

## Approximation of functions and data

---

Approximating a function  $f$  consists of replacing it by another function  $\tilde{f}$  of simpler form that may be used as its surrogate. This strategy is used frequently in numerical integration where, instead of computing  $\int_a^b f(x)dx$ , one carries out the exact computation of  $\int_a^b \tilde{f}(x)dx$ ,  $\tilde{f}$  being a function simple to integrate (e.g. a polynomial), as we will see in the next chapter. In other instances the function  $f$  may be available only partially through its values at some selected points. In these cases we aim at constructing a continuous function  $\tilde{f}$  that could represent the empirical law which is behind the finite set of data. We provide some examples which illustrate this kind of approach.

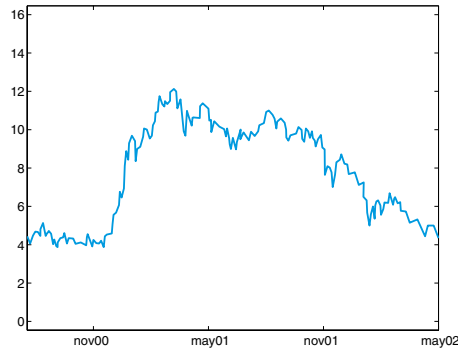
**Problem 3.1 (Climatology)** The air temperature near the ground depends on the concentration  $K$  of the carbon acid ( $\text{H}_2\text{CO}_3$ ) therein. In Table 3.1 (taken from Philosophical Magazine 41, 237 (1896)) we report for different latitudes on the Earth and for four different values of  $K$ , the variation  $\delta_K = \theta_K - \theta_{\bar{K}}$  of the average temperature with respect to the average temperature corresponding to a reference value  $\bar{K}$  of  $K$ . Here  $\bar{K}$  refers to the value measured in 1896, and is normalized to one. In this case we can generate a function that, on the basis of the available data, provides an approximate value of the average temperature at any possible latitude and for other values of  $K$  (see Example 3.1). ■

**Problem 3.2 (Finance)** In Figure 3.1 we report the price of a stock at the Zurich stock exchange over two years. The curve was obtained by joining with a straight line the prices reported at every day's closure. This simple representation indeed implicitly assumes that the prices change linearly in the course of the day (we anticipate that this approximation is called composite linear interpolation). We ask whether from this graph one could predict the stock price for a short time interval beyond the time of the last quotation. We will see in Section 3.4 that this kind of

Latitude	$\delta_K$			
	$K = 0.67$	$K = 1.5$	$K = 2.0$	$K = 3.0$
65	-3.1	3.52	6.05	9.3
55	-3.22	3.62	6.02	9.3
45	-3.3	3.65	5.92	9.17
35	-3.32	3.52	5.7	8.82
25	-3.17	3.47	5.3	8.1
15	-3.07	3.25	5.02	7.52
5	-3.02	3.15	4.95	7.3
-5	-3.02	3.15	4.97	7.35
-15	-3.12	3.2	5.07	7.62
-25	-3.2	3.27	5.35	8.22
-35	-3.35	3.52	5.62	8.8
-45	-3.37	3.7	5.95	9.25
-55	-3.25	3.7	6.1	9.5

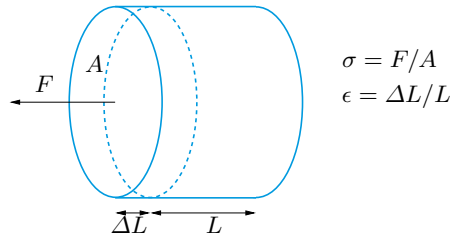
**Table 3.1.** Variation of the average yearly temperature on the Earth for four different values of the concentration  $K$  of carbon acid at different latitudes

prediction could be guessed by resorting to a special technique known as *least-squares* approximation of data (see Example 3.9). ■



**Fig. 3.1.** Price variation of a stock over two years

**Problem 3.3 (Biomechanics)** We consider a mechanical test to establish the link between stresses (MPa= 100 N/cm<sup>2</sup>) and deformations of a sample of biological tissue (an intervertebral disc, see Figure 3.2). Starting from the data collected in Table 3.2 (taken from P.Komarek, Chapt. 2 of *Biomechanics of Clinical Aspects of Biomedicine*, 1993, J.Valenta ed., Elsevier) in Example 3.10 we will estimate the deformation corresponding to a stress  $\sigma = 0.9$  MPa. ■



**Fig. 3.2.** A schematic representation of an intervertebral disc

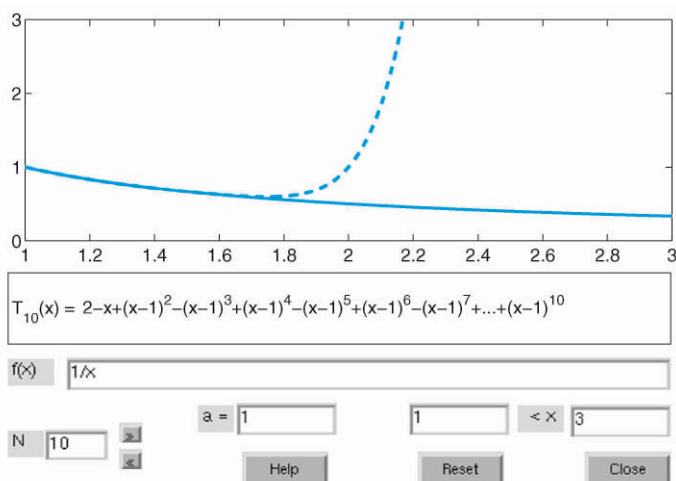
test	stress $\sigma$	stress $\epsilon$	test	stress $\sigma$	stress $\epsilon$
1	0.00	0.00	5	0.31	0.23
2	0.06	0.08	6	0.47	0.25
3	0.14	0.14	7	0.60	0.28
4	0.25	0.20	8	0.70	0.29

**Table 3.2.** Values of the deformation for different values of a stress applied on an intervertebral disc

**Problem 3.4 (Robotics)** We want to approximate the planar trajectory followed by a robot (idealized as a material point) during a working cycle in an industry. The robot should satisfy a few constraints: it must be steady at the point  $(0, 0)$  in the plane at the initial time (say,  $t = 0$ ), transit through the point  $(1, 2)$  at  $t = 1$ , get the point  $(4, 4)$  at  $t = 2$ , stop and restart immediately and reach the point  $(3, 1)$  at  $t = 3$ , return to the initial point at time  $t = 5$ , stop and restart a new working cycle. In Example 3.7 we will solve this problem using the *splines* functions. ■

A function  $f$  can be replaced in a given interval by its Taylor polynomial, which was introduced in Section 1.4.3. This technique is computationally expensive since it requires the knowledge of  $f$  and its derivatives up to the order  $n$  (the polynomial degree) at a given point  $x_0$ . Moreover, the Taylor polynomial may fail to accurately represent  $f$  far enough from the point  $x_0$ . For instance, in Figure 3.3 we compare the behavior of  $f(x) = 1/x$  with that of its Taylor polynomial of degree 10 built around the point  $x_0 = 1$ . This picture also shows the graphical interface of the MATLAB function `taylortool` which allows the computation of Taylor's polynomial of arbitrary degree for any given function  $f$ . The agreement between the function and its Taylor polynomial is very good in a small neighborhood of  $x_0 = 1$  while it becomes unsatisfactory when  $x - x_0$  gets large. Fortunately, this is not the case of other functions such as the exponential function which is approximated quite nicely for all  $x \in \mathbb{R}$  by its Taylor polynomial related to  $x_0 = 0$ , provided that the degree  $n$  is sufficiently large.

In the course of this chapter we will introduce approximation methods that are based on alternative approaches.



**Fig. 3.3.** Comparison between the function  $f(x) = 1/x$  (solid line) and its Taylor polynomial of degree 10 related to the point  $x_0 = 1$  (dashed line). The explicit form of the Taylor polynomial is also reported

### 3.1 Interpolation

As seen in Problems 3.1, 3.2 and 3.3, in several applications it may happen that a function is known only through its values at some given points. We are therefore facing a (general) case where  $n + 1$  couples  $\{x_i, y_i\}$ ,  $i = 0, \dots, n$ , are given; the points  $x_i$  are all distinct and are called *nodes*.

For instance in the case of Table 3.1,  $n$  is equal to 12, the nodes  $x_i$  are the values of the latitude reported in the first column, while the  $y_i$  are the corresponding values (of the temperature) in the remaining columns.

In such a situation it seems natural to require the approximate function  $\tilde{f}$  to satisfy the set of relations

$$\tilde{f}(x_i) = y_i, \quad i = 0, 1, \dots, n \quad (3.1)$$

Such an  $\tilde{f}$  is called *interpolant* of the set of data  $\{y_i\}$  and equations (3.1) are the interpolation conditions.

Several kinds of interpolants could be envisaged, such as:

- *polynomial interpolant*:

$$\tilde{f}(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n;$$

- *trigonometric interpolant*:

$$\tilde{f}(x) = a_{-M}e^{-iMx} + \dots + a_0 + \dots + a_Me^{iMx}$$

where  $M$  is an integer equal to  $n/2$  if  $n$  is even,  $(n-1)/2$  if  $n$  is odd, and  $i$  is the imaginary unit;

- *rational interpolant*:

$$\tilde{f}(x) = \frac{a_0 + a_1x + \dots + a_kx^k}{a_{k+1} + a_{k+2}x + \dots + a_{k+n+1}x^n}.$$

For simplicity we only consider those interpolants which depend linearly on the unknown coefficients  $a_i$ . Both polynomial and trigonometric interpolation fall into this category, whereas the rational interpolant does not.

### 3.1.1 Lagrangian polynomial interpolation

Let us focus on the polynomial interpolation. The following result holds:

**Proposition 3.1** *For any set of couples  $\{x_i, y_i\}$ ,  $i = 0, \dots, n$ , with distinct nodes  $x_i$ , there exists a unique polynomial of degree less than or equal to  $n$ , which we indicate by  $\Pi_n$  and call interpolating polynomial of the values  $y_i$  at the nodes  $x_i$ , such that*

$$\Pi_n(x_i) = y_i, i = 0, \dots, n \quad (3.2)$$

*In the case where the  $\{y_i, i = 0, \dots, n\}$  represent the values of a continuous function  $f$ ,  $\Pi_n$  is called interpolating polynomial of  $f$  (in short, interpolant of  $f$ ) and will be denoted by  $\Pi_n f$ .*

To verify uniqueness we proceed by contradiction and suppose that there exist two distinct polynomials of degree  $n$ ,  $\Pi_n$  and  $\Pi_n^*$ , both satisfying the nodal relation (3.2). Their difference,  $\Pi_n - \Pi_n^*$ , would be a polynomial of degree  $n$  which vanishes at  $n+1$  distinct points. Owing to a well known theorem of Algebra, such a polynomial should vanish identically, and then  $\Pi_n^*$  must coincide with  $\Pi_n$ .

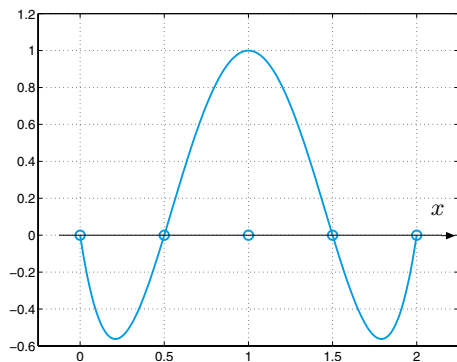
In order to obtain an expression for  $\Pi_n$ , we start from a very special case where  $y_i$  vanishes for all  $i$  apart from  $i = k$  (for a fixed  $k$ ) for which  $y_k = 1$ . Then setting  $\varphi_k(x) = \Pi_n(x)$ , we must have (see Figure 3.4)

$$\varphi_k \in \mathbb{P}_n, \varphi_k(x_j) = \delta_{jk} = \begin{cases} 1 & \text{if } j = k, \\ 0 & \text{otherwise,} \end{cases}$$

where  $\delta_{jk}$  is the Kronecker symbol.

The functions  $\varphi_k$  have the following expression:

$$\varphi_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^n \frac{x - x_j}{x_k - x_j}, \quad k = 0, \dots, n. \quad (3.3)$$



**Fig. 3.4.** The polynomial  $\varphi_2 \in \mathbb{P}_4$  associated with a set of 5 equispaced nodes

We move now to the general case where  $\{y_i, i = 0, \dots, n\}$  is a set of arbitrary values. Using an obvious superposition principle we can obtain the following expression for  $\Pi_n$

$$\Pi_n(x) = \sum_{k=0}^n y_k \varphi_k(x) \quad (3.4)$$

Indeed, this polynomial satisfies the interpolation conditions (3.2), since

$$\Pi_n(x_i) = \sum_{k=0}^n y_k \varphi_k(x_i) = \sum_{k=0}^n y_k \delta_{ik} = y_i, \quad i = 0, \dots, n.$$

Due to their special role, the functions  $\varphi_k$  are called *Lagrange characteristic polynomials*, and (3.4) is the *Lagrange form* of the interpolant. In MATLAB we can store the  $n+1$  couples  $\{(x_i, y_i)\}$  in the vectors  $\mathbf{x}$  and  $\mathbf{y}$ , and then the instruction `c=polyfit(x,y,n)` will provide the coefficients of the interpolating polynomial. Precisely, `c(1)` will contain the coefficient of  $x^n$ , `c(2)` that of  $x^{n-1}$ ,  $\dots$  and `c(n+1)` the value of  $\Pi_n(0)$ . (More on this command can be found in Section 3.4.) As already seen in Chapter 1, we can then use the instruction `p=polyval(c,z)` to compute the value `p(j)` attained by the interpolating polynomial at `z(j)`,  $j=1, \dots, m$ , the latter being a set of  $m$  arbitrary points.

In the case when the explicit form of the function  $\mathbf{f}$  is available, we can use the instruction `y=eval(f)` in order to obtain the vector  $\mathbf{y}$  of values of  $\mathbf{f}$  at some specific nodes (which should be stored in a vector  $\mathbf{x}$ ).

**Example 3.1 (Climatology)** To obtain the interpolating polynomial for the data of Problem 3.1 relating to the value  $K = 0.67$  (first column of Table 3.1), using only the values of the temperature for the latitudes 65, 35, 5, -25, -55, we can use the following MATLAB instructions:

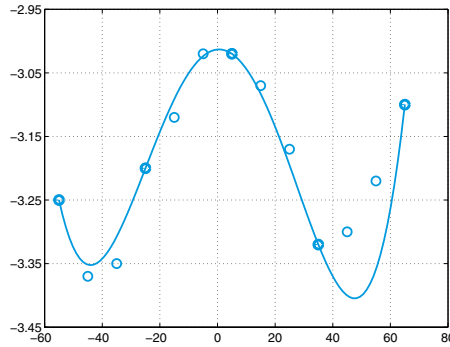
```
x=[-55 -25 5 35 65]; y=[-3.25 -3.2 -3.02 -3.32 -3.1];
format short e; c=polyfit(x,y,4)
```

```
c =
 8.2819e-08 -4.5267e-07 -3.4684e-04 3.7757e-04 -3.0132e+00
```

The graph of the interpolating polynomial can be obtained as follows:

```
z=linspace(x(1),x(end),100);
p=polyval(c,z);
plot(z,p);hold on;plot(x,y,'o');grid on;
```

In order to get a smooth curve we have evaluated our polynomial at 101 equispaced points in the interval  $[-55, 65]$  (as a matter of fact, MATLAB plots are always constructed on piecewise linear interpolation between neighboring points). Note that the instruction `x(end)` picks up directly the last component of the vector `x`, without specifying the length of the vector. In Figure 3.5 the filled circles correspond to those values which have been used to construct the interpolating polynomial, whereas the empty circles correspond to values that have not been used. We can appreciate the qualitative agreement between the curve and the data distribution. ■



**Fig. 3.5.** The interpolating polynomial of degree 4 introduced in Example 3.1

Using the following result we can evaluate the error obtained by replacing  $f$  with its interpolating polynomial  $\Pi_n f$ :

**Proposition 3.2** *Let  $I$  be a bounded interval, and consider  $n + 1$  distinct interpolation nodes  $\{x_i, i = 0, \dots, n\}$  in  $I$ . Let  $f$  be continuously differentiable up to order  $n + 1$  in  $I$ .*

Then  $\forall x \in I \exists \xi \in I$  such that

$$E_n f(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^n (x - x_i) \quad (3.5)$$

Obviously,  $E_n f(x_i) = 0$ ,  $i = 0, \dots, n$ .

Result (3.5) can be better specified in the case of a uniform distribution of nodes, that is when  $x_i = x_{i-1} + h$  for  $i = 1, \dots, n$ , for a given  $h > 0$  and a given  $x_0$ . As stated in Exercise 3.1,  $\forall x \in (x_0, x_n)$  one can verify that

$$\left| \prod_{i=0}^n (x - x_i) \right| \leq n! \frac{h^{n+1}}{4}, \quad (3.6)$$

and therefore

$$\max_{x \in I} |E_n f(x)| \leq \frac{\max_{x \in I} |f^{(n+1)}(x)|}{4(n+1)!} h^{n+1}. \quad (3.7)$$

Unfortunately, we cannot deduce from (3.7) that the error tends to 0 when  $n \rightarrow \infty$ , in spite of the fact that  $h^{n+1}/[4(n+1)!]$  tends to 0. In fact, as shown in Example 3.2, there exist functions  $f$  for which the limit can even be infinite, that is

$$\lim_{n \rightarrow \infty} \max_{x \in I} |E_n f(x)| = \infty.$$

This striking result indicates that by increasing the degree  $n$  of the interpolating polynomial we do not necessarily obtain a better reconstruction of  $f$ . For instance, should we use all data of the second column of Table 3.1, we would obtain the interpolating polynomial  $\Pi_{12} f$  represented in Figure 3.6, whose behavior in the vicinity of the left-hand of the interval is far less satisfactory than that obtained in Figure 3.5 using a much smaller number of nodes. An even worse result may arise for a special class of functions, as we report in the next example.

**Example 3.2 (Runge)** If the function  $f(x) = 1/(1+x^2)$  is interpolated at equispaced nodes in the interval  $I = (-5, 5)$ , the error  $\max_{x \in I} |E_n f(x)|$  tends to infinity when  $n \rightarrow \infty$ . This is due to the fact that if  $n \rightarrow \infty$  the order of magnitude of  $\max_{x \in I} |f^{(n+1)}(x)|$  outweighs the infinitesimal order of  $h^{n+1}/[4(n+1)!]$ . This conclusion can be verified by computing the maximum of  $f$  and its derivatives up to the order 21 by means of the following MATLAB instructions:

```
syms x; n=20; f=1/(1+x^2); df=diff(f,1);
cdf = char(df);
for i = 1:n+1, df = diff(df,1); cdfn = char(df);
    x = fzero(cdfn,0); M(i) = abs(eval(cdf)); cdf = cdfn;
end
```



The maximum of the absolute values of the functions  $f^{(n)}$ ,  $n = 1, \dots, 21$ , are stored in the vector **M**. Notice that the command **char** converts the symbolic expression **df** into a string that can be evaluated by the function **fzero**. In particular, the absolute values of  $f^{(n)}$  for  $n = 3, 9, 15, 21$  are:

```
>> M([3,9,15,21]) =
ans =
    4.6686e+00    3.2426e+05    1.2160e+12    4.8421e+19
```

while the corresponding values of the maximum of  $\prod_{i=0}^n (x - x_i)/(n+1)!$  are

```
z = linspace(-5,5,10000);
for n=0:20; h=10/(n+1); x=[-5:h:5];
    c=poly(x);
    r(n+1)=max(polyval(c,z));
    r(n+1)=r(n+1)/prod([1:n+2]);
end
r([3,9,15,21])
```

```
ans =
    2.8935e+00    5.1813e-03    8.5854e-07    2.1461e-11
```

**c=poly(x)** is a vector whose components are the coefficients of that polynomial whose roots are the elements of the vector **x**. It follows that  $\max_{x \in I} |E_n f(x)|$  attains the following values:

poly

```
>> format short e;
    1.3509e+01    1.6801e+03    1.0442e+06    1.0399e+09
```

for  $n = 3, 9, 15, 21$ , respectively.

The lack of convergence is also indicated by the presence of severe oscillations in the graph of the interpolating polynomial with respect to the graph of  $f$ , especially near the endpoints of the interval (see Figure 3.6, right). This behavior is known as *Runge's phenomenon*. ■

Besides (3.7), the following inequality can also be proved:

$$\max_{x \in I} |f'(x) - (\Pi_n f)'(x)| \leq Ch^n \max_{x \in I} |f^{(n+1)}(x)|,$$

where  $C$  is a constant independent of  $h$ . Therefore, if we approximate the first derivative of  $f$  by the first derivative of  $\Pi_n f$ , we loose an order of convergence with respect to  $h$ .

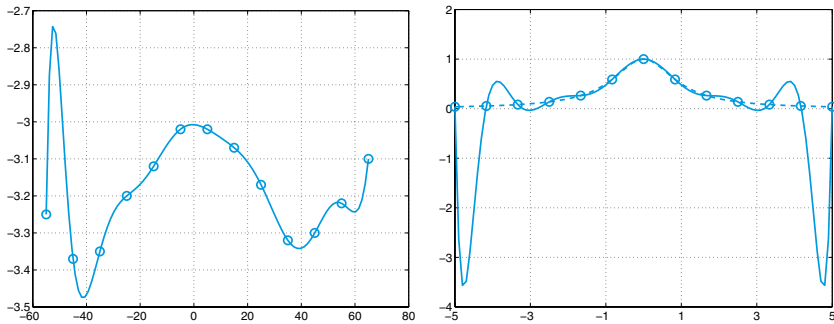
In MATLAB,  $(\Pi_n f)'$  can be computed using the instruction **[d]=polyder(c)**, where **c** is the input vector in which we store the coefficients of the interpolating polynomial, while **d** is the output vector where we store the coefficients of its first derivative (see Section 1.4.2).

polyder

**Octave 3.1** The analogous command in Octave is **[d]=polyderiv (c)**. ■

See the Exercises 3.1-3.4.





**Fig. 3.6.** Two examples of Runge's phenomenon: to the left,  $\Pi_{12}$  computed for the data of Table 3.1, column  $K = 0.67$ ; to the right,  $\Pi_{12}f$  (solid line) computed on 13 equispaced nodes for the function  $f(x) = 1/(1+x^2)$  (dashed line)

### 3.1.2 Chebyshev interpolation

Runge's phenomenon can be avoided if a suitable distribution of nodes is used. In particular, in an arbitrary interval  $[a, b]$ , we can consider the so called *Chebyshev nodes* (see Figure 3.7, right):

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \hat{x}_i, \text{ where } \hat{x}_i = -\cos(\pi i/n), i = 0, \dots, n \quad (3.8)$$

Obviously,  $x_i = \hat{x}_i$ ,  $i = 0, \dots, n$ , when  $[a, b] = [-1, 1]$ . Indeed, for this special distribution of nodes it is possible to prove that, if  $f$  is a continuous and differentiable function in  $[a, b]$ ,  $\Pi_n f$  converges to  $f$  as  $n \rightarrow \infty$  for all  $x \in [a, b]$ .

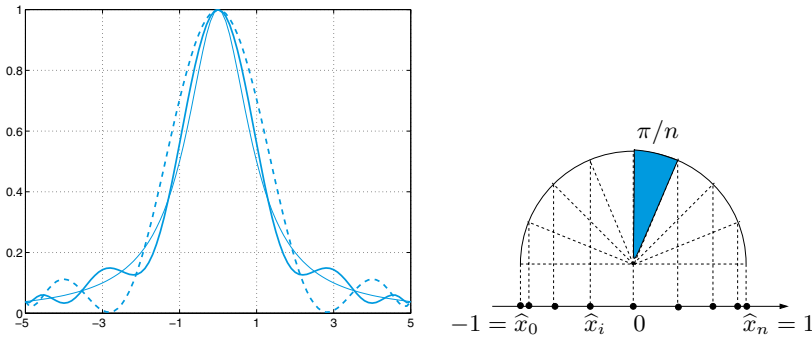
The Chebyshev nodes, which are the abscissas of equispaced nodes on the unit semi-circumference, lie inside  $[a, b]$  and are clustered near the endpoints of this interval (see Figure 3.7).

Another non-uniform distribution of nodes in the interval  $(a, b)$ , sharing the same convergence properties of Chebyshev nodes, is provided by:

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos\left(\frac{2i+1}{n+1} \frac{\pi}{2}\right), i = 0, \dots, n \quad (3.9)$$

**Example 3.3** We consider anew the function  $f$  of Runge's example and compute its interpolating polynomial at Chebyshev nodes. The latter can be obtained through the following MATLAB instructions:

```
xc = -cos(pi*[0:n]/n); x = (a+b)*0.5+(b-a)*xc*0.5;
```



**Fig. 3.7.** The left side picture shows the comparison between the function  $f(x) = 1/(1+x^2)$  (thin solid line) and its Chebyshev interpolating polynomials of degree 8 (dashed line) and 12 (solid line). Note that the amplitude of spurious oscillations decreases as the degree increases. The right side picture shows the distribution of Chebyshev nodes in the interval  $[-1, 1]$

where  $n+1$  is the number of nodes, while  $a$  and  $b$  are the endpoints of the interpolation interval (in the sequel we choose  $a=-5$  and  $b=5$ ). Then we compute the interpolating polynomial by the following instructions:

```
f = '1./(1+x.^2)'; y = eval(f); c = polyfit(x,y,n);
```

Now let us compute the absolute values of the differences between  $f$  and its Chebyshev interpolant at as many as 1001 equispaced points in the interval  $[-5, 5]$  and take the maximum error values:

```
x = linspace(-5,5,1000); p=polyval(c,x);
fx = eval(f); err = max(abs(p-fx));
```

As we see in Table 3.3, the maximum of the error decreases when  $n$  increases. ■

$n$	5	10	20	40
$E_n$	0.6386	0.1322	0.0177	0.0003

**Table 3.3.** The Chebyshev interpolation error for Runge's function  $f(x) = 1/(1+x^2)$

### 3.1.3 Trigonometric interpolation and FFT

We want to approximate a periodic function  $f : [0, 2\pi] \rightarrow \mathbb{C}$ , i.e. one satisfying  $f(0) = f(2\pi)$ , by a trigonometric polynomial  $\tilde{f}$  which interpolates  $f$  at the  $n+1$  nodes  $x_j = 2\pi j/(n+1)$ ,  $j = 0, \dots, n$ , i.e.

$$\tilde{f}(x_j) = f(x_j), \text{ for } j = 0, \dots, n. \quad (3.10)$$



The *trigonometric interpolant*  $\tilde{f}$  is obtained by a linear combination of sines and cosines.

In particular, if  $n$  is even,  $\tilde{f}$  will have the form

$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^M [a_k \cos(kx) + b_k \sin(kx)], \quad (3.11)$$

where  $M = n/2$  while, if  $n$  is odd,

$$\begin{aligned} \tilde{f}(x) = \\ \frac{a_0}{2} + \sum_{k=1}^M [a_k \cos(kx) + b_k \sin(kx)] + a_{M+1} \cos((M+1)x), \end{aligned} \quad (3.12)$$

where  $M = (n-1)/2$ . We can rewrite (3.11) as

$$\tilde{f}(x) = \sum_{k=-M}^M c_k e^{ikx}, \quad (3.13)$$

$i$  being the imaginary unit. The complex coefficients  $c_k$  are related to the coefficients  $a_k$  and  $b_k$  (complex too) as follows:

$$a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}), \quad k = 0, \dots, M. \quad (3.14)$$

Indeed, from (1.5) it follows that  $e^{ikx} = \cos(kx) + i \sin(kx)$  and

$$\begin{aligned} \sum_{k=-M}^M c_k e^{ikx} &= \sum_{k=-M}^M c_k (\cos(kx) + i \sin(kx)) \\ &= \sum_{k=1}^M [c_k (\cos(kx) + i \sin(kx)) + c_{-k} (\cos(kx) - i \sin(kx))] + c_0. \end{aligned}$$

Therefore we derive (3.11), thanks to the relations (3.14).

Analogously, when  $n$  is odd, (3.12) becomes

$$\tilde{f}(x) = \sum_{k=-(M+1)}^{M+1} c_k e^{ikx}, \quad (3.15)$$

where the coefficients  $c_k$  for  $k = 0, \dots, M$  are the same as before, while  $c_{M+1} = c_{-(M+1)} = a_{M+1}/2$ . In both cases, we could write

$$\tilde{f}(x) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikx}, \quad (3.16)$$

with  $\mu = 0$  if  $n$  is even and  $\mu = 1$  if  $n$  is odd. Should  $f$  be real valued, its coefficients  $c_k$  satisfy  $c_{-k} = \bar{c}_k$ ; from (3.14) it follows that the coefficients  $a_k$  and  $b_k$  are all real.

Because of its analogy with Fourier series,  $\tilde{f}$  is called a *discrete Fourier series*. Imposing the interpolation condition at the nodes  $x_j = jh$ , with  $h = 2\pi/(n+1)$ , we find that

$$\sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjh} = f(x_j), \quad j = 0, \dots, n. \quad (3.17)$$

For the computation of the coefficients  $\{c_k\}$  let us multiply equations (3.17) by  $e^{-imx_j} = e^{-imjh}$ , where  $m$  is an integer between 0 and  $n$ , and then sum with respect to  $j$ :

$$\sum_{j=0}^n \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjh} e^{-imjh} = \sum_{j=0}^n f(x_j) e^{-imjh}. \quad (3.18)$$

We now require the following identity:

$$\sum_{j=0}^n e^{ijh(k-m)} = (n+1)\delta_{km}.$$

This identity is obviously true if  $k = m$ . When  $k \neq m$ , we have

$$\sum_{j=0}^n e^{ijh(k-m)} = \frac{1 - (e^{i(k-m)h})^{n+1}}{1 - e^{i(k-m)h}}.$$

The numerator on the right hand side is null, since

$$\begin{aligned} 1 - e^{i(k-m)h(n+1)} &= 1 - e^{i(k-m)2\pi} \\ &= 1 - \cos((k-m)2\pi) - i \sin((k-m)2\pi). \end{aligned}$$

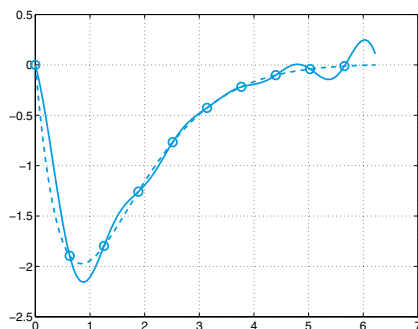
Therefore, from (3.18) we get the following explicit expression for the coefficients of  $\tilde{f}$ :

$$c_k = \frac{1}{n+1} \sum_{j=0}^n f(x_j) e^{-ikjh}, \quad k = -(M+\mu), \dots, M+\mu \quad (3.19)$$

The computation of all the coefficients  $\{c_k\}$  can be accomplished with an order  $n \log_2 n$  operations by using the *fast Fourier transform* (FFT), which is implemented in the MATLAB program `fft` (see Example 3.4). Similar conclusions hold for the inverse transform through which we obtain the values  $\{f(x_j)\}$  from the coefficients  $\{c_k\}$ . The inverse fast Fourier transform is implemented in the MATLAB program `ifft`.

`fft`  
`ifft`

**Example 3.4** Consider the function  $f(x) = x(x - 2\pi)e^{-x}$  for  $x \in [0, 2\pi]$ . To use the MATLAB program `fft` we first compute the values of  $f$  at the nodes  $x_j = j\pi/5$  for  $j = 0, \dots, 9$  by the following instructions (recall that `.*` is the component-by-component vector product):



**Fig. 3.8.** The function  $f(x) = x(x - 2\pi)e^{-x}$  (dashed line) and the corresponding trigonometric interpolant (continuous line) relative to 10 equispaced nodes

```
x=pi/5*[0:9]; y=x.*(x-2*pi).*exp(-x);
```

Now by the FFT we compute the vector of the Fourier coefficients,  $Y = (n+1)[c_0, \dots, c_{M+\mu}, c_{-M}, \dots, c_{-1}]$ , by the following instructions:

```
Y=fft(y);
```

```
Y =
Columns 1 and 2:
-6.52032 + 0.00000i -0.46728 + 4.20012i
Columns 3 and 4:
1.26805 + 1.62110i 1.09849 + 0.60080i
Columns 5 and 6:
0.92585 + 0.21398i 0.87010 + 0.00000i
Columns 7 and 8:
0.92585 - 0.21398i 1.09849 - 0.60080i
Columns 9 and 10:
1.26805 - 1.62110i -0.46728 - 4.20012i
```

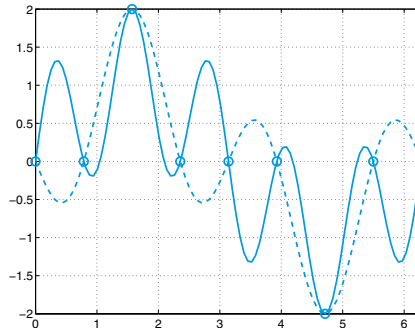
Note that the program `ifft` achieves the maximum efficiency when  $n$  is a power of 2, even though it works for any value of  $n$ . ■

`interpft`

The command `interpft` provides the trigonometric interpolant of a set of data. It requires in input an integer  $m$  and a vector of values which represent the values taken by a function (periodic with period  $p$ ) at the set of points  $x_j = jp/(n+1)$ ,  $j = 0, \dots, n$ . `interpft` returns the  $m$  values of the trigonometric interpolant, obtained by the Fourier transform, at the nodes  $t_i = ip/m$ ,  $i = 0, \dots, m-1$ . For instance, let us reconsider the function of Example 3.4 in  $[0, 2\pi]$  and take its values at 10 equispaced nodes  $x_j = j\pi/5$ ,  $j = 0, \dots, 9$ . The values of the trigonometric interpolant at, say, the 100 equispaced nodes  $t_i = i\pi/100$ ,  $i = 0, \dots, 99$  can be obtained as follows (see Figure 3.8)

```
x=pi/5*[0:9]; y=x.*(x-2*pi).*exp(-x); z=interpft(y,100);
```

In some cases the accuracy of trigonometric interpolation can dramatically downgrade, as shown in the following example.



**Fig. 3.9.** The effects of aliasing: comparison between the function  $f(x) = \sin(x) + \sin(5x)$  (solid line) and its trigonometric interpolant (3.11) with  $M = 3$  (dashed line)

**Example 3.5** Let us approximate the function  $f(x) = f_1(x) + f_2(x)$ , with  $f_1(x) = \sin(x)$  and  $f_2(x) = \sin(5x)$ , using nine equispaced nodes in the interval  $[0, 2\pi]$ . The result is shown in Figure 3.9. Note that in some intervals the trigonometric approximant shows even a phase inversion with respect to the function  $f$ . ■

This lack of accuracy can be explained as follows. At the nodes considered, the function  $f_2$  is indistinguishable from  $f_3(x) = -\sin(3x)$  which has a lower frequency (see Figure 3.10). The function that is actually approximated is therefore  $F(x) = f_1(x) + f_3(x)$  and not  $f(x)$  (in fact, the dashed line of Figure 3.9 does coincide with  $F$ ).

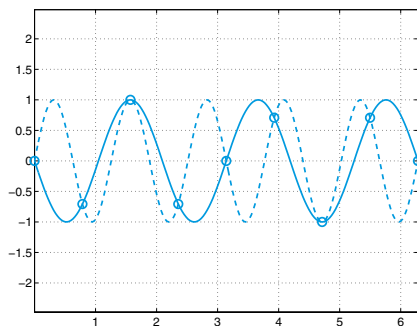
This phenomenon is known as *aliasing* and may occur when the function to be approximated is the sum of several components having different frequencies. As soon as the number of nodes is not enough to resolve the highest frequencies, the latter may interfere with the low frequencies, giving rise to inaccurate interpolants. To get a better approximation for functions with higher frequencies, one has to increase the number of interpolation nodes.

A real life example of aliasing is provided by the apparent inversion of the sense of rotation of spoked wheels. Once a certain critical velocity is reached the human brain is no longer able to accurately sample the moving image and, consequently, produces distorted images.

### Let us summarize

1. Approximating a set of data or a function  $f$  in  $[a, b]$  consists of finding a suitable function  $\tilde{f}$  that represents them with enough accuracy;
2. the interpolation process consists of determining a function  $\tilde{f}$  such that  $\tilde{f}(x_i) = y_i$ , where the  $\{x_i\}$  are given nodes and  $\{y_i\}$  are either the values  $\{f(x_i)\}$  or a set of prescribed values;





**Fig. 3.10.** The phenomenon of aliasing: the functions  $\sin(5x)$  (dashed line) and  $-\sin(3x)$  (dotted line) take the same values at the interpolation nodes. This circumstance explains the severe loss of accuracy shown in Figure 3.9

3. if the  $n + 1$  nodes  $\{x_i\}$  are distinct, there exists a unique polynomial of degree less than or equal to  $n$  interpolating a set of prescribed values  $\{y_i\}$  at the nodes  $\{x_i\}$ ;
4. for an equispaced distribution of nodes in  $[a, b]$  the interpolation error at any point of  $[a, b]$  does not necessarily tend to 0 as  $n$  tends to infinity. However, there exist special distributions of nodes, for instance the Chebyshev nodes, for which this convergence property holds true for all continuous functions;
5. trigonometric interpolation is well suited to approximate periodic functions, and is based on choosing  $\tilde{f}$  as a linear combination of sine and cosine functions. The FFT is a very efficient algorithm which allows the computation of the Fourier coefficients of a trigonometric interpolant from its node values and admits an equally fast inverse, the IFFT.

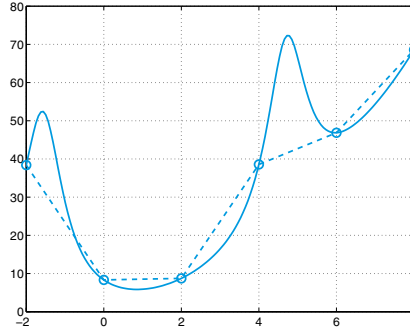
## 3.2 Piecewise linear interpolation

The Chebyshev interpolant provides an accurate approximation of smooth functions  $f$  whose expression is known. In the case when  $f$  is nonsmooth or when  $f$  is only known by its values at a set of given points (which do not coincide with the Chebyshev nodes), one can resort to a different interpolation method which is called linear composite interpolation.

More precisely, given a distribution (not necessarily uniform) of nodes  $x_0 < x_1 < \dots < x_n$ , we denote by  $I_i$  the interval  $[x_i, x_{i+1}]$ . We approximate  $f$  by a continuous function which, on each interval, is given by the segment joining the two points  $(x_i, f(x_i))$  and  $(x_{i+1}, f(x_{i+1}))$  (see Figure 3.11). This function, denoted by  $\Pi_1^H f$ , is called *piecewise linear interpolation polynomial* of  $f$  and its expression is:



$$\Pi_1^H f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) \quad \text{for } x \in I_i.$$



**Fig. 3.11.** The function  $f(x) = x^2 + 10/(\sin(x) + 1.2)$  (solid line) and its piecewise linear interpolation polynomial  $\Pi_1^H f$  (dashed line)

The upper-index  $H$  denotes the maximum length of the intervals  $I_i$ .

The following result can be inferred from (3.7) setting  $n = 1$  and  $h = H$ :

**Proposition 3.3** *If  $f \in C^2(I)$ , where  $I = [x_0, x_n]$ , then*

$$\max_{x \in I} |f(x) - \Pi_1^H f(x)| \leq \frac{H^2}{8} \max_{x \in I} |f''(x)|.$$

Consequently, for all  $x$  in the interpolation interval,  $\Pi_1^H f(x)$  tends to  $f(x)$  when  $H \rightarrow 0$ , provided that  $f$  is sufficiently smooth.

Through the instruction `s1=interp1(x,y,z)` one can compute the values at arbitrary points, which are stored in the vector `z`, of the piecewise linear polynomial that interpolates the values `y(i)` at the nodes `x(i)`, for  $i = 1, \dots, n+1$ . Note that `z` can have arbitrary dimension. If the nodes are in increasing order (i.e. `x(i+1) > x(i)`, for  $i=1, \dots, n$ ) then we can use the quicker version `interp1q` (q stands for quickly). Notice that `interp1q` is quicker than `interp1` on non-uniformly spaced data because it does not make any input checking.

It is worth mentioning that the command `fplot`, which is used to display the graph of a function  $f$  on a given interval  $[a, b]$ , does indeed replace the function by its piecewise linear interpolant. The set of interpolating nodes is generated automatically from the function, following the criterion of clustering these nodes around points where  $f$  shows strong variations. A procedure of this type is called *adaptive*.

**Octave 3.2** `interp1q` is not available in Octave. ■

### 3.3 Approximation by spline functions



As done for piecewise linear interpolation, piecewise polynomial interpolation of degree  $n \geq 2$  can be defined as well. For instance, the piecewise quadratic interpolation  $\Pi_2^H f$  is a continuous function that on each interval  $I_i$  replaces  $f$  by its quadratic interpolation polynomial at the endpoints of  $I_i$  and at its midpoint. If  $f \in C^3(I)$ , the error  $f - \Pi_2^H f$  in the maximum norm decays as  $H^3$  if  $H$  tends to zero.

The main drawback of this piecewise interpolation is that  $\Pi_k^H f$  with  $k \geq 1$ , is nothing more than a global continuous function. As a matter of fact, in several applications, e.g. in computer graphics, it is desirable to get approximation by smooth functions which have at least a continuous derivative.

With this aim, we can construct a function  $s_3$  with the following properties:

1. on each interval  $I_i = [x_i, x_{i+1}]$ , for  $i = 0, \dots, n-1$ ,  $s_3$  is a polynomial of degree 3 which interpolates the pairs of values  $(x_j, f(x_j))$  for  $j = i, i+1$ ;
2.  $s_3$  has continuous first and second derivatives in the nodes  $x_i$ ,  $i = 1, \dots, n-1$ .

For its complete determination, we need four conditions on each interval, therefore a total of  $4n$  equations, which we can provide as follows:

- $n+1$  conditions arise from the interpolation requirement at the nodes  $x_i$ ,  $i = 0, \dots, n$ ;
- $n-1$  further equations follow from the requirement of continuity of the polynomial at the internal nodes  $x_1, \dots, x_{n-1}$ ;
- $2(n-1)$  new equations are obtained by requiring that both first and second derivatives be continuous at the internal nodes.

We still lack two further equations, which we can e.g. choose as

$$s_3''(x_0) = 0, s_3''(x_n) = 0. \quad (3.20)$$

The function  $s_3$  which we obtain in this way, is called a *natural interpolating cubic spline*.

By choosing suitably the unknowns (see [QSS06, Section 8.6.1]) to represent  $s_3$  we arrive at a  $(n+1) \times (n+1)$  system with a tridiagonal matrix whose solution can be accomplished by a number of operations proportional to  $n$  (see Section 5.4) whose solutions are the values  $s''(x_i)$  for  $i = 0, \dots, n$ .

Using Program 3.1, this solution can be obtained with a number of operations equal to the dimension of the system itself (see Section 5.4). The input parameters are the vectors  $\mathbf{x}$  and  $\mathbf{y}$  of the nodes and the data to interpolate, plus the vector  $\mathbf{z}_i$  of the abscissae where we want the spline  $s_3$  to be evaluated.

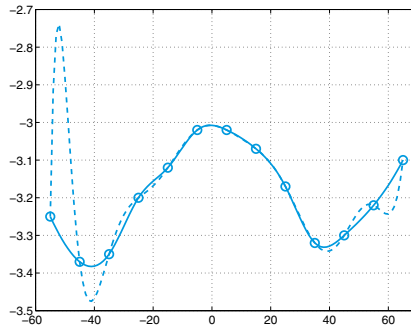
Other conditions can be chosen in place of (3.20) in order to close the system of equations; for instance we could prescribe the value of the first derivative of  $s_3$  at both endpoints  $x_0$  and  $x_n$ .

Unless otherwise specified, Program 3.1 computes the natural interpolation cubic spline. The optimal parameters **type** and **der** (a vector with two components) serve the purpose of selecting other types of splines. With **type**=0 Program 3.1 computes the interpolating cubic spline whose first derivative is given by **der**(1) at  $x_0$  and **der**(2) at  $x_n$ . With **type**=1 we obtain the interpolating cubic spline whose values of the second derivative at the endpoints is given by **der**(1) at  $x_0$  and **der**(2) at  $x_n$ .

**Program 3.1. cubicspline:** interpolating cubic spline

```
function s=cubicspline(x,y,zi,type,der)
%CUBICSPLINE compute a cubic spline
% S=CUBICSPLINE(X,Y,ZI) computes the value at the
% abscissae ZI of the natural interpolating cubic
% spline that interpolates the values Y at the nodes X.
% S=CUBICSPLINE(X,Y,ZI,TYPE,DER) if TYPE=0 computes the
% values at the abscissae ZI of the cubic spline
% interpolating the values Y with first derivative at
% the endpoints equal to the values DER(1) and DER(2).
% If TYPE=1 the values DER(1) and DER(2) are those of
% the second derivative at the endpoints.
[n,m]=size(x);
if n == 1
    x = x';    y = y';    n = m;
end
if nargin == 3
    der0 = 0; dern = 0; type = 1;
else
    der0 = der(1); dern = der(2);
end
h = x(2:end)-x(1:end-1);
e = 2*[h(1); h(1:end-1)+h(2:end); h(end)];
A = spdiags([h; 0] e [0; h]),-1:1,n,n);
d = (y(2:end)-y(1:end-1))./h;
rhs = 3*(d(2:end)-d(1:end-1));
if type == 0
    A(1,1) = 2*h(1);    A(1,2) = h(1);
    A(n,n) = 2*h(end); A(end,end-1) = h(end);
    rhs = [3*(d(1)-der0); rhs; 3*(dern-d(end))];
else
    A(1,:) = 0; A(1,1) = 1;
    A(n,:) = 0; A(n,n) = 1;
    rhs = [der0; rhs; dern];
end
S = zeros(n,4);
S(:,3) = A\rhs;
```





**Fig. 3.12.** Comparison between the interpolating cubic spline and the Lagrange interpolant for the case considered in Example 3.6

```

for m = 1:n-1
    S(m,4) = (S(m+1,3)-S(m,3))/3/h(m);
    S(m,2) = d(m) - h(m)/3*(S(m+1,3)+2*S(m,3));
    S(m,1) = y(m);
end
S = S(1:n-1, 4:-1:1);  pp = mkpp(x,S);  s = ppval(pp,zi);
return

```

**spline**

The MATLAB command **spline** (see also the toolbox **spines**) enforces the third derivative of  $s_3$  to be continuous at  $x_1$  and  $x_{n-1}$ . To this condition is given the curious name of *not-a-knot condition*. The input parameters are the vectors **x** and **y** and the vector **zi** (same meaning as before). The commands **mkpp** and **ppval** that are used in Program 3.1 are useful to build up and evaluate a composite polynomial.

**mkpp**  
**ppval**

**Example 3.6** Let us reconsider the data of Table 3.1 corresponding to the column  $K = 0.67$  and compute the associated interpolating cubic spline  $s_3$ . The different values of the latitude provide the nodes  $x_i, i = 0, \dots, 12$ . If we are interested in computing the values  $s_3(z_i)$ , where  $z_i = -55 + i, i = 0, \dots, 120$ , we can proceed as follows:

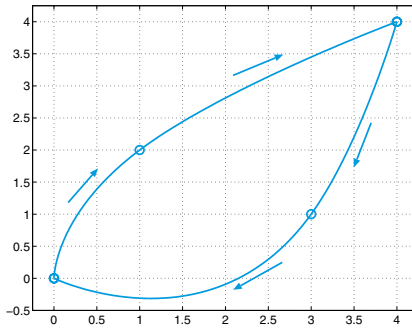
```

x = [-55:10:65];
y = [-3.25 -3.37 -3.35 -3.2 -3.12 -3.02 -3.02 ...
     -3.07 -3.17 -3.32 -3.3 -3.22 -3.1];
z = [-55:1:65];
s = spline(x,y,z);

```

The graph of  $s_3$ , which is reported in Figure 3.12, looks more plausible than that of the Lagrange interpolant at the same nodes. ■

**Example 3.7 (Robotics)** To find the trajectory of the robot satisfying the given constraints, we split the time interval  $[0, 5]$  in the two subintervals  $[0, 2]$  and  $[2, 5]$ . Then in each subinterval we look for two splines,  $x = x(t)$  and  $y = y(t)$ , that interpolate the given values and have null derivative at the endpoints. Using Program 3.1 we obtain the desired result by the following instructions:



**Fig. 3.13.** The trajectory in the  $xy$  plane of the robot described in Problem 3.4. Circles represent the position of the control points through which the robot should pass during its motion

```
x1 = [0 1 4]; y1 = [0 2 4];
t1 = [0 1 2]; ti1 = [0:0.01:2];
x2 = [0 3 4]; y2 = [0 1 4];
t2 = [0 2 3]; ti2 = [0:0.01:3]; d=[0,0];
six1 = cubicspline(t1,x1,ti1,0,d);
siy1 = cubicspline(t1,y1,ti1,0,d);
six2 = cubicspline(t2,x2,ti2,0,d);
siy2 = cubicspline(t2,y2,ti2,0,d);
```

The trajectory obtained is drawn in Figure 3.13. ■

The error that we obtain in approximating a function  $f$  (continuously differentiable up to its fourth derivative) by the natural interpolating cubic spline satisfies the following inequalities:

$$\max_{x \in I} |f^{(r)}(x) - s_3^{(r)}(x)| \leq C_r H^{4-r} \max_{x \in I} |f^{(4)}(x)|, \quad r = 0, 1, 2, 3,$$

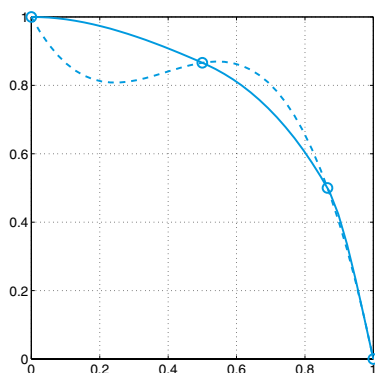
where  $I = [x_0, x_n]$  and  $H = \max_{i=0, \dots, n-1} (x_{i+1} - x_i)$ , while  $C_r$  is a suitable constant depending on  $r$ , but independent of  $H$ . It is then clear that not only  $f$ , but also its first, second and third derivatives are well approximated by  $s_3$  when  $H$  tends to 0.

**Remark 3.1** In general cubic splines do not preserve monotonicity between neighbouring nodes. For instance, by approximating the unitary circumference in the first quarter using the points  $(x_k = \sin(k\pi/6), y_k = \cos(k\pi/6))$ , for  $k = 0, \dots, 3$ , we would obtain an oscillatory spline (see Figure 3.14). In these cases, other approximation techniques can be better suited. For instance, the MATLAB command **pchip** provides the Hermite piecewise cubic interpolant which is locally monotone and interpolates the function as well as its first derivative at the nodes  $\{x_i, i = 1, \dots, n-1\}$  (see Figure 3.14). The Hermite interpolant can be obtained by using the following instructions:

```
t = linspace(0,pi/2,4)
x = cos(t); y = sin(t);
```

**pchip**

```
xx = linspace(0,1,40);
plot(x,y,'o',xx,[pchip(x,y,xx);spline(x,y,xx)])
```



**Fig. 3.14.** Approximation of the first quarter of the circumference of the unitary circle using only 4 nodes. The dashed line is the cubic spline, while the continuous line is the piecewise cubic Hermite interpolant



See the Exercises 3.5-3.8.



### 3.4 The least-squares method

As already noticed, a Lagrange interpolation does not guarantee a better approximation of a given function when the polynomial degree gets large. This problem can be overcome by composite interpolation (such as piecewise linear polynomials or splines). However, neither are suitable to extrapolate information from the available data, that is, to generate new values at points lying outside the interval where interpolation nodes are given.

**Example 3.8 (Finance)** On the basis of the data reported in Figure 3.1, we would like to predict whether the stock price will increase or diminish in the coming days. The Lagrange polynomial interpolation is impractical, as it would require a (tremendously oscillatory) polynomial of degree 719 which will provide a completely erroneous prediction. On the other hand, piecewise linear interpolation, whose graph is reported in Figure 3.1, provides extrapolated results by exploiting only the values of the last two days, thus completely neglecting the previous history. To get a better result we should avoid the interpolation requirement, by invoking least-squares approximation as indicated below. ■

Assume that the data  $\{(x_i, y_i), i = 0, \dots, n\}$  are available, where now  $y_i$  could represent the values  $f(x_i)$  attained by a given function  $f$  at the nodes  $x_i$ . For a given integer  $m \geq 1$  (usually,  $m \ll n$ ) we look for a polynomial  $\tilde{f} \in \mathbb{P}_m$  which satisfies the inequality

$$\sum_{i=0}^n [y_i - \tilde{f}(x_i)]^2 \leq \sum_{i=0}^n [y_i - p_m(x_i)]^2 \quad (3.21)$$

for every polynomial  $p_m \in \mathbb{P}_m$ . Should it exist,  $\tilde{f}$  will be called the *least-squares approximation* in  $\mathbb{P}_m$  of the set of data  $\{(x_i, y_i), i = 0, \dots, n\}$ . Unless  $m \geq n$ , in general it will not be possible to guarantee that  $\tilde{f}(x_i) = y_i$  for all  $i = 0, \dots, n$ .

Setting

$$\tilde{f}(x) = a_0 + a_1x + \dots + a_mx^m, \quad (3.22)$$

where the coefficients  $a_0, \dots, a_m$  are unknown, the problem (3.21) can be restated as follows: find  $a_0, a_1, \dots, a_m$  such that

$$\Phi(a_0, a_1, \dots, a_m) = \min_{\{b_i, i=0, \dots, m\}} \Phi(b_0, b_1, \dots, b_m)$$

where

$$\Phi(b_0, b_1, \dots, b_m) = \sum_{i=0}^n [y_i - (b_0 + b_1x_i + \dots + b_mx_i^m)]^2.$$

We solve this problem in the special case when  $m = 1$ . Since

$$\Phi(b_0, b_1) = \sum_{i=0}^n [y_i^2 + b_0^2 + b_1^2x_i^2 + 2b_0b_1x_i - 2b_0y_i - 2b_1x_iy_i],$$

the graph of  $\Phi$  is a convex paraboloid. The point  $(a_0, a_1)$  at which  $\Phi$  attains its minimum satisfies the conditions

$$\frac{\partial \Phi}{\partial b_0}(a_0, a_1) = 0, \quad \frac{\partial \Phi}{\partial b_1}(a_0, a_1) = 0,$$

where the symbol  $\partial \Phi / \partial b_j$  denotes the partial derivative (that is, the rate of variation) of  $\Phi$  with respect to  $b_j$ , after having frozen the remaining variable (see the definition 8.3).

By explicitly computing the two partial derivatives we obtain

$$\sum_{i=0}^n [a_0 + a_1x_i - y_i] = 0, \quad \sum_{i=0}^n [a_0x_i + a_1x_i^2 - x_iy_i] = 0,$$

which is a system of two equations for the two unknowns  $a_0$  and  $a_1$ :

$$\begin{aligned}
a_0(n+1) + a_1 \sum_{i=0}^n x_i &= \sum_{i=0}^n y_i, \\
a_0 \sum_{i=0}^n x_i + a_1 \sum_{i=0}^n x_i^2 &= \sum_{i=0}^n y_i x_i.
\end{aligned} \tag{3.23}$$

Setting  $D = (n+1) \sum_{i=0}^n x_i^2 - (\sum_{i=0}^n x_i)^2$ , the solution reads:

$$\begin{aligned}
a_0 &= \frac{1}{D} \left( \sum_{i=0}^n y_i \sum_{j=0}^n x_j^2 - \sum_{j=0}^n x_j \sum_{i=0}^n x_i y_i \right), \\
a_1 &= \frac{1}{D} \left( (n+1) \sum_{i=0}^n x_i y_i - \sum_{j=0}^n x_j \sum_{i=0}^n y_i \right).
\end{aligned} \tag{3.24}$$

The corresponding polynomial  $\tilde{f}(x) = a_0 + a_1 x$  is known as the *least-squares straight line*, or *regression line*.

The previous approach can be generalized in several ways. The first generalization is to the case of an arbitrary  $m$ . The associated  $(m+1) \times (m+1)$  linear system, which is symmetric, will have the form:

$$\begin{array}{rclcl}
a_0(n+1) + a_1 \sum_{i=0}^n x_i & + \dots + a_m \sum_{i=0}^n x_i^m & = & \sum_{i=0}^n y_i, \\
a_0 \sum_{i=0}^n x_i & + a_1 \sum_{i=0}^n x_i^2 & + \dots + a_m \sum_{i=0}^n x_i^{m+1} & = & \sum_{i=0}^n x_i y_i, \\
\vdots & \vdots & & \vdots & \vdots \\
a_0 \sum_{i=0}^n x_i^m & + a_1 \sum_{i=0}^n x_i^{m+1} & + \dots + a_m \sum_{i=0}^n x_i^{2m} & = & \sum_{i=0}^n x_i^m y_i.
\end{array}$$

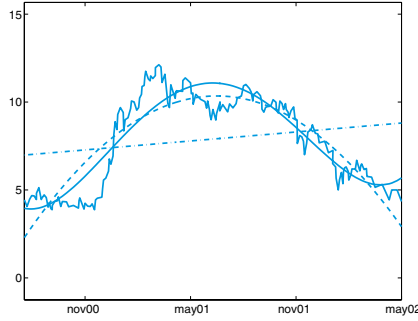
When  $m = n$ , the least-squares polynomial must coincide with the Lagrange interpolating polynomial  $\Pi_n$  (see Exercise 3.9).

The MATLAB command `c=polyfit(x,y,m)` computes by default the coefficients of the polynomial of degree  $m$  which approximates  $n+1$  pairs of data  $(x(i), y(i))$  in the least-squares sense. As already noticed in Section 3.1.1, when  $m$  is equal to  $n$  it returns the interpolating polynomial.

**Example 3.9 (Finance)** In Figure 3.15 we draw the graphs of the least-squares polynomials of degree 1, 2 and 4 that approximate in the least-squares sense the data of Figure 3.1. The polynomial of degree 4 reproduces quite reasonably the behavior of the stock price in the considered time interval and suggests that in the near future the quotation will increase. ■

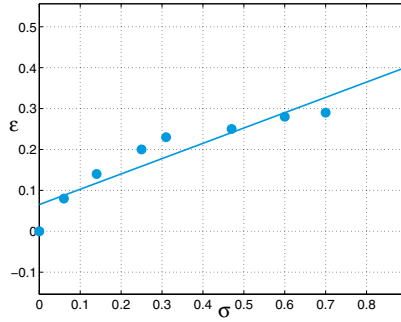
**Example 3.10 (Biomechanics)** Using the least-squares method we can answer the question in Problem 3.3 and discover that the line which better approximates the given data has equation  $\epsilon(\sigma) = 0.3471\sigma + 0.0654$  (see Figure





**Fig. 3.15.** Least-squares approximation of the data of Problem 3.2 of degree 1 (*dashed-dotted line*), degree 2 (*dashed line*) and degree 4 (*thick solid line*). The exact data are represented by the *thin solid line*

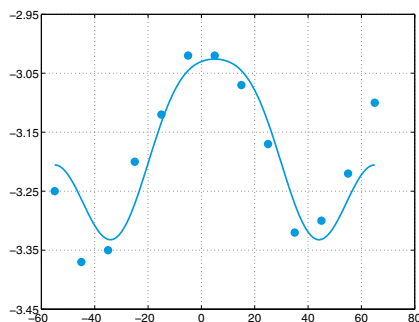
3.16); when  $\sigma = 0.9$  it provides the estimate  $\epsilon = 0.2915$  for the deformation. ■



**Fig. 3.16.** Linear least-squares approximation of the data of Problem 3.3

A further generalization of the least-squares approximation consists of using in (3.21)  $\tilde{f}$  and  $p_m$  that are no-longer polynomials but functions of a space  $V_m$  obtained by linearly combining  $m + 1$  independent functions  $\{\psi_j, j = 0, \dots, m\}$ . Special instances are provided, e.g., by the trigonometric functions  $\psi_j(x) = \cos(\gamma j x)$  (for a given parameter  $\gamma \neq 0$ ), by the exponential functions  $\psi_j(x) = e^{\delta j x}$  (for some  $\delta > 0$ ), or by a suitable set of spline functions.

The choice of the functions  $\{\psi_j\}$  is actually dictated by the conjectured behavior of the law underlying the given data distribution. For instance, in Figure 3.17 we draw the graph of the least-squares approximation of the data of the Example 3.1 computed using the trigonometric functions  $\psi_j(x) = \cos(jt(x))$ ,  $j = 0, \dots, 4$ , with  $t(x) = 120(\pi/2)(x + 55)$ . We assume that the data are periodic with period  $120(\pi/2)$ .



**Fig. 3.17.** The least-squares approximation of the data of the Problem 3.1 using a cosine basis. The exact data are represented by the small circles

The reader can verify that the unknown coefficients of

$$\tilde{f}(x) = \sum_{j=0}^m a_j \psi_j(x),$$

can be obtained by solving the following system (of *normal equations*)

$$\mathbf{B}^T \mathbf{B} \mathbf{a} = \mathbf{B}^T \mathbf{y} \quad (3.25)$$

where  $\mathbf{B}$  is the rectangular matrix  $(n+1) \times (m+1)$  of entries  $b_{ij} = \psi_j(x_i)$ ,  $\mathbf{a}$  is the vector of the unknown coefficients, while  $\mathbf{y}$  is the vector of the data.



### Let us summarize

1. The composite piecewise linear interpolant of a function  $f$  is a piecewise continuous linear function  $\tilde{f}$ , which interpolates  $f$  at a given set of nodes  $\{x_i\}$ . With this approximation we avoid Runge's type phenomena when the number of nodes increases;
2. interpolation by cubic splines allows the approximation of  $f$  by a piecewise cubic function  $\tilde{f}$  which is continuous together with its first and second derivatives;
3. in least-squares approximation we look for an approximant  $\tilde{f}$  which is a polynomial of degree  $m$  (typically,  $m \ll n$ ) that minimizes the mean-square error  $\sum_{i=0}^n [y_i - \tilde{f}(x_i)]^2$ . The same minimization criterion can be applied for a class of functions that are not polynomials.



See the Exercises 3.9-3.14.

### 3.5 What we haven't told you

For a more general introduction to the theory of interpolation and approximation the reader is referred to, e.g., [Dav63], [Mei67] and [Gau97].

Polynomial interpolation can also be used to approximate data and functions in several dimensions. In particular, composite interpolation, based on piecewise linear or spline functions, is well suited when the region  $\Omega$  at hand is partitioned into polygons in 2D (triangles or quadrilaterals) and polyhedra in 3D (tetrahedra or prisms).

A special situation occurs when  $\Omega$  is a rectangle or a parallelepiped in which case the MATLAB commands `interp2`, and `interp3`, respectively, can be used. In both cases it is assumed that we want to represent on a regular, fine lattice (or grid) a function whose values are available on a regular, coarser lattice.

`interp2`

`interp3`

Consider for instance the values of  $f(x, y) = \sin(2\pi x) \cos(2\pi y)$  on a (coarse)  $6 \times 6$  lattice of equispaced nodes on the square  $[0, 1]^2$ ; these values can be obtained using the commands:

```
[x,y]=meshgrid(0:0.2:1,0:0.2:1);
z=sin(2*pi*x).*cos(2*pi*y);
```

By the command `interp2` a cubic spline is first computed on this coarse grid, then evaluated at the nodal points of a finer grid of  $21 \times 21$  equispaced nodes:

```
xi = [0:0.05:1]; yi=[0:0.05:1];
[xf,yf]=meshgrid(xi,yi);
pi3=interp2(x,y,z,xf,yf);
```

The command `meshgrid` transforms the set of the couples  $(x_i(k), y_i(j))$  into two matrices `xf` and `yf` that can be used to evaluate functions of two variables and to plot three dimensional surfaces. The rows of `xf` are copies of the vector `xi`, the columns of `yf` are copies of `yi`. Alternatively to the above procedure we can use the command `griddata`, available also for three-dimensional data (`griddata3`) and for the approximation of  $n$ -dimensional surfaces (`griddatan`).

`meshgrid`

`griddata`

The commands described below are for MATLAB only.

When  $\Omega$  is a two-dimensional domain of arbitrary shape, it can be partitioned into triangles using the graphical interface `pdetool`.

`pdetool`

For a general presentation of spline functions see, e.g., [Die93] and [PBP02]. The MATLAB toolbox `splines` allows one to explore several applications of spline functions. In particular, the `spdemo` command gives the user the possibility to investigate the properties of the most important type of spline functions. Rational splines, i.e. functions which are the ratio of two splines functions, are accessible through the commands `rpmak` and `rsmak`. Special instances are the so-called NURBS splines, which are commonly used in CAGD (*Computer Assisted Geometric Design*).

`spdemo`

`rpmak`

`rsmak`

In the same context of Fourier approximation, we mention the approximation based on *wavelets*. This type of approximation is largely used for image reconstruction and compression and in signal analysis (for an introduction, see [DL92], [Urb02]). A rich family of wavelets (and their applications) can be found in the MATLAB toolbox `wavelet`.

### 3.6 Exercises

**Exercise 3.1** Prove inequality (3.6).

**Exercise 3.2** Provide an upper bound of the Lagrange interpolation error for the following functions:

$$\begin{aligned} f_1(x) &= \cosh(x), \quad f_2(x) = \sinh(x), \quad x_k = -1 + 0.5k, \quad k = 0, \dots, 4, \\ f_3(x) &= \cos(x) + \sin(x), \quad x_k = -\pi/2 + \pi k/4, \quad k = 0, \dots, 4. \end{aligned}$$

**Exercise 3.3** The following data are related to the life expectation of citizens of two European regions:

	1975	1980	1985	1990
Western Europe	72.8	74.2	75.2	76.4
Eastern Europe	70.2	70.2	70.3	71.2

Use the interpolating polynomial of degree 3 to estimate the life expectation in 1970, 1983 and 1988. Then extrapolate a value for the year 1995. It is known that the life expectation in 1970 was 71.8 years for the citizens of the West Europe, and 69.6 for those of the East Europe. Recalling these data, is it possible to estimate the accuracy of life expectation predicted in the 1995?

**Exercise 3.4** The price (in euros) of a magazine has changed as follows:

Nov.87	Dec.88	Nov.90	Jan.93	Jan.95	Jan.96	Nov.96	Nov.00
4.5	5.0	6.0	6.5	7.0	7.5	8.0	8.0

Estimate the price in November 2002 by extrapolating these data.

**Exercise 3.5** Repeat the computations carried out in Exercise 3.3, using now the cubic interpolating spline computed by the function `spline`. Then compare the results obtained with the two approaches.

**Exercise 3.6** In the table below we report the values of the sea water density  $\rho$  (in Kg/m<sup>3</sup>) corresponding to different values of the temperature  $T$  (in degrees Celsius):

$T$	4°	8°	12°	16°	20°
$\rho$	1000.7794	1000.6427	1000.2805	999.7165	998.9700

Compute the associated cubic interpolating spline on 4 subintervals of the temperature interval  $[4, 20]$ . Then compare the results provided by the spline interpolant with the following ones (which correspond to further values of  $T$ ):

$T$	$6^\circ$	$10^\circ$	$14^\circ$	$18^\circ$
$\rho$	1000.74088	1000.4882	1000.0224	999.3650

**Exercise 3.7** The Italian production of citrus fruit has changed as follows:

year	1965	1970	1980	1985	1990	1991
production ( $\times 10^5$ Kg)	17769	24001	25961	34336	29036	33417

Use interpolating cubic splines of different kinds to estimate the production in 1962, 1977 and 1992. Compare these results with the real values: 12380, 27403 and 32059, respectively. Compare the results with those that would be obtained using the Lagrange interpolating polynomial.

**Exercise 3.8** Evaluate the function  $f(x) = \sin(2\pi x)$  at 21 equispaced nodes in the interval  $[-1, 1]$ . Compute the Lagrange interpolating polynomial and the cubic interpolating spline. Compare the graphs of these two functions with that of  $f$  on the given interval. Repeat the same calculation using the following perturbed set of data:  $f(x_i) = \sin(2^* \pi^* x_i) + (-1)^{i+1} 10^{-4}$ , and observe that the Lagrange interpolating polynomial is more sensitive to small perturbations than the cubic spline.

**Exercise 3.9** Verify that if  $m = n$  the least-squares polynomial of a function  $f$  at the nodes  $x_0, \dots, x_n$  coincides with the interpolating polynomial  $\Pi_n f$  at the same nodes.

**Exercise 3.10** Compute the least-squares polynomial of degree 4 that approximates the values of  $K$  reported in the different columns of Table 3.1.

**Exercise 3.11** Repeat the computations carried out in Exercise 3.7 using now a least-squares approximation of degree 3.

**Exercise 3.12** Express the coefficients of system (3.23) in terms of the *average*  $M = \frac{1}{(n+1)} \sum_{i=0}^n x_i$  and the *variance*  $v = \frac{1}{(n+1)} \sum_{i=0}^n (x_i - M)^2$  of the set of data  $\{x_i, i = 0, \dots, n\}$ .

**Exercise 3.13** Verify that the regression line passes through the point whose abscissa is the average of  $\{x_i\}$  and ordinate is the average of  $\{f(x_i)\}$ .

**Exercise 3.14** The following values

flow rate	0	35	0.125	5	0	5	1	0.5	0.125	0
-----------	---	----	-------	---	---	---	---	-----	-------	---

represent the measured values of the blood flow-rate in a cross-section of the carotid artery during a heart beat. The frequency of acquisition of the data is constant and is equal to  $10/T$ , where  $T = 1$  s is the beat period. Represent these data by a continuous function of period equal to  $T$ .



## Numerical differentiation and integration

In this chapter we propose methods for the numerical approximation of derivatives and integrals of functions. Concerning integration, quite often for a generic function it is not possible to find a primitive in an explicit form. Even when a primitive is known, its use might not be easy. This is, e.g., the case of the function  $f(x) = \cos(4x) \cos(3 \sin(x))$ , for which we have

$$\int_0^{\pi} f(x) dx = \pi \left(\frac{3}{2}\right)^4 \sum_{k=0}^{\infty} \frac{(-9/4)^k}{k!(k+4)!};$$

the task of computing an integral is transformed into the equally troublesome one of summing a series. In other circumstances the function that we want to integrate or differentiate could only be known on a set of nodes (for instance, when the latter represent the results of an experimental measurement), exactly as happens in the case of function approximation, which was discussed in Chapter 3.

In all these situations it is necessary to consider numerical methods in order to obtain an approximate value of the quantity of interest, independently of how difficult is the function to integrate or differentiate.

**Problem 4.1 (Hydraulics)** The height  $q(t)$  reached at time  $t$  by a fluid in a straight cylinder of radius  $R = 1$  m with a circular hole of radius  $r = 0.1$  m on the bottom, has been measured every 5 seconds yielding the following values

$t$	0	5	10	15	20
$q(t)$	0.6350	0.5336	0.4410	0.3572	0.2822

We want to compute an approximation of the emptying velocity  $q'(t)$  of the cylinder, then compare it with the one predicted by Torricelli's

law:  $q'(t) = -\gamma(r/R)^2 \sqrt{2gq(t)}$ , where  $g$  is the gravity acceleration and  $\gamma = 0.6$  is a correction factor. For the solution of this problem, see Example 4.1. ■

**Problem 4.2 (Optics)** In order to plan a room for infrared beams we are interested in calculating the energy emitted by a black body (that is, an object capable of irradiating in all the spectrum to the ambient temperature) in the (infrared) spectrum comprised between  $3\mu\text{m}$  and  $14\mu\text{m}$  wavelength. The solution of this problem is obtained by computing the integral

$$E(T) = 2.39 \cdot 10^{-11} \int_{3 \cdot 10^{-4}}^{14 \cdot 10^{-4}} \frac{dx}{x^5 (e^{1.432/(Tx)} - 1)}, \quad (4.1)$$

which is the Planck equation for the energy  $E(T)$ , where  $x$  is the wavelength (in cm) and  $T$  the temperature (in Kelvin) of the black body. For its computation see Exercise 4.17. ■

**Problem 4.3 (Electromagnetism)** Consider an electric wire sphere of arbitrary radius  $r$  and conductivity  $\sigma$ . We want to compute the density distribution of the current  $\mathbf{j}$  as a function of  $r$  and  $t$  (the time), knowing the initial distribution of the current density  $\rho(r)$ . The problem can be solved using the relations between the current density, the electric field and the charge density and observing that, for the symmetry of the problem,  $\mathbf{j}(r, t) = j(r, t)\mathbf{r}/|\mathbf{r}|$ , where  $j = |\mathbf{j}|$ . We obtain

$$j(r, t) = \gamma(r)e^{-\sigma t/\varepsilon_0}, \quad \gamma(r) = \frac{\sigma}{\varepsilon_0 r^2} \int_0^r \rho(\xi) \xi^2 d\xi, \quad (4.2)$$

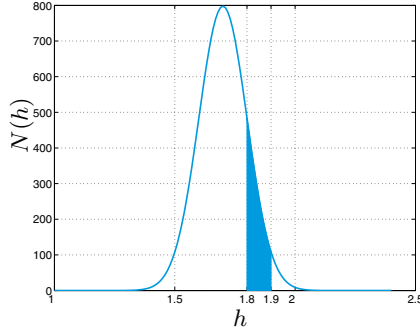
where  $\varepsilon_0 = 8.859 \cdot 10^{-12}$  farad/m is the dielectric constant of the void. For the computation of this integral, see Exercise 4.16. ■

**Problem 4.4 (Demography)** We consider a population of a very large number  $M$  of individuals. The distribution  $N(h)$  of their height can be represented by a "bell" function characterized by the mean value  $\bar{h}$  of the height and the standard deviation  $\sigma$

$$N(h) = \frac{M}{\sigma\sqrt{2\pi}} e^{-(h-\bar{h})^2/(2\sigma^2)}.$$

Then





**Fig. 4.1.** Height distribution of a population of  $M = 200$  individuals

$$N = \int_h^{h+\Delta h} N(h) \, dh \quad (4.3)$$

represents the number of individuals whose height is between  $h$  and  $h + \Delta h$  (for a positive  $\Delta h$ ). An instance is provided in Figure 4.1, which corresponds to the case  $M = 200$ ,  $\bar{h} = 1.7$  m,  $\sigma = 0.1$  m, and the area of the shadowed region gives the number of individuals whose height is in the range  $1.8 \div 1.9$  m. For the solution of this problem see Example 4.2. ■

## 4.1 Approximation of function derivatives

Consider a function  $f : [a, b] \rightarrow \mathbb{R}$  continuously differentiable in  $[a, b]$ . We seek an approximation of the first derivative of  $f$  at a generic point  $\bar{x}$  in  $(a, b)$ .

In view of the definition (1.10), for  $h$  sufficiently small and positive, we can assume that the quantity

$$(\delta_+ f)(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x})}{h} \quad (4.4)$$

is an approximation of  $f'(\bar{x})$  which is called the *forward finite difference*. To estimate the error, it suffices to expand  $f$  in a Taylor series; if  $f \in C^2(a, b)$ , we have

$$f(\bar{x} + h) = f(\bar{x}) + hf'(\bar{x}) + \frac{h^2}{2}f''(\xi), \quad (4.5)$$

where  $\xi$  is a suitable point in the interval  $(\bar{x}, \bar{x} + h)$ . Therefore

$$(\delta_+ f)(\bar{x}) = f'(\bar{x}) + \frac{h}{2} f''(\xi), \quad (4.6)$$

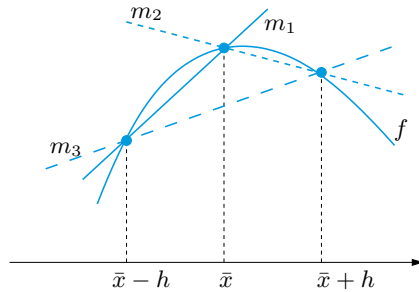
and thus  $(\delta_+ f)(\bar{x})$  provides a first-order approximation to  $f'(\bar{x})$  with respect to  $h$ . Still assuming  $f \in C^2(a, b)$ , with a similar procedure we can derive from the Taylor expansion

$$f(\bar{x} - h) = f(\bar{x}) - hf'(\bar{x}) + \frac{h^2}{2} f''(\eta) \quad (4.7)$$

with  $\eta \in (\bar{x} - h, \bar{x})$ , the *backward finite difference*

$$(\delta_- f)(\bar{x}) = \frac{f(\bar{x}) - f(\bar{x} - h)}{h} \quad (4.8)$$

which is also first-order accurate. Note that formulae (4.4) and (4.8) can also be obtained by differentiating the linear polynomial interpolating  $f$  at the points  $\{\bar{x}, \bar{x} + h\}$  and  $\{\bar{x} - h, \bar{x}\}$ , respectively. In fact, these schemes amount to approximating  $f'(\bar{x})$  by the slope of the straight line passing through the two points  $(\bar{x}, f(\bar{x}))$  and  $(\bar{x} + h, f(\bar{x} + h))$ , or  $(\bar{x} - h, f(\bar{x} - h))$  and  $(\bar{x}, f(\bar{x}))$ , respectively (see Figure 4.2).



**Fig. 4.2.** Finite difference approximation of  $f'(\bar{x})$ : backward (*solid line*), forward (*dotted line*) and centered (*dashed line*).  $m_1 = (\delta_- f)(\bar{x})$ ,  $m_2 = (\delta_+ f)(\bar{x})$  and  $m_3 = (\delta f)(\bar{x})$  denote the slopes of the three straight lines

Finally, we introduce the *centered finite difference* formula

$$(\delta f)(\bar{x}) = \frac{f(\bar{x} + h) - f(\bar{x} - h)}{2h} \quad (4.9)$$

If  $f \in C^3(a, b)$ , this formula provides a second-order approximation to  $f'(\bar{x})$  with respect to  $h$ . Indeed, by expanding  $f(\bar{x} + h)$  and  $f(\bar{x} - h)$  at the third order around  $\bar{x}$  and summing up the two expressions, we obtain

$$f'(\bar{x}) - (\delta f)(\bar{x}) = \frac{h^2}{12}[f'''(\xi) + f'''(\eta)], \quad (4.10)$$

where  $\eta$  and  $\xi$  are suitable points in the intervals  $(\bar{x}-h, \bar{x})$  and  $(\bar{x}, \bar{x}+h)$ , respectively (see Exercise 4.2).

By (4.9)  $f'(\bar{x})$  is approximated by the slope of the straight line passing through the points  $(\bar{x}-h, f(\bar{x}-h))$  and  $(\bar{x}+h, f(\bar{x}+h))$ .

**Example 4.1 (Hydraulics)** Let us solve Problem 4.1, using formulae (4.4), (4.8) and (4.9), with  $h = 5$ , to approximate  $q'(t)$  at five different points. We obtain:

$t$	0	5	10	15	20
$q'(t)$	-0.0212	-0.0194	-0.0176	-0.0159	-0.0141
$\delta_+ q$	-0.0203	-0.0185	-0.0168	-0.0150	--
$\delta_- q$	--	-0.0203	-0.0185	-0.0168	-0.0150
$\delta q$	--	-0.0194	-0.0176	-0.0159	--

The agreement between the exact derivative and the one computed from the finite difference formulae is more satisfactory when using formula (4.9) rather than (4.8) or (4.4). ■

In general, we can assume that the values of  $f$  are available at  $n+1$  equispaced points  $x_i = x_0 + ih$ ,  $i = 0, \dots, n$ , with  $h > 0$ . In this case in the numerical derivation  $f'(x_i)$  can be approximated by taking one of the previous formulae (4.4), (4.8) or (4.9) with  $\bar{x} = x_i$ .

Note that the centered formula (4.9) cannot be used at the extrema  $x_0$  and  $x_n$ . For these nodes we could use the values

$$\begin{aligned} \frac{1}{2h} [-3f(x_0) + 4f(x_1) - f(x_2)] & \quad \text{at } x_0, \\ \frac{1}{2h} [3f(x_n) - 4f(x_{n-1}) + f(x_{n-2})] & \quad \text{at } x_n, \end{aligned} \quad (4.11)$$

which are also second-order accurate with respect to  $h$ . They are obtained by computing at the point  $x_0$  (respectively,  $x_n$ ) the first derivative of the polynomial of degree 2 interpolating  $f$  at the nodes  $x_0, x_1, x_2$  (respectively,  $x_{n-2}, x_{n-1}, x_n$ ).

See Exercises 4.1-4.4.



## 4.2 Numerical integration

In this section we introduce numerical methods suitable for approximating the integral

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

where  $f$  is an arbitrary continuous function in  $[a, b]$ . We start by introducing some simple formulae, which are indeed special instances of the family of Newton-Cotes formulae. Then we will introduce the so-called Gaussian formulae, that feature the highest possible degree of exactness for a given number of evaluations of the function  $f$ .

### 4.2.1 Midpoint formula

A simple procedure to approximate  $I(f)$  can be devised by partitioning the interval  $[a, b]$  into subintervals  $I_k = [x_{k-1}, x_k]$ ,  $k = 1, \dots, M$ , with  $x_k = a + kH$ ,  $k = 0, \dots, M$  and  $H = (b - a)/M$ . Since

$$I(f) = \sum_{k=1}^M \int_{I_k} f(x) dx, \quad (4.12)$$

on each sub-interval  $I_k$  we can approximate the exact integral of  $f$  by that of a polynomial  $\bar{f}$  approximating  $f$  on  $I_k$ . The simplest solution consists in choosing  $\bar{f}$  as the constant polynomial interpolating  $f$  at the middle point of  $I_k$ :

$$\bar{x}_k = \frac{x_{k-1} + x_k}{2}.$$

In such a way we obtain the *composite midpoint quadrature formula*

$$I_{mp}^c(f) = H \sum_{k=1}^M f(\bar{x}_k) \quad (4.13)$$

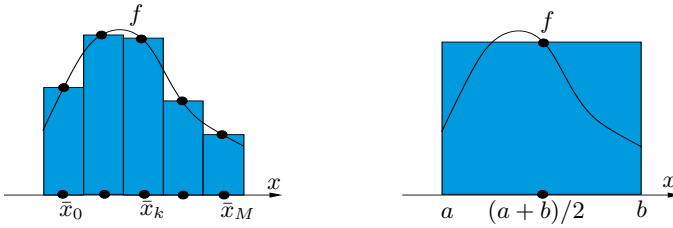
The symbol  $mp$  stands for midpoint, while  $c$  stands for composite. This formula is second-order accurate with respect to  $H$ . More precisely, if  $f$  is continuously differentiable up to its second derivative in  $[a, b]$ , we have

$$I(f) - I_{mp}^c(f) = \frac{b-a}{24} H^2 f''(\xi), \quad (4.14)$$

where  $\xi$  is a suitable point in  $[a, b]$  (see Exercise 4.6). Formula (4.13) is also called the *composite rectangle quadrature formula* because of its geometrical interpretation, which is evident from Figure 4.3. The classical *midpoint formula* (or *rectangle formula*) is obtained by taking  $M = 1$  in (4.13), i.e. using the midpoint rule directly on the interval  $(a, b)$ :

$$I_{mp}(f) = (b-a) f[(a+b)/2] \quad (4.15)$$

The error is now given by



**Fig. 4.3.** The composite midpoint formula (*left*); the midpoint formula (*right*)

$$I(f) - I_{mp}(f) = \frac{(b-a)^3}{24} f''(\xi), \quad (4.16)$$

where  $\xi$  is a suitable point in  $[a, b]$ . Relation (4.16) follows as a special case of (4.14), but it can also be proved directly. Indeed, setting  $\bar{x} = (a+b)/2$ , we have

$$\begin{aligned} I(f) - I_{mp}(f) &= \int_a^b [f(x) - f(\bar{x})] dx \\ &= \int_a^b f'(\bar{x})(x - \bar{x}) dx + \frac{1}{2} \int_a^b f''(\eta(x))(x - \bar{x})^2 dx, \end{aligned}$$

where  $\eta(x)$  is a suitable point in the interval whose endpoints are  $x$  and  $\bar{x}$ . Then (4.16) follows because  $\int_a^b (x - \bar{x}) dx = 0$  and, by the mean value theorem for integrals, there exists  $\xi \in [a, b]$  such that

$$\frac{1}{2} \int_a^b f''(\eta(x))(x - \bar{x})^2 dx = \frac{1}{2} f''(\xi) \int_a^b (x - \bar{x})^2 dx = \frac{(b-a)^3}{24} f''(\xi).$$

The *degree of exactness* of a quadrature formula is the maximum integer  $r \geq 0$  for which the approximate integral (produced by the quadrature formula) of any polynomial of degree  $r$  is equal to the exact integral. We can deduce from (4.14) and (4.16) that the midpoint formula has degree of exactness 1, since it integrates exactly all polynomials of degree less than or equal to 1 (but not all those of degree 2).

The midpoint composite quadrature formula is implemented in Program 4.1. Input parameters are the endpoints of the integration interval **a** and **b**, the number of subintervals **M** and the MATLAB function **f** to define the function  $f$ .

**Program 4.1. midpointc:** composite midpoint quadrature formula

```
function Imp=midpointc(a,b,M,f,varargin)
% MIDPOINTC Composite midpoint numerical integration.
% IMP = MIDPOINTC(A,B,M,FUN) computes an approximation
```



```
% of the integral of the function FUN via the midpoint
% method (with M equispaced intervals). FUN accepts a
% real vector input x and returns a real vector value.
% FUN can also be an inline object.
% IMP=MIDPOINT(A,B,M,FUN,P1,P2,...) calls the function
% FUN passing the optional parameters P1,P2,... as
% FUN(X,P1,P2,...).
H=(b-a)/M;
x = linspace(a+H/2,b-H/2,M);
fmp=feval(f,x,varargin{:})*ones(1,M);
Imp=H*sum(fmp);
return
```



See the Exercises 4.5-4.8.

### 4.2.2 Trapezoidal formula

Another formula can be obtained by replacing  $f$  on  $I_k$  by the linear polynomial interpolating  $f$  at the nodes  $x_{k-1}$  and  $x_k$  (equivalently, replacing  $f$  by  $\Pi_1^H f$ , see Section 3.2, on the whole interval  $(a, b)$ ). This yields

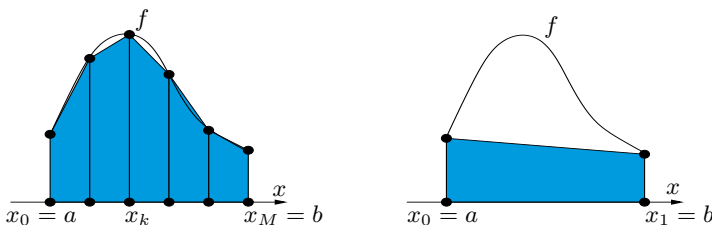
$$\begin{aligned} I_t^c(f) &= \frac{H}{2} \sum_{k=1}^M [f(x_k) + f(x_{k-1})] \\ &= \frac{H}{2} [f(a) + f(b)] + H \sum_{k=1}^{M-1} f(x_k) \end{aligned} \quad (4.17)$$

This formula is called the *composite trapezoidal formula*, and is second-order accurate with respect to  $H$ . In fact, one can obtain the expression

$$I(f) - I_t^c(f) = -\frac{b-a}{12} H^2 f''(\xi) \quad (4.18)$$

for the quadrature error for a suitable point  $\xi \in [a, b]$ , provided that  $f \in C^2([a, b])$ . When (4.17) is used with  $M = 1$ , we obtain

$$I_t(f) = \frac{b-a}{2} [f(a) + f(b)] \quad (4.19)$$



**Fig. 4.4.** Composite trapezoidal formula (*left*); trapezoidal formula (*right*)

which is called the *trapezoidal formula* because of its geometrical interpretation. The error induced is given by

$$I(f) - I_t(f) = -\frac{(b-a)^3}{12} f''(\xi), \quad (4.20)$$

where  $\xi$  is a suitable point in  $[a, b]$ . We can deduce that (4.19) has degree of exactness equal to 1, as is the case of the midpoint rule.

The composite trapezoidal formula (4.17) is implemented in the MATLAB programs `trapz` and `cumtrapz`. If  $\mathbf{x}$  is a vector whose components are the abscissae  $x_k$ ,  $k = 0, \dots, M$  (with  $x_0 = a$  and  $x_M = b$ ), and  $\mathbf{y}$  that of the values  $f(x_k)$ ,  $k = 0, \dots, M$ , `z=cumtrapz(x,y)` returns the vector  $\mathbf{z}$  whose components are  $z_k \simeq \int_a^{x_k} f(x)dx$ , the integral being approximated by the composite trapezoidal rule. Thus `z(M+1)` is an approximation of the integral of  $f$  on  $(a, b)$ .

`trapz`  
`cumtrapz`



See the Exercises 4.9-4.11.

### 4.2.3 Simpson formula

The Simpson formula can be obtained by replacing the integral of  $f$  over each  $I_k$  by that of its interpolating polynomial of degree 2 at the nodes  $x_{k-1}$ ,  $\bar{x}_k = (x_{k-1} + x_k)/2$  and  $x_k$ ,

$$\begin{aligned} \Pi_2 f(x) &= \frac{2(x - \bar{x}_k)(x - x_k)}{H^2} f(x_{k-1}) \\ &+ \frac{4(x_{k-1} - x)(x - x_k)}{H^2} f(\bar{x}_k) + \frac{2(x - \bar{x}_k)(x - x_{k-1})}{H^2} f(x_k). \end{aligned}$$

The resulting formula is called the *composite Simpson quadrature formula*, and reads

$$I_s^c(f) = \frac{H}{6} \sum_{k=1}^M [f(x_{k-1}) + 4f(\bar{x}_k) + f(x_k)] \quad (4.21)$$

One can prove that it induces the error

$$I(f) - I_s^c(f) = -\frac{b-a}{180} \frac{H^4}{16} f^{(4)}(\xi), \quad (4.22)$$

where  $\xi$  is a suitable point in  $[a, b]$ , provided that  $f \in C^4([a, b])$ . It is therefore fourth-order accurate with respect to  $H$ . When (4.21) is applied to only one interval, say  $(a, b)$ , we obtain the so-called *Simpson quadrature formula*

$$I_s(f) = \frac{b-a}{6} [f(a) + 4f((a+b)/2) + f(b)] \quad (4.23)$$

The error is now given by

$$I(f) - I_s(f) = -\frac{1}{16} \frac{(b-a)^5}{180} f^{(4)}(\xi), \quad (4.24)$$

for a suitable  $\xi \in [a, b]$ . Its degree of exactness is therefore equal to 3.

The composite Simpson rule is implemented in Program 4.2.

**Program 4.2. simpsonc:** composite Simpson quadrature formula



```
function [Isic]=simpsonc(a,b,M,f,varargin)
%SIMPSONC Composite Simpson numerical integration.
% ISIC = SIMPSONC(A,B,M,FUN) computes an approximation
% of the integral of the function FUN via the Simpson
% method (using M equispaced intervals). FUN accepts
% real vector input x and returns a real vector value.
% FUN can also be an inline object.
% ISIC = SIMPSONC(A,B,M,FUN,P1,P2,...) calls the
% function FUN passing the optional parameters
% P1,P2,... as FUN(X,P1,P2,...).
H=(b-a)/M;
x=linspace(a,b,M+1);
fpm=feval(f,x,varargin{:}).*ones(1,M+1);
fpm(2:end-1) = 2*fpm(2:end-1);
Isic=H*sum(fpm)/6;
x=linspace(a+H/2,b-H/2,M);
fpm=feval(f,x,varargin{:}).*ones(1,M);
Isic = Isic+2*H*sum(fpm)/3;
return
```

**Example 4.2 (Demography)** Let us consider Problem 4.4. To compute the number of individuals whose height is between 1.8 and 1.9 m, we need to solve the integral (4.3) for  $h = 1.8$  and  $\Delta h = 0.1$ . For that we use the composite Simpson formula with 100 sub-intervals

```
N = inline(['M/(sigma*sqrt(2*pi))*exp(-(h-hbar).^2'...
          './(2*sigma^2))'], 'h', 'M', 'hbar', 'sigma')
```

```
N =
```

```
Inline function:
```

```
N(h,M,hbar,sigma) = M/(sigma * sqrt(2*pi)) * exp(-(h -
hbar).^2./(2*sigma^2))
```

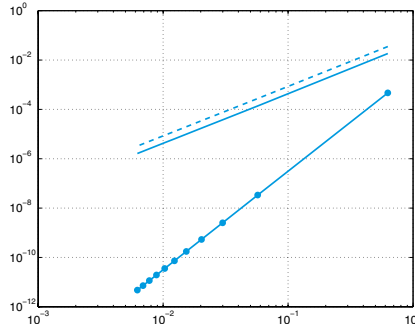
```
M = 200; hbar = 1.7; sigma = 0.1;
int = simpsonc(1.8, 1.9, 100, N, M, hbar, sigma)
```

```
int =
```

```
27.1810
```

We therefore estimate that the number of individuals in this range of height is 27.1810, corresponding to the 15.39 % of all individuals. ■





**Fig. 4.5.** Logarithmic representation of the errors versus  $H$  for Simpson (*solid line with circles*), midpoint (*solid line*) and trapezoidal (*dashed line*) composite quadrature formulae

**Example 4.3** We want to compare the approximations of the integral  $I(f) = \int_0^{2\pi} x e^{-x} \cos(2x) dx = -1/25(10\pi - 3 + 3e^{2\pi})/e^{2\pi} \simeq -0.122122604618968$  obtained by using the composite midpoint, trapezoidal and Simpson formulae. In Figure 4.5 we plot on the logarithmic scale the errors versus  $H$ . As pointed out in Section 1.5, in this type of plot the greater the slope of the curve, the higher the order of convergence of the corresponding formula. As expected from the theoretical results, the midpoint and trapezoidal formulae are second-order accurate, whereas the Simpson formula is fourth-order accurate. ■

### 4.3 Interpolatory quadratures

All (non-composite) quadrature formulae introduced in the previous sections are remarkable instances of a more general quadrature formula of the form:

$$I_{appr}(f) = \sum_{j=0}^n \alpha_j f(y_j) \quad (4.25)$$

The real numbers  $\{\alpha_j\}$  are the *quadrature weights*, while the points  $\{y_j\}$  are the *quadrature nodes*. In general, one requires that (4.25) integrates exactly at least a constant function: this property is ensured if  $\sum_{j=0}^n \alpha_j = b - a$ . We can get a degree of exactness equal to (at least)  $n$  taking

$$I_{appr}(f) = \int_a^b \Pi_n f(x) dx,$$

where  $\Pi_n f \in \mathbb{P}_n$  is the Lagrange interpolating polynomial of the function  $f$  at the nodes  $y_i, i = 0, \dots, n$ , given by (3.4). This yields the following expression for the weights

$$\alpha_i = \int_a^b \varphi_i(x) dx, \quad i = 0, \dots, n,$$

where  $\varphi_i \in \mathbb{P}_n$  is the  $i$ -th characteristic Lagrange polynomial such that  $\varphi_i(y_j) = \delta_{ij}$ , for  $i, j = 0, \dots, n$ , that was introduced in (3.3).

**Example 4.4** For the trapezoidal formula (4.19) we have  $n = 1$ ,  $y_0 = a$ ,  $y_1 = b$  and

$$\begin{aligned} \alpha_0 &= \int_a^b \varphi_0(x) dx = \int_a^b \frac{x-b}{a-b} dx = \frac{b-a}{2}, \\ \alpha_1 &= \int_a^b \varphi_1(x) dx = \int_a^b \frac{x-a}{b-a} dx = \frac{b-a}{2}. \end{aligned}$$

■

The question that arises is whether suitable choices of the nodes exist such that the degree of exactness is greater than  $n$ , more precisely, equal to  $r = n + m$  for some  $m > 0$ . We can simplify our discussion by restricting ourselves to a reference interval, say  $(-1, 1)$ . Indeed, once a set of quadrature nodes  $\{\bar{y}_j\}$  and weights  $\{\bar{\alpha}_j\}$  are available on  $[-1, 1]$ , then owing to the change of variable (3.8) we can immediately obtain the corresponding nodes and weights,

$$y_j = \frac{a+b}{2} + \frac{b-a}{2} \bar{y}_j, \quad \alpha_j = \frac{b-a}{2} \bar{\alpha}_j$$

on an arbitrary integration interval  $[a, b]$ .

The answer to the previous question is furnished by the following result (see, [QSS06, Chapter 10]):

**Proposition 4.1** *For a given  $m > 0$ , the quadrature formula  $\sum_{j=0}^n \bar{\alpha}_j f(\bar{y}_j)$  has degree of exactness  $n + m$  iff it is of interpolatory type and the nodal polynomial  $\omega_{n+1} = \prod_{i=0}^n (x - \bar{y}_i)$  associated with the nodes  $\{\bar{y}_i\}$  is such that*

$$\int_{-1}^1 \omega_{n+1}(x) p(x) dx = 0, \quad \forall p \in \mathbb{P}_{m-1}. \quad (4.26)$$

The maximum value that  $m$  can take is  $n + 1$  and is achieved provided  $\omega_{n+1}$  is proportional to the so-called Legendre polynomial of degree  $n + 1$ ,  $L_{n+1}(x)$ . The Legendre polynomials can be computed recursively, through the following three-term relation

$n$	$\{\bar{y}_j\}$	$\{\bar{\alpha}_j\}$
1	$\{\pm 1/\sqrt{3}\}$	$\{1\}$
2	$\{\pm\sqrt{15}/5, 0\}$	$\{5/9, 8/9\}$
3	$\left\{\pm(1/35)\sqrt{525 - 70\sqrt{30}}, \right.$ $\left.\pm(1/35)\sqrt{525 + 70\sqrt{30}}\right\}$	$\{(1/36)(18 + \sqrt{30}),$ $(1/36)(18 - \sqrt{30})\}$
4	$\left\{0, \pm(1/21)\sqrt{245 - 14\sqrt{70}} \right.$ $\left.\pm(1/21)\sqrt{245 + 14\sqrt{70}}\right\}$	$\{128/225, (1/900)(322 + 13\sqrt{70})$ $(1/900)(322 - 13\sqrt{70})\}$

**Table 4.1.** Nodes and weights for some quadrature formulae of Gauss-Legendre on the interval  $(-1, 1)$ . Weights corresponding to symmetric couples of nodes are reported only once

$$L_0(x) = 1, \quad L_1(x) = x,$$

$$L_{k+1}(x) = \frac{2k+1}{k+1}xL_k(x) - \frac{k}{k+1}L_{k-1}(x), \quad k = 1, 2, \dots$$

For every  $n = 0, 1, \dots$ , every polynomial in  $\mathbb{P}_n$  can be obtained by a linear combination of the polynomials  $L_0, L_1, \dots, L_n$ . Moreover,  $L_{n+1}$  is orthogonal to all the polynomials of degree less than or equal to  $n$ , i.e.,  $\int_{-1}^1 L_{n+1}(x)L_j(x)dx = 0$  for all  $j = 0, \dots, n$ . This explains why (4.26) is true with  $m$  less than or equal to  $n+1$ .

The maximum degree of exactness is therefore equal to  $2n+1$ , and is obtained for the so-called *Gauss-Legendre formula* ( $I_{GL}$  in short), whose nodes and weights are given by:

$$\begin{cases} \bar{y}_j = \text{zeros of } L_{n+1}(x), \\ \bar{\alpha}_j = \frac{2}{(1 - \bar{y}_j^2)[L'_{n+1}(\bar{y}_j)]^2}, \end{cases} \quad j = 0, \dots, n. \quad (4.27)$$

The weights  $\bar{\alpha}_j$  are all positive and the nodes are internal to the interval  $(-1, 1)$ . In Table 4.1 we report nodes and weights for the Gauss-Legendre quadrature formulae with  $n = 1, 2, 3, 4$ . If  $f \in C^{(2n+2)}([-1, 1])$ , the corresponding error is

$$I(f) - I_{GL}(f) = \frac{2^{2n+3}((n+1)!)^4}{(2n+3)((2n+2)!)^3} f^{(2n+2)}(\xi),$$

where  $\xi$  is a suitable point in  $(-1, 1)$ .

It is often useful to include also the endpoints of the interval among the quadrature nodes. By doing so, the Gauss formula with the highest degree of exactness  $(2n-1)$  is the one that employs the so-called *Gauss-Legendre-Lobatto* nodes (briefly, GLL): for  $n \geq 1$

$$\bar{y}_0 = -1, \quad \bar{y}_n = 1, \quad \bar{y}_j = \text{zeros of } L'_n(x), \quad j = 1, \dots, n-1, \quad (4.28)$$

$n$	$\{\bar{y}_j\}$	$\{\bar{\alpha}_j\}$
1	$\{\pm 1\}$	$\{1\}$
2	$\{\pm 1, 0\}$	$\{1/3, 4/3\}$
3	$\{\pm 1, \pm \sqrt{5}/5\}$	$\{1/6, 5/6\}$
4	$\{\pm 1, \pm \sqrt{21}/7, 0\}$	$\{1/10, 49/90, 32/45\}$

**Table 4.2.** Nodes and weights for some quadrature formulae of Gauss-Legendre-Lobatto on the interval  $(-1, 1)$ . Weights corresponding to symmetric couples of nodes are reported only once

$$\bar{\alpha}_j = \frac{2}{n(n+1)} \frac{1}{[L_n(\bar{y}_j)]^2}, \quad j = 0, \dots, n.$$

If  $f \in C^{(2n)}([-1, 1])$ , the corresponding error is given by

$$I(f) - I_{GLL}(f) = -\frac{(n+1)n^3 2^{2n+1} ((n-1)!)^4}{(2n+1)((2n)!)^3} f^{(2n)}(\xi),$$

for a suitable  $\xi \in (-1, 1)$ . In Table 4.2 we give a table of nodes and weights on the reference interval  $(-1, 1)$  for  $n = 1, 2, 3, 4$ . (For  $n = 1$  we recover the trapezoidal rule.)

**quadl**

Using the MATLAB instruction `quadl(fun,a,b)` it is possible to compute an integral with a composite Gauss-Legendre-Lobatto quadrature formula. The function `fun` can be an inline object. For instance, to integrate  $f(x) = 1/x$  over  $[1, 2]$ , we must first define the function

```
fun=inline('1./x','x');
```

then call `quadl(fun,1,2)`. Note that in the definition of function  $f$  we have used an element by element operation (indeed MATLAB will evaluate this expression component by component on the vector of quadrature nodes).

The specification of the number of subintervals is not requested as it is automatically computed in order to ensure that the quadrature error is below the default tolerance of  $10^{-3}$ . A different tolerance can be provided by the user through the extended command `quadl(fun,a,b,tol)`. In Section 4.4 we will introduce a method to estimate the quadrature error and, consequently, to change  $H$  adaptively.



## Let us summarize

1. A quadrature formula is a formula to approximate the integral of continuous functions on an interval  $[a, b]$ ;
2. it is generally expressed as a linear combination of the values of the function at specific points (called *nodes*) with coefficients which are called *weights*;

3. the *degree of exactness* of a quadrature formula is the highest degree of the polynomials which are integrated exactly by the formula. It is one for the midpoint and trapezoidal rules, three for the Simpson rule,  $2n + 1$  for the Gauss-Legendre formula using  $n + 1$  quadrature nodes, and  $2n - 1$  for the Gauss-Legendre-Lobatto formula using  $n + 1$  nodes;
4. the *order of accuracy* of a composite quadrature formula is its order with respect to the size  $H$  of the subintervals. The order of accuracy is two for composite midpoint and trapezoidal formulae, four for composite Simpson formula.

See the Exercises 4.12-4.18.



## 4.4 Simpson adaptive formula



The integration step-length  $H$  of a quadrature composite formula can be chosen in order to ensure that the quadrature error is less than a prescribed tolerance  $\varepsilon > 0$ . For instance, when using the Simpson composite formula, thanks to (4.22) this goal can be achieved if

$$\frac{b-a}{180} \frac{H^4}{16} \max_{x \in [a,b]} |f^{(4)}(x)| < \varepsilon, \quad (4.29)$$

where  $f^{(4)}$  denotes the fourth-order derivative of  $f$ . Unfortunately, when the absolute value of  $f^{(4)}$  is large only in a small part of the integration interval, the maximum  $H$  for which (4.29) holds true can be too small. The goal of the adaptive Simpson quadrature formula is to yield an approximation of  $I(f)$  within a fixed tolerance  $\varepsilon$  by a *nonuniform* distribution of the integration step-sizes in the interval  $[a, b]$ . In such a way we retain the same accuracy of the composite Simpson rule, but with a lower number of quadrature nodes and, consequently, a reduced number of evaluations of  $f$ .

To this end, we must find an error estimator and an automatic procedure to modify the integration step-length  $H$ , according to the achievement of the prescribed tolerance. We start by analyzing this procedure, which is independent of the specific quadrature formula that one wants to apply.

In the first step of the adaptive procedure, we compute an approximation  $I_s(f)$  of  $I(f) = \int_a^b f(x)dx$ . We set  $H = b - a$  and we try to estimate the quadrature error. If the error is less than the prescribed tolerance, the adaptive procedure is stopped; otherwise the step-size  $H$  is halved until the integral  $\int_a^{a+H} f(x)dx$  is computed with the prescribed accuracy. When the test is passed, we consider the interval  $(a + H, b)$

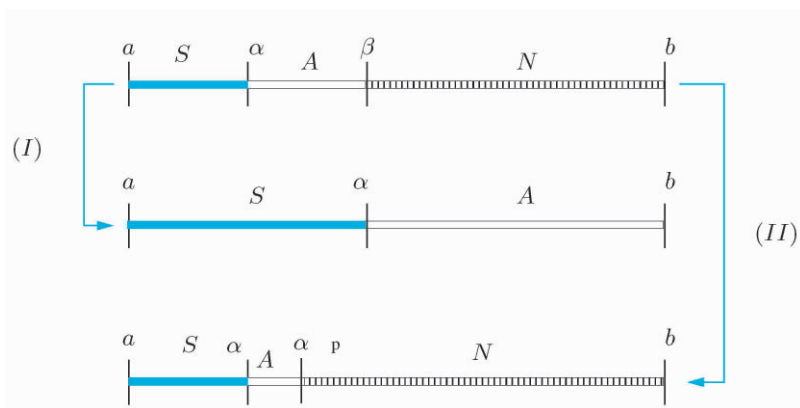
and we repeat the previous procedure, choosing as the first step-size the length  $b - (a + H)$  of that interval.

We use the following notations:

1.  $A$ : the *active* integration interval, i.e. the interval where the integral is being computed;
2.  $S$ : the integration interval already examined, for which the error is less than the prescribed tolerance;
3.  $N$ : the integration interval yet to be examined.

At the beginning of the integration process we have  $N = [a, b]$ ,  $A = N$  and  $S = \emptyset$ , while the situation at the generic step of the algorithm is depicted in Figure 4.6. Let  $J_S(f)$  indicate the computed approximation of  $\int_a^\alpha f(x)dx$ , with  $J_S(f) = 0$  at the beginning of the process; if the algorithm successfully terminates,  $J_S(f)$  yields the desired approximation of  $I(f)$ . We also denote by  $J_{(\alpha,\beta)}(f)$  the approximate integral of  $f$  over the active interval  $[\alpha, \beta]$ . This interval is drawn in gray in Figure 4.6. The generic step of the adaptive integration method is organized as follows:

1. if the estimation of the error ensures that the prescribed tolerance is satisfied, then:
  - (i)  $J_S(f)$  is increased by  $J_{(\alpha,\beta)}(f)$ , that is  $J_S(f) \leftarrow J_S(f) + J_{(\alpha,\beta)}(f)$ ;
  - (ii) we let  $S \leftarrow S \cup A$ ,  $A = N$  (corresponding to the path (I) in Figure 4.6) and  $\alpha \leftarrow \beta$  and  $\beta \leftarrow b$ ;
2. if the estimation of the error fails the prescribed tolerance, then:
  - (j)  $A$  is halved, and the new active interval is set to  $A = [\alpha, \alpha']$  with  $\alpha' = (\alpha + \beta)/2$  (corresponding to the path (II) in Figure 4.6);
  - (jj) we let  $N \leftarrow N \cup [\alpha', \beta]$ ,  $\beta \leftarrow \alpha'$ ;
  - (jjj) a new error estimate is provided.



**Fig. 4.6.** Distribution of the integration intervals at the generic step of the adaptive algorithm and updating of the integration grid

Of course, in order to prevent the algorithm from generating too small step-sizes, it is convenient to monitor the width of  $A$  and warn the user, in case of an excessive reduction of the step-length, about the presence of a possible singularity in the integrand function.

The problem now is to find a suitable estimator of the error. To this end, it is convenient to restrict our attention to a generic subinterval  $[\alpha, \beta]$  in which we compute  $I_s(f)$ : of course, if on this interval the error is less than  $\varepsilon(\beta - \alpha)/(b - a)$ , then the error on the interval  $[a, b]$  will be less than the prescribed tolerance  $\varepsilon$ . Since from (4.24) we get

$$E_s(f; \alpha, \beta) = \int_{\alpha}^{\beta} f(x) dx - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi),$$

to ensure the achievement of the tolerance, it will be sufficient to verify that  $E_s(f; \alpha, \beta) < \varepsilon(\beta - \alpha)/(b - a)$ . In practical computation, this procedure is not feasible since the point  $\xi \in [\alpha, \beta]$  is unknown.

To estimate the error  $E_s(f; \alpha, \beta)$  without using explicitly the value  $f^{(4)}(\xi)$ , we employ again the composite Simpson formula to compute  $\int_{\alpha}^{\beta} f(x) dx$ , but with a step-length  $(\beta - \alpha)/2$ . From (4.22) with  $a = \alpha$  and  $b = \beta$ , we deduce that

$$\int_{\alpha}^{\beta} f(x) dx - I_s^c(f) = -\frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta), \quad (4.30)$$

where  $\eta$  is a suitable point different from  $\xi$ . Subtracting the last two equations, we get

$$\Delta I = I_s^c(f) - I_s(f) = -\frac{(\beta - \alpha)^5}{2880} f^{(4)}(\xi) + \frac{(\beta - \alpha)^5}{46080} f^{(4)}(\eta). \quad (4.31)$$


Let us now make the assumption that  $f^{(4)}(x)$  is approximately a constant on the interval  $[\alpha, \beta]$ . In this case  $f^{(4)}(\xi) \simeq f^{(4)}(\eta)$ . We can compute  $f^{(4)}(\eta)$  from (4.31) and, putting this value in the equation (4.30), we obtain the following estimation of the error:

$$\int_{\alpha}^{\beta} f(x) dx - I_s(f) \simeq \frac{1}{15} \Delta I.$$

The step-length  $(\beta - \alpha)/2$  (that is the step-length employed to compute  $I_s^c(f)$ ) will be accepted if  $|\Delta I|/15 < \varepsilon(\beta - \alpha)/[2(b - a)]$ . The quadrature formula that uses this criterion in the adaptive procedure described previously, is called *adaptive Simpson formula*. It is implemented in Program 4.3. Among the input parameters, **f** is the string in which the function  $f$  is defined, **a** and **b** are the endpoints of the integration interval,

`tol` is the prescribed tolerance on the error and `hmin` is the minimum admissible value for the integration step-length (in order to ensure that the adaptation procedure always terminates).

**Program 4.3. `simpadpt`:** adaptive Simpson formula



```
function [JSf,nodes]=simpadpt(f,a,b,tol,hmin,varargin)
%SIMPADPT Numerically evaluate integral, adaptive
% Simpson quadrature.
%
% JSF = SIMPADPT(FUN,A,B,TOL,HMIN) tries to approximate
% the integral of function FUN from A to B to within an
% error of TOL using recursive adaptive Simpson
% quadrature. The inline function Y = FUN(V) should
% accept a vector argument V and return a vector result
% Y, the integrand evaluated at each element of X.
% JSF = SIMPADPT(FUN,A,B,TOL,HMIN,P1,P2,...) calls the
% function FUN passing the optional parameters
% P1,P2,... as FUN(X,P1,P2,...).
% [JSF,NODES] = SIMPADPT(...) returns the distribution
% of nodes used in the quadrature process.
A=[a,b]; N=[]; S=[]; JSf = 0; ba = b - a; nodes=[];
while ~isempty(A),
    [deltaI,ISc]=caldeltai(A,f,varargin{:});
    if abs(deltaI) <= 15*tol*(A(2)-A(1))/ba;
        JSf = JSf + ISc;      S = union(S,A);
        nodes = [nodes, A(1) (A(1)+A(2))*0.5 A(2)];
        S = [S(1), S(end)]; A = N; N = [];
    elseif A(2)-A(1) < hmin
        JSf=JSf+ISc;          S = union(S,A);
        S = [S(1), S(end)]; A=N; N=[];
        warning('Too small integration-step');
    else
        Am = (A(1)+A(2))*0.5;
        A = [A(1) Am];
        N = [Am, b];
    end
end
nodes=unique(nodes);
return

function [deltaI,ISc]=caldeltai(A,f,varargin)
L=A(2)-A(1);
t=[0; 0.25; 0.5; 0.5; 0.75; 1];
x=L*t+A(1);
L=L/6;
w=[1; 4; 4; 1];
fx=feval(f,x,varargin{:}).*ones(6,1);
IS=L*sum(fx([1 3 6]).*w);
ISc=0.5*L*sum(fx.*[w;w]);
deltaI=IS-ISc;
return
```

**Example 4.5** Let us compute the integral  $I(f) = \int_{-1}^1 e^{-10(x-1)^2} dx$  by using the adaptive Simpson formula. Using Program 4.3 with

```
>> fun=inline('exp(-10*(x-1).^2)'); tol = 1.e-04; hmin = 1.e-03;
```



we find the approximate value 0.28024765884708, instead of the exact value 0.28024956081990. The error is less than the prescribed tolerance `tol=10-5`.

To obtain this result it was sufficient to use only 10 nonuniform subintervals. Note that the corresponding composite formula with uniform step-size would have required 22 subintervals to ensure the same accuracy. ■

## 4.5 What we haven't told you

The midpoint, trapezoidal and Simpson formulae are particular cases of a larger family of quadrature rules known as *Newton-Cotes formulae*. For an introduction, see [QSS06, Chapter 10]. Similarly, the Gauss-Legendre and the Gauss-Legendre-Lobatto formulae that we have introduced in Section 4.3 are special cases of a more general family of Gaussian quadrature formulae. These are *optimal* in the sense that they maximize the degree of exactness for a prescribed number of quadrature nodes. For an introduction to Gaussian formulae, see [QSS06, Chapter 10] or [RR85]. Further developments on numerical integration can be found, e.g., in [DR75] and [PdDKÜK83].

Numerical integration can also be used to compute integrals on unbounded intervals. For instance, to approximate  $\int_0^\infty f(x)dx$ , a first possibility is to find a point  $\alpha$  such that the value of  $\int_\alpha^\infty f(x)dx$  can be neglected with respect to that of  $\int_0^\alpha f(x)dx$ . Then we compute by a quadrature formula this latter integral on a bounded interval. A second possibility is to resort to Gaussian quadrature formulae for unbounded intervals (see [QSS06, Chapter 10]).

Finally, numerical integration can also be used to compute multidimensional integrals. In particular, we mention the MATLAB instruction `dblquad('f',xmin,xmax,ymin,ymax)` by which it is possible to compute the integral of a function contained in the MATLAB file `f.m` over the rectangular domain  $[xmin,xmax] \times [ymin,ymax]$ . Note that the function `f` must have at least two input parameters corresponding to the variables `x` and `y` with respect to which the integral is computed.

`dblquad`

**Octave 4.1** In Octave, `dblquad` is not available; however there are some Octave functions featuring the same functionalities:

1. `quad2dg` for two-dimensional integration, which uses a Gaussian quadrature integration scheme;
2. `quad2dc` for two-dimensional integration, which uses a Gaussian-Chebyshev quadrature integration scheme.

`quad2dg`

`quad2dc`

■

## 4.6 Exercises

**Exercise 4.1** Verify that, if  $f \in C^3$  in a neighborhood  $I_0$  of  $x_0$  (respectively,  $I_n$  of  $x_n$ ) the error of formula (4.11) is equal to  $-\frac{1}{3}f'''(\xi_0)h^2$  (respectively,  $-\frac{1}{3}f'''(\xi_n)h^2$ ), where  $\xi_0$  and  $\xi_n$  are two suitable points belonging to  $I_0$  and  $I_n$ , respectively.

**Exercise 4.2** Verify that if  $f \in C^3$  in a neighborhood of  $\bar{x}$  the error of the formula (4.9) is equal to (4.10).

**Exercise 4.3** Compute the order of accuracy with respect to  $h$  of the following formulae for the numerical approximation of  $f'(x_i)$ :

- a. 
$$\frac{-11f(x_i) + 18f(x_{i+1}) - 9f(x_{i+2}) + 2f(x_{i+3})}{6h},$$
- b. 
$$\frac{f(x_{i-2}) - 6f(x_{i-1}) + 3f(x_i) + 2f(x_{i+1})}{6h},$$
- c. 
$$\frac{-f(x_{i-2}) - 12f(x_i) + 16f(x_{i+1}) - 3f(x_{i+2})}{12h}.$$

**Exercise 4.4 (Demography)** The following values represent the time evolution of the number  $n(t)$  of individuals of a given population whose birth rate is constant ( $b = 2$ ) and mortality rate is  $d(t) = 0.01n(t)$ :

$t$ (months)	0	0.5	1	1.5	2	2.5	3
$n$	100	147	178	192	197	199	200

Use this data to approximate as accurately as possible the rate of variation of this population. Then compare the obtained results with the exact rate  $n'(t) = 2n(t) - 0.01n^2(t)$ .

**Exercise 4.5** Find the minimum number  $M$  of subintervals to approximate with an absolute error less than  $10^{-4}$  the integrals of the following functions:

$$\begin{aligned} f_1(x) &= \frac{1}{1 + (x - \pi)^2} \quad \text{in } [0, 5], \\ f_2(x) &= e^x \cos(x) \quad \text{in } [0, \pi], \\ f_3(x) &= \sqrt{x(1-x)} \quad \text{in } [0, 1], \end{aligned}$$

using the composite midpoint formula. Verify the results obtained using the Program 4.1.

**Exercise 4.6** Prove (4.14) starting from (4.16).

**Exercise 4.7** Why does the midpoint formula lose one order of convergence when used in its composite mode?

**Exercise 4.8** Verify that, if  $f$  is a polynomial of degree less than or equal 1, then  $I_{mp}(f) = I(f)$  i.e. the midpoint formula has degree of exactness equal to 1.

**Exercise 4.9** For the function  $f_1$  in Exercise 4.5, compute (numerically) the values of  $M$  which ensure that the quadrature error is less than  $10^{-4}$  when the integral is approximated by the composite trapezoidal and Gauss quadrature formulae.

**Exercise 4.10** Let  $I_1$  and  $I_2$  be two values obtained by the composite trapezoidal formula applied with two different step-lengths,  $H_1$  and  $H_2$ , for the approximation of  $I(f) = \int_a^b f(x)dx$ . Verify that, if  $f^{(2)}$  has a mild variation on  $(a, b)$ , the value

$$I_R = I_1 + (I_1 - I_2)/(H_2^2/H_1^2 - 1) \quad (4.32)$$

is a better approximation of  $I(f)$  than  $I_1$  and  $I_2$ . This strategy is called the *Richardson extrapolation method*. Derive (4.32) from (4.18).

**Exercise 4.11** Verify that, among all formulae of the form  $I_{app}(f) = \alpha f(\bar{x}) + \beta f(\bar{z})$  where  $\bar{x}, \bar{z} \in [a, b]$  are two unknown nodes and  $\alpha$  and  $\beta$  two undetermined weights, the Gauss formula with  $n = 1$  of Table 4.1 features the maximum degree of exactness.

**Exercise 4.12** For the first two functions of Exercise 4.5, compute the minimum number of intervals such that the quadrature error of the composite Simpson quadrature formula is less than  $10^{-4}$ .

**Exercise 4.13** Compute  $\int_0^2 e^{-x^2/2} dx$  using the Simpson formula (4.23) and the Gauss-Legendre formula of Table 4.1 for  $n = 1$ , then compare the obtained results.

**Exercise 4.14** To compute the integrals  $I_k = \int_0^1 x^k e^{x-1} dx$  for  $k = 1, 2, \dots$ , one can use the following recursive formula:  $I_k = 1 - kI_{k-1}$ , with  $I_1 = 1/e$ . Compute  $I_{20}$  using the composite Simpson formula in order to ensure that the quadrature error is less than  $10^{-3}$ . Compare the Simpson approximation with the result obtained using the above recursive formula.

**Exercise 4.15** Apply the Richardson extrapolation formula (4.32) for the approximation of the integral  $I(f) = \int_0^2 e^{-x^2/2} dx$ , with  $H_1 = 1$  and  $H_2 = 0.5$  using first the Simpson formula (4.23), then the Gauss-Legendre formula for  $n = 1$  of Table 4.1. Verify that in both cases  $I_R$  is more accurate than  $I_1$  and  $I_2$ .

**Exercise 4.16 (Electromagnetism)** Compute using the composite Simpson formula the function  $j(r)$  defined in (4.2) for  $r = k/10$  m with  $k = 1, \dots, 10$ , with  $\rho(\xi) = e^\xi$  and  $\sigma = 0.36$  W/(mK). Ensure that the quadrature error is less than  $10^{-10}$ . (Recall that: m=meters, W=watts, K=degrees Kelvin.)

**Exercise 4.17 (Optics)** By using the composite Simpson and Gauss-Legendre with  $n = 1$  formulae compute the function  $E(T)$ , defined in (4.1), for  $T$  equal to 213 K, up to at least 10 exact significant digits.

**Exercise 4.18** Develop a strategy to compute  $I(f) = \int_0^1 |x^2 - 0.25| dx$  by the composite Simpson formula such that the quadrature error is less than  $10^{-2}$ .