}x.$$

In undergraduate courses of Calculus it is common to learn how calculate very simple integrals like

$$\int_0^1 e^x\,\mathrm{d}x \quad \text{or} \quad \int_0^\pi \cos(x)\,\mathrm{d}x.$$

by the use of a table of integrals containing the most common closed-form primitives. However, if, instead, we try to compute the slightly more complicate expressions

$$\int_0^1 e^{x^2}\,\mathrm{d}x \quad \text{or} \quad \int_0^\pi \cos(x^2)\,\mathrm{d}x,$$

we realize that this approach becomes quickly infeasible. Indeed, we know that it is not always possible to find a closed form for a primitive. Even when we know it, it may be difficult to use it! This is the case of $f(x) = \cos(4x)\cos(3\sin(x))$ for which

$$\int_0^\pi f(x)\,\mathrm{d}x = \pi\left(\frac{3}{2}\right)^4 \sum_{k=0}^{\infty} \frac{\left(-\frac{9}{4}\right)^k}{k!\,(k+4)!}.$$

Evaluating very complicated definite integrals is a common problem arising in many areas of science and engineering. In this regard, the practitioner should have at his disposal some ad hoc strategies relying on the use of a computer.

The goal of this chapter is to introduce the reader to some well-known numerical methods that approximate definite integrals. In the following, unless otherwise stated, we are going to assume that $f \in C^0([a,b])$.

2.1 The Newton-Cotes composite formulae

First of all, let us subdivide $[a, b]$ using $n+1$ equally spaced points $x_i = a+iH$, for $i = 0,\dots,n$, where $H = \frac{b-a}{n}$. To fix ideas, an example is provided in Figure 2.1.

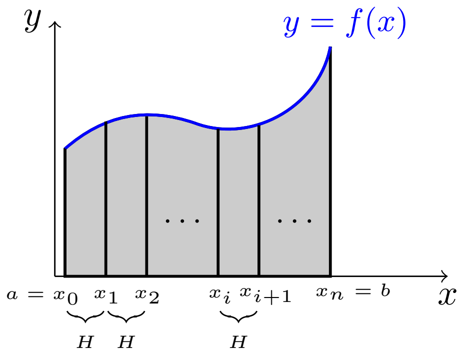


Figure 2.1: Representation of a numerical method for approximating the integral of $f(x)$.

Using the additivity of the Lebesgue integral (we refer again to Figure 2.1) we can write

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x.$$

Then, by employing a change of variables

$$\sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x = \sum_{i=0}^{n-1} H \int_0^1 f(x_i + tH)\,\mathrm{d}t.$$

By finally denoting $g(t) := f(x_i+tH)$, we observe that we have reduced our task to the computation of definite integrals between 0 and 1. So, the next section will be devoted to the construction of algorithms for the computation of

$$\int_0^1 g(t)\,\mathrm{d}t.$$

2.1.1 Newton-Cotes integration formulae on $[0,1]$

Since polynomials are easy to integrate, the idea is to approximate the function g by polynomials interpolating g at some specific points in $[0,1]$. We will come back to the problem of interpolation in the next chapter.

In the following we will denote as $\mathbb{P}_s$ the vector space of polynomials with real coefficients and having degree s. We will start introducing some classical formulas, where the function g is approximated by polynomials of low degree. Then, we will introduce the general Newton-Cotes integration formula, which approximates the function g with a polynomial of arbitrary degree.

1. **Midpoint formula**
   Let us approximate $\int_0^1 g(t)\,\mathrm{d}t$ by exactly integrating $p_0 \in \mathbb{P}_0$, the polynomial of degree 0

passing through the point $\left(\frac{1}{2}, g\left(\frac{1}{2}\right)\right)$, see for instance Figure 2.2. We denote by $Q_1^{nc}(g)$ such an integral, i.e.

$$\int_0^1 g(t)\,\mathrm{d}t \approx Q_1^{nc}(g) := \int_0^1 p_0(t)\,\mathrm{d}t = g\left(\frac{1}{2}\right).$$

Note that the index 1 in Q_1^{nc} stands for the number of points in which the function g is evaluated and nc stands for "Newton-Cotes".

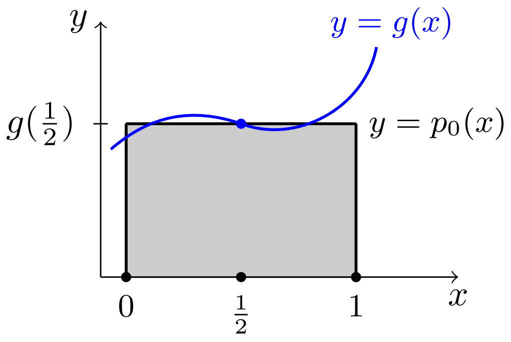


Figure 2.2: Midpoint quadrature rule.

The point $\frac{1}{2}$ is the node of the midpoint formula, its weight is 1.

2. **Trapezoidal formula**

This time let us approximate $\int_0^1 g(t)\,\mathrm{d}t$ by integrating $p_1 \in \mathbb{P}_1$, the polynomial of degree 1 passing through the points $(0, g(0))$ and $(1, g(1))$. An example is depicted in Figure 2.3. Let us compute the expression for $p_1(t) = at + b$, where $a, b \in \mathbb{R}$ need to be determined. By imposing $p_1(0) = g(0)$ and $p_1(1) = g(1)$, we get $p_1(t) = \left(g(1) - g(0)\right) t + g(0)$. Hence

$$\int_0^1 g(t)\,\mathrm{d}t \approx Q_2^{nc}(g) := \int_0^1 p_1(t)\,\mathrm{d}t = \frac{1}{2}\left(g(0) + g(1)\right).$$

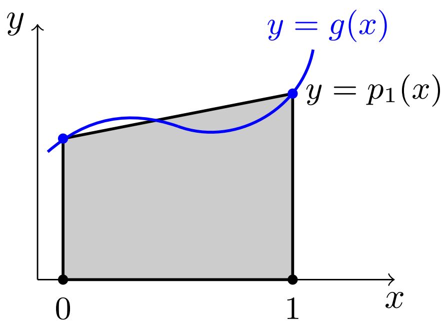


Figure 2.3: Trapezoidal quadrature rule.

The nodes of the trapezoidal formula are $0, 1$, the weights are $\frac{1}{2}, \frac{1}{2}$.

3. **Simpson formula**

   We approximate the integral $\int_0^1 g(t)\mathrm{d}t$ by the integral of $p_2 \in \mathbb{P}_2$, the polynomial of degree 2 passing through the points $\left(0, g(0)\right)$, $\left(\frac{1}{2}, g\left(\frac{1}{2}\right)\right)$ and $\left(1, g(1)\right)$, where an example can be found in Figure 2.4. Again by imposing $p_2(0) = g(0)$, $p_2\left(\frac{1}{2}\right) = g\left(\frac{1}{2}\right)$, $p_2(1) = g(1)$ to $p_2(t) = at^2 + bt + c$, $a, b, c \in \mathbb{R}$, we obtain $p_2(t) = \left(2g(0) - 4g\left(\frac{1}{2}\right) + 2g(1)\right) t^2 + \left(-3g(0) + 4g\left(\frac{1}{2}\right) - g(1)\right) t + g(0)$. A simple computation yields:

$$\int_0^1 g(t)\,\mathrm{d}t \approx Q_3^{nc} := \int_0^1 p_2(t)\,\mathrm{d}t = \frac{1}{6}\left(g(0) + 4g\left(\frac{1}{2}\right) + g(1)\right).$$

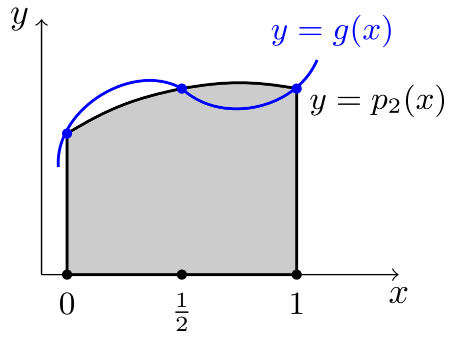


Figure 2.4: Simpson quadrature rule.

Here the nodes are $0, \frac{1}{2}, 1$ and the weights are $\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$.

The previous formulae can be generalized by interpolating with polynomials of higher degree, that require the evaluation of the function at more nodes. For equidistant nodes, these are called Newton-Cotes formulae. We can therefore generalize the previous quadrature rules to the following.

4. **Newton-Cotes formulae**

   We define the Newton-Cotes formula with S nodes as:

$$\int_0^1 g(t)\,\mathrm{d}t \approx Q_S^{nc}(g) := \int_0^1 p_S(t)\,\mathrm{d}t,$$

with $p_S \in \mathbb{P}_{S-1}$, the polynomial of degree $S-1$ passing through the points $\left(c_i, g(c_i)\right)$, $i = 1, \ldots, S$, where $c_i = (i-1)\,h$, $h > 0$, $i = 1, \ldots, S$ are the equidistant nodes of the Newton-Cotes quadrature formula. By imposing the interpolation conditions $p(c_i) = g(c_i)$, for $i = 1, \ldots, S$, the integral $\int_0^1 p_S(t)\,\mathrm{d}t$ can be expressed in terms of the quadrature weights b_i as

$$\int_0^1 p_S(t)\,\mathrm{d}t = \sum_{i=1}^{S} g(c_i) b_i.$$

The Newton-Cotes formulae are quadrature rules, used to approximate the integral $\int_0^1 g(t)\,\mathrm{d}t$, which can be written as

$$\int_0^1 g(t)\,\mathrm{d}t \approx Q_S(g) := \sum_{i=1}^{S} g(c_i) b_i, \tag{2.1}$$

where b_i are the quadrature weights and c_i the quadrature nodes. While the Newton-Cotes formulae make use of equispaced nodes, in general this is not a requirement for quadrature rules of the form (2.1).

Let us provide some basic definitions and properties for this type of quadrature formulae.

Definition 2.1 (Order of a quadrature formula)**.** *The quadrature formula Q_S, described by the S nodes $\{c_i\}_{i=1}^S$ and S weights $\{b_i\}_{i=1}^S$, is said to be of order r if it is exact for any polynomial p of degree $\le r-1$, that is, if*

$$Q_S(p) = \sum_{i=1}^{S} p(c_i) b_i, \quad \forall p \in \mathbb{P}_{r-1}.$$

Remark 2.1. *By construction, the Newton-Cotes formulae Q_S^{nc} are of order S.*

Once S nodes are fixed, as, e.g., equidistant points, we wish to compute the weights guaranteeing that the corresponding formula is of order $S+1$. The next theorem tells us how to do so.

Theorem 2.1. *Given S distinct nodes $\{c_i\}_{i=1}^S$, the quadrature formula Q_S, constructed with $(\{c_i\},\{b_i\})_{i=1}^S$, is of order p if and only if the weights verify:*

$$\sum_{i=1}^{S} c_i^{q-1} b_i = \frac{1}{q}, \qquad \forall q = 1,\dots,p. \tag{2.2}$$

Proof. If the formula is of order p, then it integrates exactly the monomials $1, t, \dots, t^{p-1}$, i.e.

$$\frac{1}{q} = \int_0^1 t^{q-1}\,\mathrm{d}t = \sum_{i=1}^{S} c_i^{q-1} b_i \qquad \forall\ q = 1,\dots,p. \tag{2.3}$$

Conversely, if the weights verify (2.2), then, by construction, the quadrature rule is exact for $1, t, \dots, t^{p-1}$. As $1, t, \dots, t^{p-1}$ forms a basis for $\mathbb{P}_{p-1}$, by linearity the formula is exact for all elements of $\mathbb{P}_{p-1}$. □

The set of equations(2.3) can be rewritten in matrix form as follows.

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ c_1 & c_2 & & c_S \\ c_1^2 & c_2^2 & & c_S^2 \\ \vdots & & \ddots & \\ c_1^{p-1} & & & c_S^{p-1} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{S-1} \\ b_S \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \vdots \\ \frac{1}{p-1} \\ \frac{1}{p} \end{pmatrix}. \tag{2.4}$$

A matrix of this form is called Vandermonde matrix. When $p = S$, it is a square matrix. In this case, it can be proven that the Vandermonde matrix is invertible as soon as the nodes $c_1,\dots,c_S$ are distinct. Thus, the weights $\{b_i\}_{i=1}^S$ are fixed and given by the solution of the linear system (2.4). This fact is summarized in the following Lemma whose proof is omitted.

Lemma 2.2. *If $c_1, \dots, c_S$ are distinct, then there exists a unique quadrature formula Q_S of order S which has $\{c_i\}_{i=1}^S$ as nodes.*

For instance, in the case of the Simpson formula, we had

$$c_1 = 0, \qquad c_2 = \frac{1}{2}, \qquad c_3 = 1$$
$$b_1 = \frac{1}{6}, \qquad b_2 = \frac{2}{3}, \qquad b_3 = \frac{1}{6}.$$

Indeed, these points and weights verify the linear system:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & \frac{1}{2} & 1 \\ 0 & \frac{1}{4} & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{6} \\ \frac{2}{3} \\ \frac{1}{6} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{pmatrix}$$

which proves that the formula has order 3. Now, we are interested to know whether the order of the Simpson rule could be higher than 4. In order for the formula to be of order 4, it should also verify:

$$b_1 c_1^3 + b_2 c_2^3 + b_3 c_3^3 \overset{?}{=} \frac{1}{4}.$$

By replacing the c_i and b_i with the values for the Simpson formula, we get

$$\frac{1}{6}0 + \frac{2}{3}\frac{1}{8} + \frac{1}{6}1 = \frac{1}{4},$$

where the equation is clearly satisfied. However, it is not of order 5, since

$$b_1 c_1^4 + b_2 c_2^4 + b_3 c_3^4 \neq \frac{1}{5}.$$

Remark 2.2. *The quadrature weights and corresponding orders of convergence for different quadrature formulae Q_S, $S = 2, 3, 4, 5, 6, 7$ can be found in Table 2.1.*

Remark 2.3. *As the reader can see from Figure 2.5, the weights "explode" for $S > 10$. This means that in order to improve the precision of our quadrature formula, we better refine the subdivision of the integration interval rather than increase S.*

2.1.2 Composite Newton-Cotes integration formulae on $[a, b]$

So far we have been developing quadrature rules in the unit interval $[0, 1]$. However, our original goal was to approximate $\int_a^b f(t)\,\mathrm{d}t$. As we have already seen at the beginning of Section 2.1, by subdividing $[a, b]$ into n subintervals of length H and employing a change of variable, we obtain

$$\int_a^b f(x)\,\mathrm{d}x = H \sum_{i=0}^{n-1} \int_0^1 f(x_i + tH)\,\mathrm{d}t.$$

S	order	weights b_i
2	2	$\frac{1}{2}\ \frac{1}{2}$
3	4	$\frac{1}{6}\ \frac{4}{6}\ \frac{1}{6}$
4	4	$\frac{1}{8}\ \frac{3}{8}\ \frac{3}{8}\ \frac{1}{8}$
5	6	$\frac{7}{90}\ \frac{32}{90}\ \frac{12}{90}\ \frac{32}{90}\ \frac{7}{90}$
6	6	$\frac{19}{288}\ \frac{75}{288}\ \frac{50}{288}\ \frac{50}{288}\ \frac{75}{288}\ \frac{19}{288}$
7	8	$\frac{41}{840}\ \frac{216}{840}\ \frac{27}{840}\ \frac{272}{840}\ \frac{27}{840}\ \frac{216}{840}\ \frac{41}{840}$

Table 2.1: Quadrature weigths and orders of Newton-Cotes formulae for $S = 2,3,4,5,6,7$.

Hence, we are done if we apply for each $i = 0,\dots,n-1$ an integration formula on $[0,1]$.

Example 2.1. *For instance, let us employ the trapezoidal quadrature rule $Q_2^{nc}\left(f(x_i+tH)\right)$, for every $i = 0,\dots,n-1$, in order to approximate $\int_a^b f(x)\,\mathrm{d}x$. By doing so, we obtain the following global integration formula in $[a,b]$:*

$$Q_{n,2}^{c,nc}(f) = H\sum_{i=0}^{n-1} Q_2^{nc}\left(f(x_i+tH)\right) = H\sum_{i=0}^{n-1}\int_0^1 p_1^i(x_i+tH)\,\mathrm{d}t = \frac{H}{2}\sum_{i=0}^{n-1}\left(f(x_i)+f(x_{i+1})\right),$$

where $p_1^i \in \mathbb{P}_1$ passes through $(0,g(0))$ and $(1,g(1))$, with $g(t) := f(x_i+tH)$. Here the superscripts c and nc stand for "composite" and "Newton-Cotes", respectively, while the subscripts n and 2 denote the number of subintervals of equal lengths in which $[a,b]$ has been subdivided and the number of quadrature points employed in each subinterval.

In general, the formula we found by applying, in each subinterval, the respective local quadrature rule in $[0,1]$ is called composite quadrature formula.

2.2 Integration formulae of higher order

In the previous Section we have seen that for $c_1,\dots,c_S$ fixed and distinct there exists a unique quadrature formula Q_S, constructed using $(\{b_i\},\{c_i\})_{i=1}^S$, with order at least S, where the weights b_i are obtained by solving the linear system (2.4). In particular, we have seen that order S can be achieved by employing evenly spaced integration points, with the Newton-Cotes formulae.
In this Section we want to investigate if there exists a choice of the quadrature nodes that provides quadrature formulae of order p, $p > S$, and, moreover, we are going to study which would be the best possible quadrature nodes and how to obtain them.

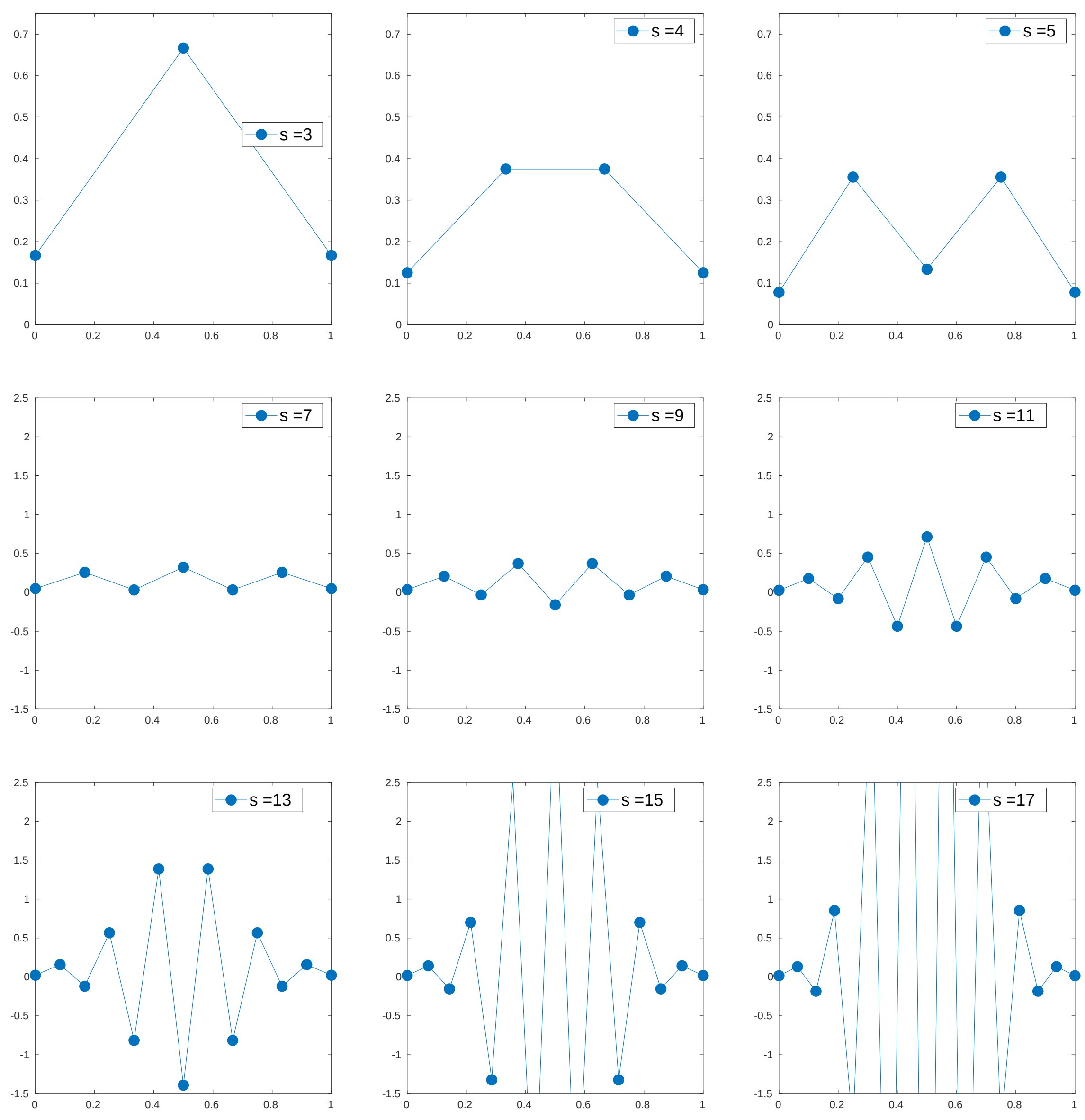


Figure 2.5: Weights of the Newton-Cotes formulae for different values of S.

Theorem 2.3. *Let Q_S be the quadrature rule constructed with $(\{b_i\},\{c_i\})$, $i = 1,\dots,S$ and*

$$M(t) = (t - c_1)(t - c_2)\dots(t - c_S).$$

The formula Q_S has order $p \geq S + m$ if and only if

$$\int_0^1 M(t)\,g(t)\;\mathrm{d}t = 0 \quad \forall\; g \in \mathbb{P}_{m-1}. \tag{2.5}$$

Proof. There are two implications to be proven.
Let us assume $\int_0^1 M(t)\,g(t)\;\mathrm{d}t = 0 \quad \forall\; g \in \mathbb{P}_{m-1}$. We want to prove that Q_S is of order $S + m$. Let

$f \in \mathbb{P}_{S+m-1}$. We consider the polynomial $r \in \mathbb{P}_{S-1}$ interpolating f at all nodes. We have $f - r \in \mathbb{P}_{S+m-1}$ and, moreover, $f - r$ vanishes in $c_1, \dots, c_S$. Thus it holds:

$$f - r = M g_f,$$

where $g_f \in \mathbb{P}_{m-1}$. (Indeed, the existence of such a g_f is guaranteed by the polynomial division). Now, we compute:

$$\int_0^1 f(t)\,\mathrm{d}t = \int_0^1 M(t)\,g_f(t)\,\mathrm{d}t + \int_0^1 r(t)\,\mathrm{d}t = \int_0^1 r(t)\,\mathrm{d}t,$$

as $\int_0^1 M(t) g_f(t)\,\mathrm{d}t = 0$ by (2.5). We can also compute $Q_S(f)$:

$$Q_S(f) = \sum_{i=1}^{S} b_i f(c_i) = \sum_{i=1}^{S} b_i M(c_i)\,g_f(c_i) + \sum_{i=1}^{S} b_i r(c_i) = \sum_{i=1}^{S} b_i r(c_i),$$

as, by construction, $M(c_i) = 0$, $i = 1, \dots, S$, and so the first term disappears. By using the exactness of $Q_S(r)$, namely $Q_S(r) = \int_0^1 r(t)\,\mathrm{d}t$, and the fact that r interpolates f at the nodes, it holds:

$$\int_0^1 f(t)\,\mathrm{d}t = \int_0^1 r(t)\,\mathrm{d}t = \sum_{i=1}^{S} b_i r(c_i) = \sum_{i=1}^{S} b_i f(c_i).$$

We now prove the other implication.
Let us assume that the formula has order $S + m$. We should prove that

$$\int_0^1 M(t)\,g(t)\,\mathrm{d}t = 0 \qquad \forall g \in \mathbb{P}_{m-1}.$$

Note that, since $Mg \in \mathbb{P}_{S+m-1}$ and the formula has order $S + m$, it holds:

$$\int_0^1 M(t)\,g(t)\,\mathrm{d}t = \sum_{i=1}^{S} b_i M(c_i)\,g(c_i) = 0.$$

□

Let us see in the next example how we can use the property (2.5) to construct the quadrature nodes $\{c_i\}$ in order to guarantee the desired order of the formula.

Example 2.2. *Let $S = 3$. For the sake of simplicity we rewrite*

$$M(t) = (t - c_1)(t - c_2)(t - c_3) = t^3 - \sigma_1 t^2 + \sigma_2 t - \sigma_3,$$

where

$$\begin{cases} \sigma_1 = c_1 + c_2 + c_3 \\ \sigma_2 = c_1 c_2 + c_1 c_3 + c_2 c_3 \\ \sigma_3 = c_1 c_2 c_3. \end{cases}$$

Let us start by setting $m = 1$ in Equation (2.5). We are looking for $\sigma_1, \sigma_2, \sigma_3$ such that

$$\int_0^1 M(t)\, \mathrm{d}t = \int_0^1 t^3 - \sigma_1 t^2 + \sigma_2 t - \sigma_3\, \mathrm{d}t = 0,$$

hence

$$0 = \int_0^1 M(t) = \frac{1}{4} - \frac{\sigma_1}{3} + \frac{\sigma_2}{2} - \sigma_3. \tag{2.6}$$

Now, let $m = 2$. This time we impose as well $\int_0^1 M(t)\, t\, \mathrm{d}t = 0$.

$$0 = \int_0^1 M(t)\, t\, \mathrm{d}t = \frac{1}{5} - \frac{\sigma_1}{4} + \frac{\sigma_2}{3} - \frac{\sigma_3}{2}. \tag{2.7}$$

For $m = 3$, we need to take into account another constraint: $\int_0^1 M(t)\, t^2\, \mathrm{d}t = 0$.

$$0 = \int_0^1 M(t)\, t^2\, \mathrm{d}t = \frac{1}{6} - \frac{\sigma_1}{5} + \frac{\sigma_2}{4} - \frac{\sigma_3}{3}. \tag{2.8}$$

Putting together (2.6),(2.7),(2.8), we finally get:

$$\sigma_1 = \frac{3}{2} \quad \sigma_2 = \frac{3}{5} \quad \sigma_3 = \frac{1}{20}.$$

As the reader might guess, the order of a formula cannot be arbitrarily high by just constructing nodes such that the orthogonality condition (2.5) holds for higher degree polynomials. In fact, the following result expresses this limitation.

Theorem 2.4. *Let p be the order of a quadrature formula Q_S with S nodes, then $p \leq 2S$.*

Proof. Assume, by contradiction, that there exists Q_S with order $p = 2S + 1$. Then by Theorem 2.3, we could choose $m = S + 1$ and then:

$$\int_0^1 M(t)\, g(t)\, \mathrm{d}t = 0 \quad \forall\, g \in \mathbb{P}_S.$$

The choice $g(t) = M(t)$ is allowed, and thus

$$\int_0^1 M^2(t)\, \mathrm{d}t = 0.$$

Since $M^2(t) \geq 0$ for every $t \in [0,1]$, this implies $M(t) = 0$ for every $t \in [0,1]$, which is a contradiction. $\square$

Remark 2.4. *As in Theorem 2.3 we have $p = S + m$, it follows that $m \leq S$.*

So, in order to construct a formula of degree of exactness $p = 2S$ we have to compute the nodes $c_1, \ldots, c_S$ such that

$$\int_0^1 M(t)\, g(t)\, \mathrm{d}t = 0 \quad \forall g \in \mathbb{P}_{S-1}.$$

In general this is not a trivial task. We will see how the theory of Legendre polynomials can come to help. However, before getting to Legendre polynomials, let us try to understand why we need the order p to be as large as possible.

2.3 Error analysis for numerical integration

Example 2.3. *In order to introduce the reader to the error analysis of integration formulae, let us start with a numerical experience. We consider a function f defined in the interval $[a,b]$, which has been subdivided into n subintervals of length $h = (b-a)/n$. For each of them, let us apply a composite Newton-Cotes quadrature formula $Q_{n,S}^{c,nc}$ and consider the error (at this stage, in an informal notation)*

$$\mathrm{err} := \int_a^b f(x)\,\mathrm{d}x - \sum_{j=0}^{n-1} h \sum_{i=1}^{S} b_i f\left(x_j + c_i h\right)$$

as a function of the <u>*number of evaluations of the function f*</u>*, which we denote by* fe*. In the case of Newton-Cotes rules, it is readily seen that* $\mathrm{fe} = n$ *for* $S = 1$ *and* $\mathrm{fe} = n(S-1)+1$ *for* $S > 1$*. Note that* fe *represents a measure of the computational time for a computer programmed to numerically approximate definite integrals. In Figure 2.6 we see the results, in a logarithmic scale, obtained for the Newton-Cotes formulae applied to:*

$$\int_0^3 e^{\cos(x)}\,\mathrm{d}x.$$

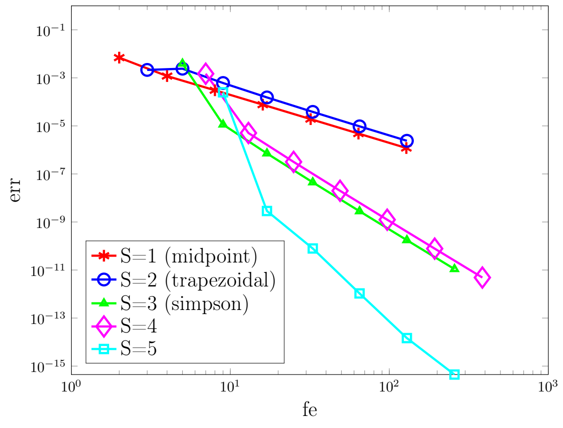


Figure 2.6: Error with respect to fe for Newton-Cotes formulae.

We clearly notice that log(err) *linearly depends on* log(fe)*. In particular, the slope of each straight line is given by the order p of the respective quadrature rule. This means that, with the same computational time, an integration formula with higher order gives a better accuracy and so is preferable!*

To understand the results of Example 2.3 we need to analyse the error we make when approximating a definite integral by a quadrature formula. Let us consider a general partition $\left(x_j\right)_{j=0}^{n}$ of the interval $[a,b]$ with $h_j := x_j - x_{j-1}$, for all $1 \le j \le n$, and $h := \max_{1\le j\le n} h_j$. We construct a general composite quadrature formula of order p, denoted as $Q_h^c(f)$, which approximates the definite integral $\int_a^b f(x)\,\mathrm{d}x$, and we are interested in evaluating the error, given by the expression

$$E_h(f) := \int_a^b f(x)\,\mathrm{d}x - Q_h^c(f),$$

and how this value depends on the order of the formula. We have

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{j=0}^{n-1}\int_{x_j}^{x_{j+1}} f(x)\,\mathrm{d}x = \sum_{j=0}^{n-1} h_j \int_0^1 f\left(x_j + h_j t\right)\mathrm{d}t.$$

On the other hand

$$Q_h^c(f) = \sum_{j=1}^{n} h_j \sum_{i=1}^{S} b_i f\left(x_j + h_j c_i\right),$$

i.e. $\int_0^1 f(x_j + h_j t)\,\mathrm{d}t$ is approximated by $Q_S\left(f\left(x_j + h_j t\right)\right)$, the local quadrature rule with S nodes. Notice that $S \le p \le 2S$. Then

$$\begin{aligned} E_h(f) &= \int_a^b f(x)\,\mathrm{d}x - \sum_{j=0}^{n-1} h_j \sum_{i=1}^{S} b_i f\left(x_j + h_j c_i\right) \\ &= \sum_{j=0}^{n-1} h_j \left(\int_0^1 f\left(x_j + h_j t\right)\mathrm{d}t - \sum_{i=1}^{S} b_i f\left(x_j + h_j c_i\right)\right). \end{aligned}$$

We define, for each $j = 0,\dots,n-1$ the local error:

$$E_h^j(f) := \int_0^1 f\left(x_j + h_j t\right)\mathrm{d}t - \sum_{i=1}^{S} b_i f\left(x_j + h_j c_i\right),$$

so that we can write

$$E_h(f) = \sum_{j=0}^{n-1} h_j E_h^j(f).$$

Let us assume that f is p-times continuously differentiable in $[a,b]$ so that it can be developed via Taylor expansion with Lagrange remainder around each x_j:

$$f(x) = \sum_{k=0}^{p-1} \frac{f^{(k)}(x_j)}{k!}\left(x - x_j\right)^k + \frac{f^{(p)}\left(\xi_j^x\right)}{p!}\left(x - x_j\right)^p, \quad \text{for some } \xi_j^x \in \left(x_j, x\right), \text{ and for } x \in \left(x_j, x_{j+1}\right).$$

Hence

$$\begin{aligned} f(x_j + t h_j) &= \sum_{k=0}^{p-1} \frac{f^{(k)}(x_j)}{k!} t^k h_j^k + \frac{f^{(p)}\left(\xi_j^1\right)}{p!} t^p h_j^p, \\ f(x_j + c_i h_j) &= \sum_{k=0}^{p-1} \frac{f^{(k)}(x_j)}{k!} c_i^k h_j^k + \frac{f^{(p)}\left(\xi_j^2\right)}{p!} c_i^p h_j^p. \end{aligned}$$

For each $j = 0, \dots, n-1$, let us estimate

$$\begin{aligned} E_h^j(f) &= \sum_{k=0}^{p-1} \frac{f^{(k)}(x_j)}{k!} h_j^k \left(\int_0^1 t^k \, \mathrm{d}t - \sum_{i=1}^S b_i c_i^k \right) + \frac{f^{(p)}(\xi_j^1)}{p!} h_j^p \int_0^1 t^p \, \mathrm{d}t - \frac{f^{(p)}(\xi_j^2)}{p!} h_j^p \sum_{i=1}^S b_i c_i^p \\ &= \sum_{k=0}^{p-1} \frac{f^{(k)}(x_j)}{k!} h_j^k \left(\frac{1}{k+1} - \sum_{i=1}^S b_i c_i^k \right) + \frac{f^{(p)}(\xi_j^1)}{p!} h_j^p \frac{1}{p+1} - \frac{f^{(p)}(\xi_j^2)}{p!} h_j^p \sum_{i=1}^S b_i c_i^p . \end{aligned}$$

Since the quadrature formula is supposed to be of order p and $p \geq S$, by Theorem 2.1 we have $\frac{1}{k+1} - \sum_{i=1}^S b_i c_i^k = 0$, for every $k = 0, \dots, p-1$, and all the terms in the summation indexed by k vanish. Hence, we obtain

$$E_h^j(f) = \frac{h_j^p}{p!} \left(f^{(p)}(\xi_j^1) \frac{1}{p+1} - f^{(p)}(\xi_j^2) \sum_{i=1}^S b_i c_i^p \right),$$

and, in particular,

$$\left| E_h^j(f) \right| \leq \frac{h_j^p}{p!} \max_{x \in [x_j, x_{j+1}]} \left| f^{(p)}(x) \right| \left(\frac{1}{p+1} + \left| \sum_{i=1}^S b_i c_i^p \right| \right).$$

Moreover, we set

$$C_p := \frac{1}{p!} \left(\frac{1}{p+1} + \left| \sum_{i=1}^S b_i c_i^p \right| \right)$$

so that

$$\left| E_h^j(f) \right| = C_p \max_{x \in [x_j, x_{j+1}]} \left| f^{(p)}(x) \right| h_j^p .$$

Note that the constant C_p does not depend on the spacing between the integration nodes. By recalling that $h := \max_{1 \leq j \leq n} h_j$ and summing over $j = 0, \dots, n-1$, we obtain:

$$\begin{aligned} \left| E_h(f) \right| = \left| \sum_{j=0}^{n-1} h_j E_h^j(f) \right| &\leq C_p \sum_{j=0}^{n-1} \max_{x \in [x_j, x_{j+1}]} \left| f^{(p)}(x) \right| h_j^{p+1} \\ &\leq C_p \max_{x \in [a,b]} \left| f^{(p)}(x) \right| h^p \sum_{j=0}^{n-1} h_j = C_p h^p (b-a) \max_{x \in [a,b]} \left| f^{(p)}(x) \right| . \end{aligned}$$

In order to simplify our notation, we denote $C_I = C_p (b-a)$, so that

$$\left| E_h(f) \right| \leq C_I h^p \max_{x \in [a,b]} \left| f^{(p)}(x) \right| .$$

From this estimate we conclude that, for a function f sufficiently regular, the error converges to zero with order p, where p represents the order of the quadrature rule. This result can be related to the convergence plots depicted in Figure 2.6. Indeed, we highlight that the number of function evaluations fe is inversely proportional to the sub-intervals maximum length h, i.e. fe $\approx \frac{1}{h}$.

Remark 2.5. *Let us remark that, if the function f is not regular enough, we recommend to avoid the use of a high order formula to approximate the integral of f. Instead, it is common practice* *to employ a low order, composite quadrature rule with a small h. For instance, if $f \in C^2$ but $f \notin C^3([a,b])$, it is better to set $p = 2$ and choose h as small as possible.*

Remark 2.6. *If we choose, for each subinterval j, a quadrature formula Q_S with order p_j, then we obtain*

$$\left|E_h(f)\right| \le C_I \max_{0\le j\le n-1} h^{p_j} \max_{x\in[a,b]} \left|f^{(p_j)}(x)\right|.$$

2.4 Optimal choice of quadrature points: the Gauss formulae

Let us consider as integration interval $[0,1]$. In Theorem 2.3, Section 2.2, we have shown that for a given integer $S > 0$ there exists a quadrature rule Q_S, constructed with nodes and weights $(\{c_i\},\{b_i\})_{i=1}^S$, of order $r \ge S+m$ if and only if

$$\int_0^1 M(t)\,g(t)\,\mathrm{d}t = 0 \qquad \forall g \in \mathbb{P}_{m-1},$$

where $M(t) = (t-c_1)(t-c_2)\dots(t-c_S)$. At this stage we want to find a way to get the quadrature nodes guaranteeing the optimal order $r = 2S$. The associated quadrature rule Q_S will be called Gauss quadrature formula.
The construction of a special class of polynomials will be useful for our task. For the sake of simplicity the construction is done on $[-1,1]$.

2.4.1 Legendre polynomials

Definition 2.2 (Legendre polynomials)**.** *The k-th Legendre polynomial p_k is a polynomial of degree k, i.e. $p_k \in \mathbb{P}_k$, such that*

$$\int_{-1}^1 p_k(t)\,g(t)\,\mathrm{d}t = 0 \quad \forall\, g \in \mathbb{P}_{k-1}. \tag{2.9}$$

Remark 2.7. *Multiplying (2.9) by a constant $c \in \mathbb{R}$, we observe that p_k is uniquely defined up to a multiplicative constant.*

In particular it holds:

$$\int_{-1}^1 p_k(t)\,p_j(t)\,\mathrm{d}t = 0 \quad \forall\, j = 0,\dots,k. \tag{2.10}$$

This means that p_k and p_j are orthogonal with respect to the scalar product: $< f,g >:= \int_{-1}^1 fg\mathrm{d}t$, implying that $\{p_k\}_{k=0}^p$ form an orthogonal basis for the vector space $\mathbb{P}_p$.
The next result provides a direct construction of Legendre polynomials.

Theorem 2.5. *The polynomial p_k defined by*

$$p_k(x) = c_k \frac{d^k}{dx^k}\left[\left(x^2-1\right)^k\right] \tag{2.11}$$

is the k-th Legendre polynomial. Here, $c_k := \frac{1}{2^k k!}$ is a multiplicative constant allowing us to have $p_k(1) = 1$.

Proof. First, we note that $p_k \in \mathbb{P}_k$, since it is obtained by computing k derivatives of a $2k$ degree polynomial. Then, we need to show that p_k is orthogonal to all vectors of $\mathbb{P}_{k-1}$ with respect to the scalar product defined above, namely

$$\int_{-1}^{1} p_k(x)\, g(x)\ \mathrm{d}t = 0 \quad \forall\, g \in \mathbb{P}_{k-1}.$$

Let us proceed via integration by parts:

$$\begin{aligned} c_k \int_{-1}^{1} \frac{d^k}{dx^k}\left[\left(x^2-1\right)\right]^k g(x)\ \mathrm{d}x &= -\,c_k \int_{-1}^{1} \frac{d^{k-1}}{dx^{k-1}}\left[\left(x^2-1\right)^k\right] \frac{d}{dx} g(x)\ \mathrm{d}x + \left[\frac{d^{k-1}}{dx^{k-1}}\left(x^2-1\right)^k g\right]_{-1}^{1} \\ &= -\,c_k \int_{-1}^{1} \frac{d^{k-1}}{dx^{k-1}}\left[\left(x^2-1\right)^k\right] \frac{d}{dx} g(x)\ \mathrm{d}x. \end{aligned}$$

We integrate by parts again:

$$\begin{aligned} -c_k \int_{-1}^{1} \frac{d^{k-1}}{dx^{k-1}}\left[\left(x^2-1\right)^k\right] \frac{d}{dx} g(x)\ \mathrm{d}x &= c_k \int_{-1}^{1} \frac{d^{k-2}}{dx^{k-2}}\left[\left(x^2-1\right)^k\right] \frac{d^2}{dx} g(x)\ \mathrm{d}x - \left[\frac{d^{k-2}}{dx^{k-2}}\left(x^2-1\right)^k \frac{d}{dx} g\right]_{-1}^{1} \\ &= c_k \int_{-1}^{1} \frac{d^{k-2}}{dx^{k-2}}\left[\left(x^2-1\right)^k\right] \frac{d^2}{dx} g(x)\ \mathrm{d}x. \end{aligned}$$

Hence, integrating by parts k times, we obtain:

$$c_k \int_{-1}^{1} \frac{d^k}{dx^k}\left[\left(x^2-1\right)\right]^k g(x)\ \mathrm{d}x = (-1)^k \int_{-1}^{1} \left(x^2-1\right) \frac{d^k}{dx^k} g(x)\,\mathrm{d}x.$$

Since $g \in \mathbb{P}_{k-1}$ its k-th derivative vanishes, hence

$$c_k \int_{-1}^{1} \frac{d^k}{dx^k}\left[\left(x^2-1\right)\right]^k g(x)\ \mathrm{d}x = 0.$$

$\square$

There also exists another representation of the Legendre polynomials that is more handy, and describes them through a recursive formula.

Proposition 2.6 (Recurrence relation)**.** *The Legendre polynomials verify the following recursive relation:*

$$(k+1)\,p_{k+1}(x) = (2k+1)\,x p_k(x) - k p_{k-1}(x) \qquad \forall\ k \geq 1.$$

Proof. see exercise series. $\square$

Theorem 2.7. *The k roots of the k-th Legendre polynomial p_k are real, distinct and lie inside the open interval* $(-1,1)$.

Proof. We proceed by contradiction. Let $\tau_1, \ldots, \tau_r$ be the real roots of p_k inside $(-1,1)$, with $r < k$. We define

$$g(x) = (x-\tau_1)(x-\tau_2)\ldots(x-\tau_r).$$

Note that $g \in \mathbb{P}_r$, thus, by definition of p_k,

$$\int_{-1}^{1} p_k(x)\, g(x)\; \mathrm{d}x = 0.$$

This is a contradiction. Note that we can write $p_k(x) = g(x)q(x)$, with $q \in \mathbb{P}_{k-r}$, where, by construction of g, $q(x)$ does not possess any root in $(-1,1)$. Therefore, $p_k g$ does not have any sign change inside $(-1,1)$, and its integral cannot vanish. □

2.4.2 Gauss quadrature rules

Let us move back to our original problem. We were looking for the optimal integration nodes $c_1, \dots, c_S$ such that $M(t) := (t-c_1)(t-c_2)\dots(t-c_S) \in \mathbb{P}_S$ satisfies:

$$\int_0^1 M(t)\, g(t)\; \mathrm{d}t = 0 \quad \forall\, g \in \mathbb{P}_{S-1}.$$

It turns out that it is sufficient to choose as nodes the roots of S-th Legendre polynomial (rescaled in $[0,1]$). Indeed, by taking $M(t) := p_S(2t-1)$, $t \in [0,1]$, we have

$$\int_0^1 p_s(2t-1)\, g(t)\; \mathrm{d}t = \int_{-1}^{1} \frac{1}{2} p_s(x)\, g\left(\frac{x+1}{2}\right)\; \mathrm{d}x = 0 \quad \forall\, g \in \mathbb{P}_{S-1}.$$

What has been said can be summarized in the following result.

Theorem 2.8 (Gauss quadrature formula)**.** *For each integer $S > 0$, there exists a unique quadrature rule Q_S^g, called Gauss quadrature formula, with optimal order $2S$ such that*

- *the nodes $c_1, \dots, c_S$ are the roots of $p_S(2t-1)$,*
- *the weights $b_1, \dots, b_S$ are given by (2.4).*

Example 2.4. *Let us exhibit the Gauss quadrature formulae for $S = 1, 2, 3$.*

- *For $S = 1$:*
$$Q_1^g(g) = g\left(\frac{1}{2}\right),$$
hence the midpoint rule is the Gauss quadrature formula with one quadrature node.
- *For $S = 2$:*
$$Q_2^g(g) = \frac{1}{2} g\left(\frac{1}{2} - \frac{\sqrt{3}}{6}\right) + \frac{1}{2} g\left(\frac{1}{2} + \frac{\sqrt{3}}{6}\right).$$
- *For $S = 3$:*
$$Q_3^g(g) = \frac{5}{18} g\left(\frac{1}{2} - \frac{\sqrt{15}}{10}\right) + \frac{8}{18} g\left(\frac{1}{2}\right) + \frac{5}{18} g\left(\frac{1}{2} + \frac{\sqrt{15}}{10}\right).$$

Remark 2.8. *There is another way to compute the quadrature weights of Q_S^g, instead of solving the* *linear system* (2.4). *We can indeed use the following explicit formula:*

$$b_i = \frac{1-\gamma_i^2}{S^2\left(p_{S-1}\left(\gamma_i\right)\right)^2} \qquad \forall\ i = 1,\dots,S,$$

where $\gamma_1,\dots,\gamma_S$ *are the roots of the* S*-th Legendre polynomial* p_S *in* $(-1,1)$. *The proof of this result is omitted.*

2.5 Implementational aspects

In this section we provide a possible implementation (written in MATLAB/Octave pseudo-code) of the most important methods introduced above.

2.5.1 Implementation of Netwon-Cotes formulae

Let us start with the algorithm for the composite Newton-Cotes formulae, implemented in Listing 2.1. We consider a method with S nodes, where the integration domain $[a,b]$ is subdivided into n equispaced sub-intervals, where all these parameters are passed as input to the function. The implementation looks as follows:

```
function Q = newton_cotes(f,a,b,n,S)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Q = NEWTON_COTES(FUN,A,B,N,S) Computes an approximation of the integral
% of FUN over the interval (A,B), subdivided into N sub-intervals,
% via a Newton-Cotes formula with S nodes.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n=(S-1)*ceil(n/(S-1));  M1=n+1;
x=linspace(a,b,M1)';
h=x(2)-x(1); g=f(x);
endpts=g(1)+g(M1);

switch S
    case{2} % Trapezoidal Rule
        Q=(h/2)*(endpts+2*sum(g(2:n)));
    case{3} % Simpson's Rule
        Q=(h/3)*(endpts+4*sum(g(2:2:n))+2*sum(g(3:2:n)));
    case{4} % Simpson's 3/8 Rule
        Q=(3*h/8)*(endpts+3*sum(g(2:3:n)+g(3:3:n))+2*sum(g(4:3:n)));
    case{5} % Boole's 4/90 Rule
        Q=(4*h/90)*(7*endpts+32*sum(g(2:4:n))+...
            12*sum(g(3:4:n))+32*sum(g(4:4:n))+14*sum(g(5:4:n)));
    case{6}
        Q=(5*h/288)*(19*endpts+75*sum(g(2:5:n)+g(5:5:n))+...
            50*sum(g(3:5:n)+g(4:5:n))+38*sum(g(6:5:n)));
    case{7}
```

```
        Q=(6*h/840)*(41*endpts+216*sum(g(2:6:n)+g(6:6:n))+...
           27*sum(g(3:6:n)+g(5:6:n))+272*sum(g(4:6:n))+...
           +82*sum(g(7:6:n)));
    case{8}
        Q=(7*h/17280)*(751*endpts+3577*sum(g(2:7:n)+g(7:7:n))+...
           1323*sum(g(3:7:n)+g(6:7:n))+2989*sum(g(4:7:n)+g(5:7:n))+...
           +1502*sum(g(8:7:n)));
    case{9}
        Q=(4*h/14175)*(989*endpts+5888*sum(g(2:8:n)+g(8:8:n))-...
           928*sum(g(3:8:n)+g(7:8:n))+10496*sum(g(4:8:n)+g(6:8:n))-...
           4540*sum(g(5:8:n))+1978*sum(g(9:8:n)));
    case{10}
        Q=(9*h/89600)*(2857*endpts+15741*sum(g(2:9:n)+g(9:9:n))+...
           1080*sum(g(3:9:n)+g(8:9:n))+19344*sum(g(4:9:n)+g(7:9:n))+...
           5778*sum(g(5:9:n)+g(6:9:n))+5714*sum(g(10:9:n)));
    case{11}
        Q=(5*h/299376)*(16067*endpts+106300*sum(g(2:10:n)+g(10:10:n))-...
           48525*sum(g(3:10:n)+g(9:10:n))+272400*sum(g(4:10:n)+g(8:10:n))-...
           260550*sum(g(5:10:n)+g(7:10:n))+427368*sum(g(6:10:n))+...
           32134*sum(g(11:10:n)));
    otherwise
        error('Order must be between 2 and 11');
end

return
```

Listing 2.1: Algorithm for computing the Newton Cotes formulae.

An example on how to use this function is provided in the following script:

```
% test Netwon-Cotes
f = @(x) sin(x);
a = 0; b = pi;
n = 4; S = 3;
Q = newton_cotes(f,a,b,n,S)
```

2.5.2 Implementation of Gauss-Legendre quadrature

In Listing 2.2, a program is provided that computes the abscissae and weights of the Gauss-Legendre quadrature nodes. The function inputs are the corresponding Gauss quadrature, denoted by S, and the interval of interest, $[a, b]$.

```
function [x,w] = gauss_nodes_and_weights(a,b,S)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [X,W] = GAUSS_NODES_AND_WEIGHTS(A,B,S) computes the Gauss-Legendre nodes
% and weights on an interval [a,b] with truncation order S
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
S=S-1;
N1=S+1; N2=S+2;
xu=linspace(-1,1,N1)';
% Initial guess
y=cos((2*(0:S)'+1)*pi/(2*S+2))+(0.27/N1)*sin(pi*xu*S/N2);
% Legendre-Gauss Vandermonde Matrix (LGVM)
L=zeros(N1,N2);
% Derivative of LGVM
Lp=zeros(N1,N2);
% Compute the zeros of the S+1 Legendre polynomial
% using the recursion relation and the Newton-Raphson method
y0=2;
% Iterate until new points are uniformly within epsilon of old points
while max(abs(y-y0)) > eps
    L(:,1)=1; Lp(:,1)=0;

    L(:,2)=y; Lp(:,2)=1;

    for k=2:N1
        L(:,k+1)=( (2*k-1)*y.*L(:,k)-(k-1)*L(:,k-1) )/k;
    end

    Lp=(N2)*( L(:,N1)-y.*L(:,N2) )./(1-y.^2);

    y0=y;
    y=y0-L(:,N2)./Lp;
end

% Linear map from[-1,1] to [a,b]
x=(a*(1-y)+b*(1+y))/2;
% Compute the weights
w=(b-a)./((1-y.^2).*Lp.^2)*(N2/N1)^2;

return
```

Listing 2.2: Algorithm for computing the Gauss-Legendre quadrature points and weights.

Once the coordinates and the weights have been computed, an approximation of the integral of a function f over the interval $[a,b]$ can be obtained with the following commands:

```
% test Gauss-Legendre
f = @(x) sin(2*pi*x);
a = 0; b = 1;
S = 2;

[x,w] = gauss_nodes_and_weights(a,b,S);

% Summing the function evaluated at the quadrature
% nodes times the corresponding weights
Q = sum(f(x).*w)
```