Chapter 2

Interpolation and approximation

2.1 Interpolation 27

2.1.1 Vandermonde matrix 28

2.1.2 Lagrange interpolation formula 28

2.1.3 Gregory–Newton interpolation 29

2.1.4 Interpolation error 34

2.1.5 Interpolation at Chebyshev nodes 36

2.1.6 Hermite interpolation 39

2.1.7 Piecewise interpolation 40

2.2 Approximation 41

2.2.1 Least squares approximation of data points 42

2.2.2 Mean square approximation of functions 43

2.2.3 Orthogonal polynomials 45

2.2.4 Orthogonal polynomials and numerical integration: an introduction . . . 47

2.3 Exercises 49

2.4 Discussion and bibliography 55

Introduction

In this chapter, we study numerical methods for interpolating and approximating functions. The Cambridge dictionary defines interpolation as *the addition of something different in the middle of a text, piece of music, etc. or the thing that is added*. The concept of interpolation in mathematics is consistent with this definition; interpolation consists in finding, given a set of points (x_i, y_i), a function f in a finite-dimensional space that goes through these points. Throughout this course, you use the `plot` function in Julia, which performs piecewise linear interpolation for drawing functions, but there are a number of other standard interpolation methods. Our first goal in this chapter is to present an overview of these methods and the associated error estimates.

In the second part of this chapter, we focus on *function approximation*, which is closely related to the subject of mathematical interpolation. Indeed, a simple manner for approximating a general function by another one in a finite-dimensional space is to select a set of real numbers on the x axis, called *nodes*, and find the associated interpolant. As we shall demonstrate, not all sets of interpolation nodes are equal, and special care is required in order to avoid undesired oscillations. The field of function approximation is vast, so our aim in this chapter is to present only an introduction to the subject. In order to quantify the quality of an approximation, a metric on the space of functions, or a subset thereof, must be specified in order to measure errors. Without a metric, saying that two functions are close is almost meaningless!

2.1 Interpolation

Assume that we are given $n+1$ nodes $x_0, \ldots, x_n$ on the x axis, together with values $u_0, \ldots, u_n$, which may be the values taken by an unknown function $u(x)$ when evaluated at these points. Suppose that we are looking for an interpolation $\widehat{u}(x)$ in a subspace $\mathrm{Span}\{\varphi_0, \ldots, \varphi_n\}$ of the vector space of continuous functions, i.e. an interpolating function of the form

$$\widehat{u}(x) = \alpha_0 \varphi_0(x) + \cdots + \alpha_n \varphi_n(x),$$

where $\alpha_0, \ldots, \alpha_n$ are real coefficients. In order for $\widehat{u}(x)$ to be an interpolating function, we must require that

$$\forall i \in \{0, \ldots, n\}, \qquad \widehat{u}(x_i) = u_i.$$

This leads to a linear system of $n+1$ equations and $n+1$ unknowns, the latter being the coefficients $\alpha_0, \ldots, \alpha_n$. This system of equations in matrix form reads

$$\begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_n(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.1}$$

2.1.1 Vandermonde matrix

Since polynomials are very convenient for evaluation, integration, and differentiation, they are a natural choice for interpolation purposes. The simplest basis of the subspace of polynomials of degree less than or equal to n is given by the monomials:

$$\varphi_0(x) = 1, \qquad \varphi_1(x) = x, \qquad \ldots, \qquad \varphi_n(x) = x^n.$$

In this case, the linear system (2.1) for determining the coefficients of the interpolant reads

$$\begin{pmatrix} 1 & x_0 & \ldots & x_0^n \\ 1 & x_1 & \ldots & x_1^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.2}$$

The matrix on the left-hand side is called a *Vandermonde* matrix. If the abcissae $x_0, \ldots, x_n$ are distinct, then this is a full rank matrix, and so (2.2) admits a unique solution, implying as a corollary that the interpolating polynomial exists and is unique. It is possible to show that the condition number of the Vandermonde increases dramatically with n. Consequently, solving (2.2) is not a viable method in practice for calculating the interpolating polynomial.

2.1.2 Lagrange interpolation formula

One may wonder whether polynomial basis functions $\varphi_0, \ldots, \varphi_n$ can be defined in such a manner that the matrix in (2.1) is the identity matrix. The answer to this question is positive; it suffices to take as a basis the *Lagrange polynomials*, which are given by

$$\varphi_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}.$$

It is simple to check that

$$\varphi_i(x_j) = \delta_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Finding the interpolant in this basis is immediate:

$$\widehat{u}(x) = u_0\varphi_0(x) + \cdots + u_n\varphi_n(x).$$

While simple, this approach to polynomial interpolation has a few disadvantages:

- First, evaluating $\widehat{u}(x)$ is computationally costly when n is large.
- Second, all the basis functions change when adding new interpolation nodes.
- Finally, Lagrange interpolation is numerically unstable because of cancellations between large terms. Indeed, it is often the case that Lagrange polynomials take very large

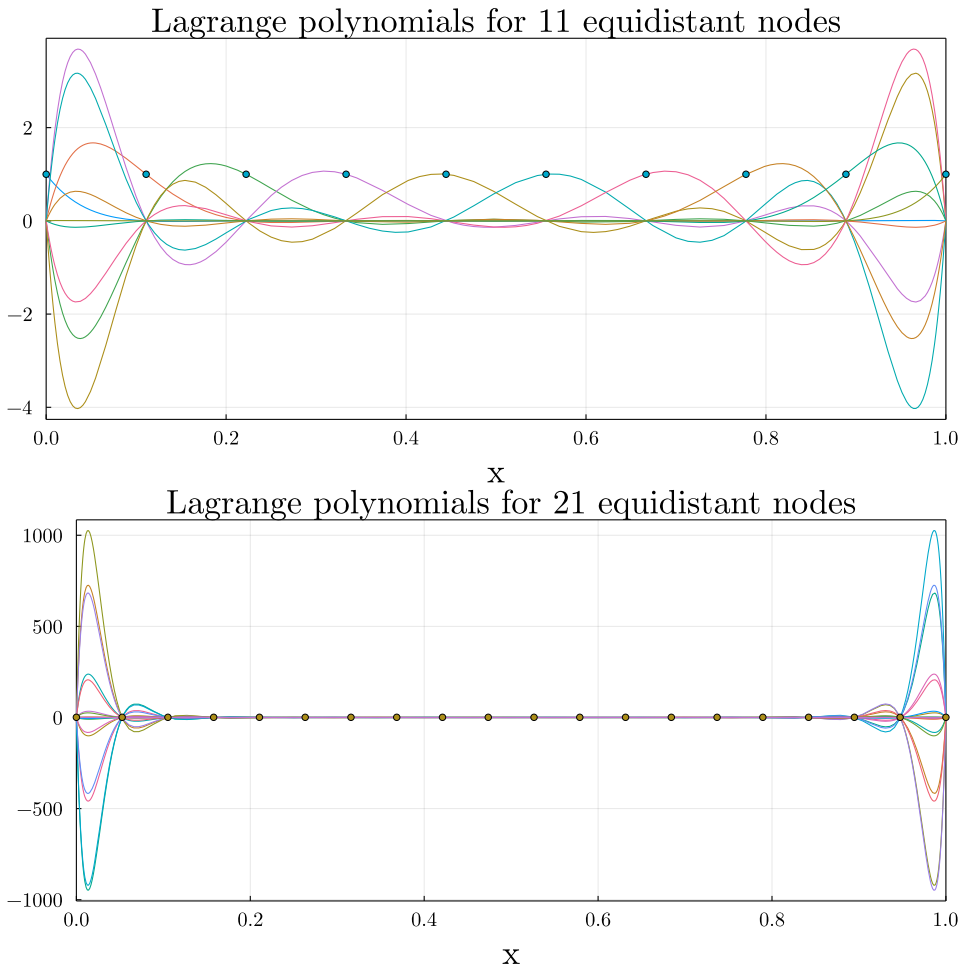

Figure 2.1: Lagrange polynomials associated with equidistant nodes over the $(0, 1)$ interval.

values over the interpolation intervals; this occurs, for example, when many equidistant interpolation nodes are employed, as illustrated in Figure 2.1.

2.1.3 Gregory–Newton interpolation

By Taylor's formula, any polynomial p of degree n may be expressed as

$$p(x) = p(0) + p'(0)x + \frac{p''(0)}{2}x^2 + \ldots + \frac{p^{(n)}(0)}{n!}x^n. \tag{2.3}$$

The constant coefficient can be obtained by evaluating the polynomial at 0, the linear coefficient can be identified by evaluating the first derivative at 0, and so on. Assume now that we are given the values taken by p when evaluated at the integer numbers $\{0, \ldots, n\}$. We ask the following question: can we find a formula similar in spirit to (2.3), but including only evaluations of p and not of its derivatives? To answer this question, we introduce the difference operator Δ which acts on functions as follows:

$$\Delta f(x) = f(x + 1) - f(x).$$

The operator Δ is a linear operator on the space of continuous functions. It maps constant functions to 0, and the linear function x to the constant function 1, suggesting a resemblance with the differentiation operator. In order to further understand this connection, let us define the *falling power* of a real number x as

$$x^{\underline{k}} = x(x-1)(x-2)\dots(x-k+1). \tag{2.4}$$

We then have that

$$\begin{aligned}\Delta x^{\underline{k}} &= (x+1)x(x-1)\dots(x-k+2) - x(x-1)(x-2)\dots(x-k+1) \\ &= \Big((x+1)-(x-k+1)\Big)\Big(x(x-1)\dots(x-k+2)\Big) = kx^{\underline{k-1}}\end{aligned} \tag{2.5}$$

In other words, the action of the difference operator on falling powers mirrors that of the differentiation operator on monomials. The falling powers form a basis of the space of polynomials, and so any polynomial in $\mathbf{P}(n)$, i.e. of degree less than or equal to n, can be expressed as

$$p(x) = \alpha_0 + \alpha_1 x^{\underline{1}} + \alpha_2 x^{\underline{2}} + \dots + \alpha_n x^{\underline{n}}. \tag{2.6}$$

It is immediate to show that $\alpha_i = \Delta^i p(0)/i!$, where $\Delta^i p$ denotes the function obtained after i applications of the operator Δ. Therefore, any polynomial of degree less than or equal to n may be expressed as

$$p(x) = p(0) + \Delta p(0) x^{\underline{1}} + \frac{\Delta^2 p(0)}{2} x^{\underline{2}} + \dots + \frac{\Delta^n p(0)}{n!} x^{\underline{n}}. \tag{2.7}$$

An expansion of the form (2.7) is called a *Newton series*, which is the discrete analog of the continuous Taylor series. From the definition of Δ, it is clear that the coefficients in (2.7) depend only on $p(0), \dots, p(n)$. We conclude that, given points $n+1$ points (i, u_i) for $i \in \{0, \dots, n\}$, the unique interpolating polynomial is given by (2.7), after replacing $p(i)$ by u_i.

Example 2.1. Let us use (2.6) in order to calculate the value of

$$S(n) := \sum_{i=0}^{n} i^2.$$

Since $\Delta S(n) = (n+1)^2$, which is a second degree polynomial in n, we deduce that $S(n)$ is a polynomial of degree 3. Let us now determine its coefficients.

n	0	1	2	3
$\Delta^0 S(n)$	**0**	1	5	14
$\Delta^1 S(n)$	**1**	4	9	
$\Delta^2 S(n)$	**3**	5		
$\Delta^3 S(n)$	**2**			

We conclude that

$$S(n) = \mathbf{1}n + \frac{\mathbf{3}}{2!}n(n-1) + \frac{\mathbf{2}}{3!}n(n-1)(n-2) = \frac{n(2n+1)(n+1)}{6}$$

Notice that when falling powers are employed as polynomial basis, the matrix in (2.1) is lower triangular, and so the algorithm described in Example 2.1 could be replaced by the forward substitution method. Whereas the coefficients of the Lagrange interpolant can be obtained immediately from the values of u at the nodes, calculating the coefficients of the expansion in (2.6) requires $\mathcal{O}(n^2)$ operations. However, Gregory–Newton interpolation has several advantages over Lagrange interpolation:

- If a point $(n+1, p_{n+1})$ is added to the set of interpolation points, only one additional term, corresponding to the falling power $x^{\underline{n+1}}$, needs to be calculated in (2.7). All the other coefficients are unchanged. Therefore, the Gregory–Newton approach is well-suited for incremental interpolation.

- The Gregory–Newton interpolation method is more numerically stable than Lagrange interpolation, because the basis functions do not take very large values.

- A polynomial in the form of a Newton series can be evaluated efficiently using Horner's method, which is based on rewriting the polynomial as

$$p(x) = \alpha_0 + x\Big(\alpha_1 + (x-1)\Big(\alpha_2 + (x-2)\Big(\alpha_3 + (x-3)\dots\Big)\Big)\Big).$$

  Evaluating this expression starting from the innermost bracket leads to an algorithm with a cost scaling linearly with the degree of the polynomial.

Non-equidistant nodes

So far, we have described the Gregory–Newton method in the simple setting where interpolation nodes are just a sequence of successive natural numbers. The method can be generalized to the setting of nodes $x_0 \neq \dots \neq x_n$ which are not necessarily equidistant. In this case, we take as basis the following functions instead of the falling powers:

$$\varphi_i(x) = (x - x_0)(x - x_1)\dots(x - x_{i-1}), \tag{2.8}$$

with the convention that the empty product is 1. By (2.1), the coefficients of the interpolating polynomial in this basis solve the following linear system:

$$\begin{pmatrix} 1 & & \dots & & 0 \\ 1 & x_1 - x_0 & & & \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & \vdots \\ \vdots & \vdots & & \ddots & \\ 1 & x_n - x_0 & \dots & \dots & \prod_{j=0}^{n-1}(x_n - x_j) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}. \tag{2.9}$$

This system could be solved using, for example, forward substitution. Clearly $\alpha_0 = u_0$ from the first equation, and then from the second equation we obtain

$$\alpha_1 = \frac{u_1 - u_0}{x_1 - x_0} =: [u_0, u_1],$$

which may be viewed as an approximation of the slope of u at x_0. The right-hand side of this equation is an example of a *divided difference*. In general, divided differences are defined recursively as follows:

$$[u_0, u_1, \ldots, u_d] := \frac{[u_1, \ldots, u_d] - [u_0, \ldots, u_{d-1}]}{x_d - x_0}, \qquad [u_i] = u_i. \tag{2.10}$$

Let us start by observing that our divided differences coincide with the coefficients in (2.6).

Lemma 2.1. *Assume that* $(0, u_0), \ldots, (n, u_n) \in \mathbf{R}^2$. *Then, provided that* $(i, k) \in \mathbf{N}^2$ *satisfies* $i + k \leqslant n$, *we have*

$$\frac{1}{k!}\Delta^k u(i) = [u_i, \ldots, u_{i+k}].$$

Proof. The result is clearly true for $k = 0$, and we shall assume by induction that it holds for $k - 1$. Using our induction hypothesis, we get

$$\begin{aligned}\frac{1}{k!}\Delta^k u(i) &= \frac{1}{k}\left(\frac{\Delta^{k-1}u(i+1)}{(k-1)!} - \frac{\Delta^{k-1}u(i)}{(k-1)!}\right) \\ &= \frac{1}{k}\Big([u_{i+1}, \ldots, u_{i+k}] - [u_i, \ldots, u_{i+k-1}]\Big).\end{aligned}$$

By the definition of divided differences in (2.10), it follows that we have

$$\frac{1}{k!}\Delta^k u(i) = \frac{1}{k}(\underbrace{x_{i+k} - x_i}_{i+k-i})[u_i, \ldots, u_{i+k}] = [u_i, \ldots, u_{i+k}],$$

which allows us to conclude the proof. □

In light of Lemma 2.1, the reader will not be surprised that the coefficients in the $\varphi_0, \ldots, \varphi_n$ basis in (2.8) are given by the divided differences.

Proposition 2.2. *Assume that* $(x_0, u_0), \ldots, (x_n, u_n)$ *are* $n + 1$ *points in the plane with distinct abcissae. Then the interpolating polynomial of degree* n *may be expressed as*

$$p(x) = \sum_{i=0}^{n} [u_0, \ldots, u_i]\varphi_i(x),$$

where $\varphi_i(x)$, *for* $i = 0, \ldots, n$, *are the basis functions defined in* (2.8).

Proof. The statement is true for $n = 0$. Reasoning by induction, we assume that it holds true for polynomials of degree up to $n - 1$. Let $p_1(x)$ and $p_2(x)$ be the interpolating polynomials at the points $x_0, x_1, \ldots, x_{n-2}, x_{n-1}$ and $x_0, x_1, \ldots, x_{n-2}, x_n$, respectively. Then

$$p(x) = p_1(x) + \frac{x - x_{n-1}}{x_n - x_{n-1}}\big(p_2(x) - p_1(x)\big) \tag{2.11}$$

is a polynomial of degree n that interpolates all the data points. By the induction hypothesis, it holds that

$$p_1(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, u_1, \ldots, u_{n-2}, \mathbf{u_{n-1}}] \prod_{i=0}^{n-2}(x - x_i),$$
$$p_2(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, u_1, \ldots, u_{n-2}, \mathbf{u_n}] \prod_{i=0}^{n-2}(x - x_i).$$

Here we used a bold font in order to emphasize the difference between the two expressions. Substituting these expressions in (2.11), we obtain

$$p(x) = u_0 + [u_0, u_1](x - x_0) + \ldots + [u_0, \ldots, u_{n-2}] \prod_{i=0}^{n-2}(x - x_i)$$
$$+ \frac{[u_0, u_1, \ldots, u_{n-2}, u_n] - [u_0, u_1, \ldots, u_{n-2}, u_{n-1}]}{x_n - x_{n-1}} \prod_{i=0}^{n-1}(x - x_i).$$

In Exercise 2.4, we show that divided differences are invariant under permutations of the data points, and so we have that

$$\frac{[u_0, u_1, \ldots, u_{n-2}, u_n] - [u_0, u_1, \ldots, u_{n-2}, u_{n-1}]}{x_n - x_{n-1}} = [u_0, \ldots, u_n],$$

which enables to conclude. □

Example 2.2. Assume that we are looking for the third degree polynomial going through the following points:

$$(-1, 10), \qquad (0, 4), \qquad (2, -2), \qquad (4, -40).$$

We have to calculate the divided difference $\alpha_i = [u_0, \ldots, u_i]$ for $i \in \{0, 1, 2, 3\}$. To this end, it is convenient to use a table:

i	0	1	2	3
$[u_i]$	**10**	4	-2	-40
$x_{i+1} - x_i$	1	2	2	
$[u_i, u_{i+1}]$	**-6**	-3	-19	
$x_{i+2} - x_i$	3	4		
$[u_i, u_{i+1}, u_{i+2}]$	**1**	-4		
$x_{i+3} - x_i$	5			
$[u_i, u_{i+1}, u_{i+2}, u_{i+3}]$	**-1**			

We deduce that the expression of the interpolating polynomial is

$$p(x) = \mathbf{10} + (\mathbf{-6})(x + 1) + \mathbf{1}(x + 1)x + (\mathbf{-1})(x + 1)x(x - 2) = -x^3 + 2x^2 + -3x + 4.$$

2.1.4 Interpolation error

Assume that $u(x)$ is a continuous function and denote by $\hat{u}(x)$ its interpolation through the points (x_i, u_i), where $u_i = u(x_i)$ for $i = 0, \dots, n$. In this section, we study the behavior of the error in the limit as $n \to \infty$.

Theorem 2.3 (Interpolation error)**.** *Assume that u: $[a, b] \to \mathbf{R}$ is a function in $C^{n+1}([a, b])$ and let $x_0, \dots, x_n$ denote $n + 1$ distinct interpolation nodes. Then for all $x \in [a, b]$, there exists $\xi = \xi(x)$ in the interval $[a, b]$ such that*

$$e_n(x) := u(x) - \hat{u}(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0) \dots (x - x_n).$$

Proof. The statement is obvious if $x \in \{x_0, \dots, x_n\}$, so we assume from now on that x does not coincide with an interpolation node. Let us use the compact notation $\omega_n = \prod_{i=0}^{n}(x - x_i)$ and introduce the function

$$g(t) = e_n(t)\omega_n(x) - e_n(x)\omega_n(t). \tag{2.12}$$

The function g is smooth and takes the value 0 when evaluated at $x_0, \dots, x_n, x$. Since g has $n + 2$ roots in the interval $[a, b]$, Rolle's theorem implies that g' has at least $n + 1$ distinct roots in (a, b). Another application of Rolle's theorem then yields that g'' has at least n distinct roots in (a, b). Iterating this reasoning, we deduce that $g^{(n+1)}$ has one root t_* in (a, b). From (2.12), we calculate that

$$g^{(n+1)}(t) = e_n^{(n+1)}(t)\omega_n(x) - e_n(x)\omega_n^{(n+1)}(t) = u^{(n+1)}(t)\omega_n(x) - e_n(x)(n+1)!. \tag{2.13}$$

Here we employed the fact that $\hat{u}^{(n+1)}(t) = 0$, because $\hat{u}$ is a polynomial of degree at most n. Evaluating (2.13) at t_* and rearranging, we obtain that

$$e_n(x) = \frac{u^{(n+1)}(t_*)}{(n+1)!}\omega_n(x),$$

which completes the proof. □

As a corollary to Theorem 2.3, we deduce the following error bound.

Corollary 2.4 (Upper bound on the interpolation error)**.** *Assume that u is smooth in the interval $[a, b]$ and let*

$$C_{n+1} = \sup_{x \in [a,b]} \left|u^{(n+1)}(x)\right|.$$

Then

$$E_n := \sup_{x \in [a,b]} \left|e_n(x)\right| \leqslant \frac{C_{n+1}}{4(n+1)} h^{n+1} \tag{2.14}$$

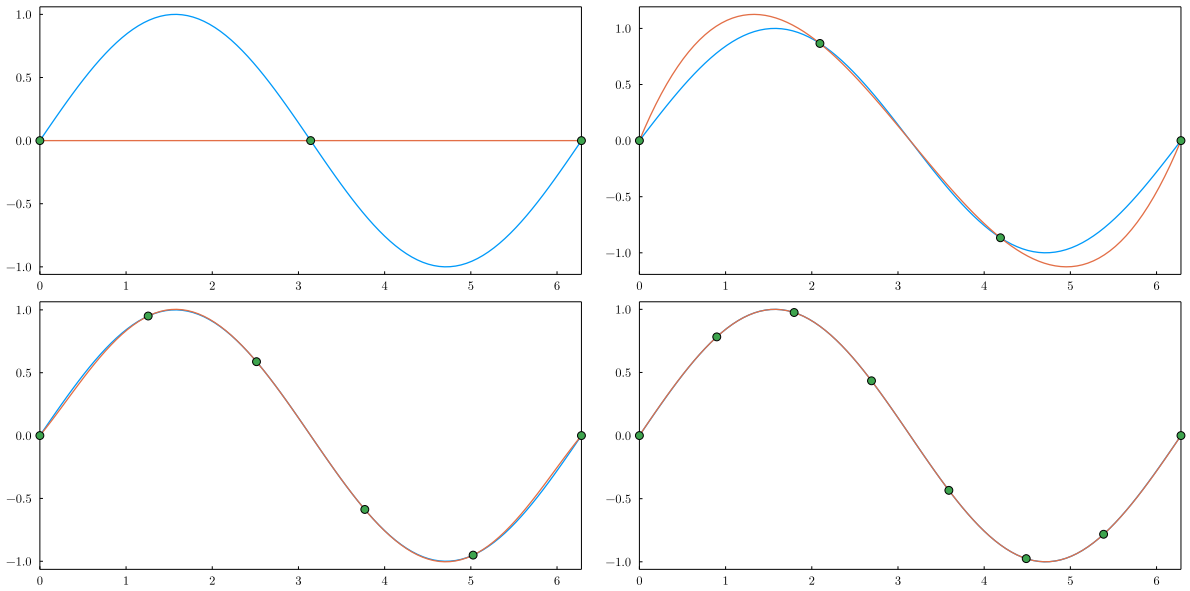

Figure 2.2: Interpolation (in orange) of the function $u(x) = \sin(x)$ (in blue) using 3, 4, 6, and 8 equidistant nodes.

where h is the maximum spacing between two successive interpolation nodes.

Proof. By Theorem 2.3, it holds that

$$\forall x \in [a, b], \qquad |e_n(x)| \leqslant \frac{C_{n+1}}{(n+1)!} \Big| (x - x_0) \dots (x - x_n) \Big|. \tag{2.15}$$

The product on the right-hand side is bounded from above by

$$\frac{h^2}{4} \times 2h \times 3h \times 4h \times \cdots \times nh = \frac{n! h^{n+1}}{4}. \tag{2.16}$$

The first factor comes from the fact that, if $x \in [x_i, x_{i+1}]$, then

$$\Big| (x - x_i)(x - x_{i+1}) \Big| \leqslant \frac{(x_{i+1} - x_i)^2}{4},$$

because the left-hand side is maximized when x is the midpoint of the interval $[x_i, x_{i+1}]$. Substituting (2.16) into (2.15), we deduce the statement. □

We now ask the following natural question: does E_n given in (2.14) tend to zero as the maximum spacing between successive nodes tends to 0? By Corollary 2.4, the answer to this question is positive when C_n does not grow too quickly as $n \to \infty$. For example the interpolation error for the function $u(x) = \sin(x)$ decreases very quickly as $n \to \infty$ when equidistant interpolation nodes are employed, as illustrated in Figure 2.2.

In some cases, however, the constant C_n grows quickly with n, to the extent that E_n may increase with n; in this case, the maximum interpolation error grows when nodes are added! The classic example illustrating this potential issue is that of the Runge function:

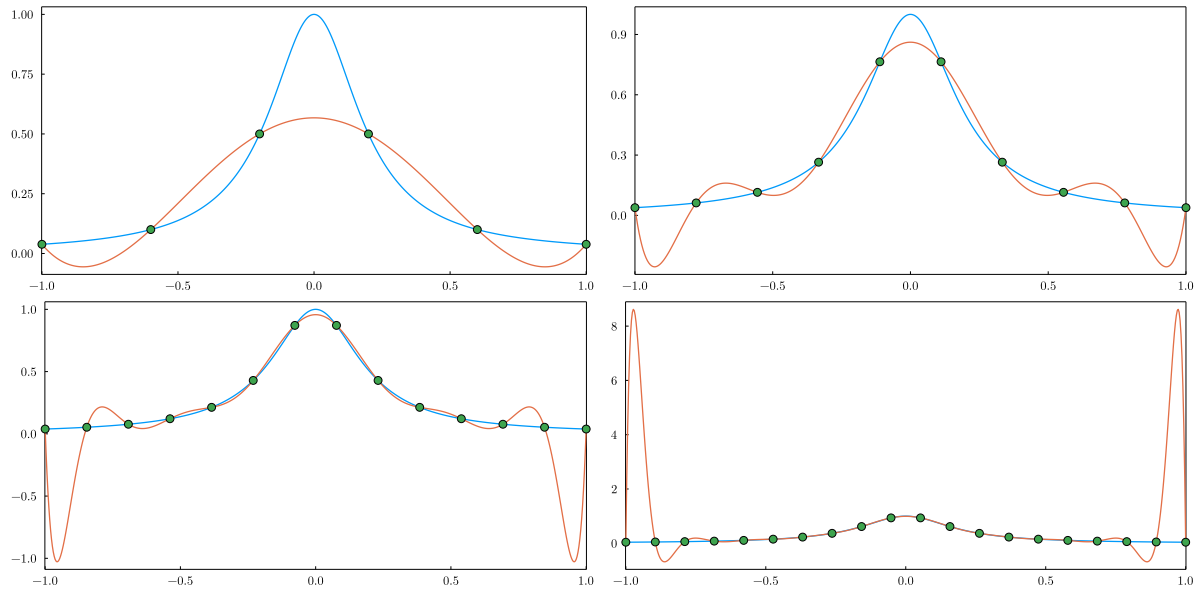

Figure 2.3: Interpolation (in orange) of the Runge function (2.17) (in blue) using 6, 10, 14, and 20 equidistant nodes.

$$u(x) = \frac{1}{1 + 25x^2}. \tag{2.17}$$

It is possible to show that, for this function, the upper bound in (2.14) tends to ∞ in the limit as the number n of interpolation nodes increases. We emphasize that this does not prove that $E_n \to \infty$ in the limit as $n \to \infty$, because (2.14) provides only an *upper bound* on the error. In fact, the interpolation error for the Runge function can either grow or decrease, depending on the choice of interpolation nodes. With equidistant nodes, it turns out that $E_n \to \infty$, as illustrated in Figure 2.3.

2.1.5 Interpolation at Chebyshev nodes

Sometimes, interpolation is employed as a technique for approximating functions. The spectral collocation method, for example, is a technique for solving partial differential equations where a discrete solution is first calculated, and then a continuous solution is constructed using polynomial or Fourier interpolation. When the interpolation nodes are not given a priori as data, it is natural to wonder whether these can be picked in such a manner that the interpolation error, measured in a function norm, is minimized. For example, given a continuous function $u(x)$ and a number of nodes $n + 1$, is it possible to choose nodes $x_0, \ldots, x_n$ such that

$$E := \sup_{x \in [a,b]} \left| u(x) - \hat{u}(x) \right|$$

is minimized? Here $\hat{u}$ is the polynomial interpolating u at the nodes. Achieving this goal in general is a difficult task, because $\xi = \xi(x)$ is unknown in the expression of the interpolation error from Theorem 2.3:

$$e_n(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0)\dots(x - x_n).$$

In view of this difficulty, we will focus on the simpler problem of finding interpolation nodes such that the product $(x - x_0)\dots(x - x_n)$ is minimized in the supremum norm. This problem amounts to finding the optimal interpolation nodes, in the sense that E is minimized, in the particular case where u is a polynomial of degree $n+1$, because in this case $u^{(n+1)}(\xi)$ is a constant factor. It turns out that this problem admits an explicit solution, which we will deduce from the following theorem.

Theorem 2.5 (Minimum ∞ norm)**.** *Assume that p is a monic polynomial of degree $n \geqslant 1$:*

$$p(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_{n-1}x^{n-1} + x^n.$$

Then it holds that

$$\sup_{x\in[-1,1]} \left|p(x)\right| \geqslant \frac{1}{2^{n-1}} =: E. \tag{2.18}$$

In addition, the lower bound is achieved for $p_*(x) = 2^{-(n-1)}T_n(x)$, where T_n is the Chebyshev polynomial of degree n:

$$T_n(x) = \cos(n \arccos x) \qquad (-1 \leqslant x \leqslant 1). \tag{2.19}$$

Proof. By Exercise C.5, the polynomial $x \mapsto 2^{-(n-1)}T_n(x)$ is indeed monic, and it is clear that it achieves the lower bound (2.18) since $|T_n(x)| \leqslant 1$ for all $x \in [-1, 1]$.

In order to prove (2.18), we assume by contradiction that there is a different monic polynomial q of degree n such that

$$\sup_{x\in[-1,1]} \left|q(x)\right| < E. \tag{2.20}$$

Let us introduce $x_i = \cos(i\pi/n)$, for $i = 0, \dots, n$, and observe that

$$p(x_i) = 2^{-(n-1)}T_n(x_i) = (-1)^i E.$$

The function $h(x) \coloneqq p(x) - q(x)$ is a polynomial of degree at most $n-1$ which, by the assumption (2.20), is strictly positive at $x_0, x_2, x_4, \dots$ and strictly negative at $x_1, x_3, x_5, \dots$. Therefore, the polynomial $h(x)$ changes sign at least n times and so, by the intermediate value theorem, it has at least n roots. But this is impossible, because $h(x) \neq 0$ and $h(x)$ is of degree at most $n-1$. □

Remark 2.1 (Derivation of Chebyshev polynomials). The polynomial p_* achieving the lower bound in (2.18) oscillates between the values $-E$ and E, which are respectively its minimum and maximum values over the interval $[-1, 1]$. It attains the values E or $-E$ at $n+1$ distinct points $x_0 < \dots < x_n$, with $x_0 = -1$ and $x_n = 1$. It turns out that these properties, which

can be shown to hold a priori using Chebyshev's *equioscillation theorem*, are sufficient to derive an explicit expression for the polynomial p_*, as we formally demonstrate hereafter.

The points $x_1, \dots, x_{n-1}$ are local extrema of p_*, and so $p_*'(x) = 0$ at these nodes. We therefore deduce that p_* satisfies the differential equation

$$n^2(E^2 - p_*(x)^2) = p_*'(x)^2(1 - x^2). \tag{2.21}$$

Indeed, both sides are polynomials of degree $2n$ with single roots at -1 and 1, with double roots at $x_1, \dots, x_{n-1}$, and with the same coefficient of the leading power. In order to solve (2.21), we rearrange the equation and take the square root:

$$\frac{p_*' \frac{x}{E}}{\sqrt{1 - \frac{p_*(x)^2}{E^2}}} = \pm \frac{n}{\sqrt{1 - x^2}} \qquad \iff \qquad \frac{\mathrm{d}}{\mathrm{d}\,x}\left(\arccos\left(\frac{p_*(x)}{E}\right)\right) = \pm n \frac{\mathrm{d}}{\mathrm{d}\,x} \arccos(x).$$

Integrating both sides and taking the cosine, we obtain

$$p_*(x) = E \cos\Big(C + n \arccos(x)\Big).$$

Requiring that $|p_*(-1)| = E$, we deduce $C = 0$.

From Theorem 2.5, we deduce the following corollary.

Corollary 2.6 (Chebyshev nodes)**.** *Assume that $x_0 < x_1 < \ldots < x_n$ are in the interval $[a, b]$. The supremum norm of the product $\omega(x) \coloneqq (x - x_0) \cdots (x - x_n)$ over $[a, b]$ is minimized when*

$$x_i = a + (b - a) \frac{1 + \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)}{2} \tag{2.22}$$

Proof. We consider the affine change of variable

$$\begin{aligned} \zeta \colon [-1, 1] &\to [a, b]; \\ y &\mapsto \frac{a + b + y(b - a)}{2}. \end{aligned}$$

The function

$$\begin{aligned} p(y) &\coloneqq \frac{2^{n+1}}{(b-a)^{n+1}} \omega\big(\zeta(y)\big) = \frac{2^{n+1}}{(b-a)^{n+1}} \big(\zeta(y) - x_0\big) \cdots \big(\zeta(y) - x_n\big) \\ &= \big(y - y_0\big) \cdots \big(y - y_n\big), \qquad y_i = \zeta^{-1}(x_i), \end{aligned}$$

is a monic polynomial of degree $n + 1$ such that

$$\sup_{y \in [-1,1]} |p(y)| = \frac{2^{n+1}}{(b-a)^{n+1}} \sup_{x \in [a,b]} |(x - x_0) \dots (x - x_n)|. \tag{2.23}$$

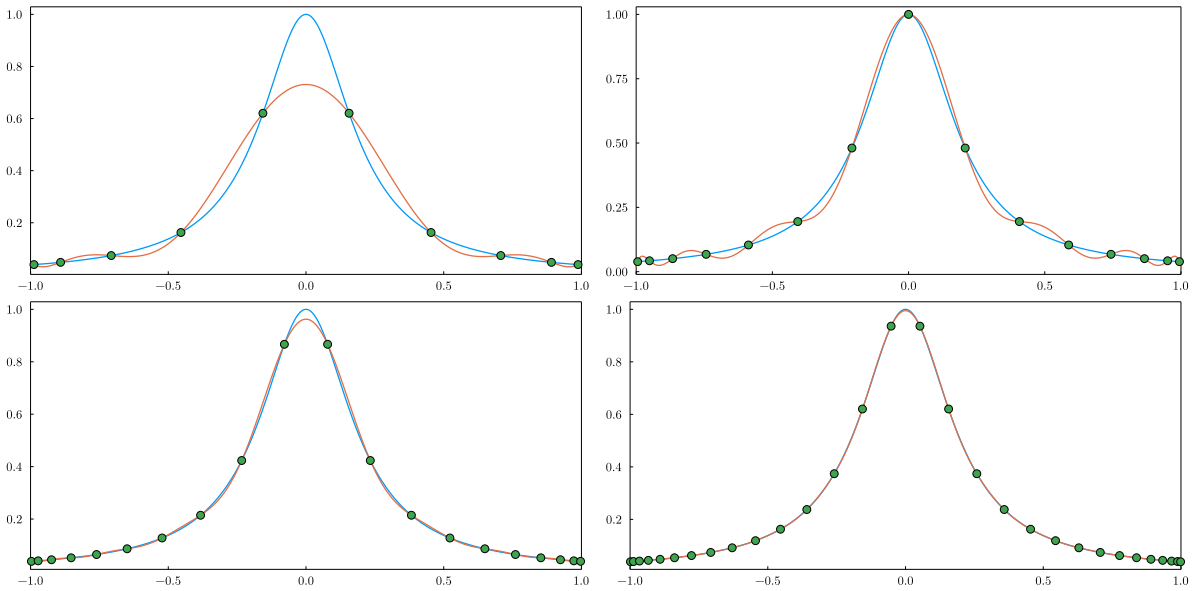

Figure 2.4: Interpolation (in orange) of the Runge function (2.17) (in blue) using 10, 15, 20, and 30 Chebyshev nodes.

By Theorem 2.5, the left-hand side is minimized when p is equal to $2^{-n}T_{n+1}$, i.e. when the roots of p coincide with the roots of T_{n+1}. This occurs when

$$y_i = \zeta^{-1}(x_i) = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right).$$

Applying the inverse change of variable $x_i = \zeta(y_i)$, we deduce the result. □

Corollary 2.6 is useful for interpolation. The nodes

$$x_i = a + (b-a)\frac{1+\cos\left(\frac{(2i+1)\pi}{2(n+1)}\right)}{2}, \qquad i = 0, \ldots, n, \tag{2.24}$$

are known as Chebyshev nodes and, more often than not, employing these nodes for interpolation produces much better results than using equidistant nodes, both in the case where u is a polynomial of degree $n+1$, as we just proved, but also for general u. As an example we plot in Figure 2.4 the interpolation of the Runge function using Chebyshev nodes. In this case, the interpolating polynomial converges uniformly to the Runge function as we increase the number of interpolation nodes!

2.1.6 Hermite interpolation

Hermite interpolation, sometimes also called Hermite–Birkoff interpolation, generalizes Lagrange interpolation to the case where, in addition to the function values $u_0, \ldots, u_n$, the values of some of the derivatives are given at the interpolation nodes. For simplicity, we assume in this section that only the first derivative is specified. In this case, the aim of Hermite interpolation is to find, given data (x_i, u_i, u_i') for $i \in \{0, \ldots, n\}$, a polynomial $\hat{u}$ of degree at most $2n+1$ such that

$$\forall i \in \{0, \dots, n\}, \qquad \hat{u}(x_i) = u_i, \qquad \hat{u}'(x_i) = u_i'.$$

In order to construct the interpolating polynomial, it is useful to define the functions

$$\psi_i(x) = \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)^2, \qquad i = 0, \dots, n.$$

The function ψ_i is the square of the usual Lagrange polynomials associated with x_i, and it satisfies

$$\psi_i(x_i) = 1, \qquad \psi_i'(x_i) = \sum_{j=0, j\neq i}^{n} \frac{2}{x_i - x_j}, \qquad \forall j \neq i \qquad \psi_i(x_j) = \psi_i'(x_j) = 0.$$

We consider the following ansatz for $\hat{u}$:

$$\hat{u}(x) = \sum_{i=0}^{n} \psi_i(x) q_i(x),$$

where q_i are polynomials to be determined of degree at most one, so that $\hat{u}$ is of degree at most $2n + 1$. We then require

$$\hat{u}(x_i) = q_i(x_i), \qquad \hat{u}'(x_i) = \psi_i'(x_i) q(x_i) + q'(x_i).$$

From the first equation, we deduce that $q_i(x_i) = u_i$, and from the second equation we then have $q'(x_i) = \hat{u}'(x_i) - \psi_i'(x_i) u_i$. We conclude that the interpolating polynomial is given by

$$\hat{u}(x) = \sum_{i=0}^{n} \psi_i(x) \Big(u_i + \big(u_i' - \psi_i'(x_i) u_i \big)(x - x_i) \Big).$$

The following theorem gives an expression of the error.

Theorem 2.7 (Hermite interpolation error)**.** *Assume that $u\colon [a, b] \to \mathbf{R}$ is a function in $C^{2n+2}([a, b])$ and let $\hat{u}$ denote the Hermite interpolation of u at the nodes $x_0, \dots, x_n$. Then for all $x \in [a, b]$, there exists $\xi = \xi(x)$ in the interval $[a, b]$ such that*

$$u(x) - \hat{u}(x) = \frac{u^{(2n+2)}(\xi)}{(2n + 2)!} (x - x_0)^2 \dots (x - x_n)^2.$$

Proof. See Exercise 2.8. □

2.1.7 Piecewise interpolation

The interpolation methods we discussed so far are in some sense global; they aim to construct one polynomial that goes through all the data points. This approach is attractive because the interpolant is infinitely smooth but, as we observed, it is not always fruitful, in particular when equidistant interpolation nodes are employed. An alternative approach is to divide the domain in a number of small intervals and perform polynomial interpolation within each interval.

Although the resulting interpolating function is usually not smooth over the full domain, this "local" approach to interpolation is more robust.

Several methods belong in the category of piecewise interpolation. We mention, for instance, piecewise Lagrange interpolation and cubic splines interpolation. In this section, we briefly describe the former method, which is widely used in the context of the *finite element method*. Information on the latter method is available in [10, Section 8.7.1.].

For simplicity, we illustrate the method in dimension 1, but piecewise Lagrange interpolation can be extended to several dimensions. Assume that we wish to approximate a function $u\colon [a,b] \to \mathbf{R}$. We consider a subdivision $a = x_0 < x_1 < \ldots < x_n = b$ of the interval $[a,b]$ and let h denote the maximum spacing:

$$h = \max_{i \in \{0,\ldots,n-1\}} |x_{i+1} - x_i|.$$

Within each subinterval $I_i = [x_i, x_{i+1}]$, we consider a further subdivision

$$x_i = x_i^{(0)} < x_i^{(1)} < \ldots < x_i^{(m)} = x_{i+1},$$

where the nodes $x_i^{(0)}, \ldots, x_i^{(m)}$ are equally spaced with distance h/m. The idea of piecewise Lagrange interpolation is to calculate, for each interval I_i in the partition, the interpolating polynomial p_i at the nodes $x_i^{(0)}, \ldots, x_i^{(m)}$. The interpolant is then defined as

$$\hat{u}(x) = p_\iota(x), \tag{2.25}$$

where $\iota = \iota(x)$ is the index of the interval to which x belongs. Since $x_i^{(m)} = x_{i+1} = x_{i+1}^{(0)}$, the interpolant defined by (2.25) is continuous. If the function u belongs to $C^{m+1}([a,b])$, then by Corollary 2.4 the interpolation error within each subinterval may be bounded from above as follows:

$$\sup_{x \in I_i} |u(x) - \hat{u}(x)| \leqslant \frac{C_{m+1}(h/m)^{m+1}}{4(m+1)}, \qquad C_{m+1} := \sup_{x \in [a,b]} |u^{(m+1)}(x)|, \tag{2.26}$$

and so we deduce

$$\sup_{x \in [a,b]} |u(x) - \hat{u}(x)| \leqslant Ch^{m+1},$$

for an appropriate constant C independent of h. This equation shows that the error is guaranteed to decrease to 0 in the limit as $h \to 0$. In practice, the number m of interpolation nodes within each interval can be small.

2.2 Approximation

In this section, we focus on the subject of *approximation*, both of discrete data points and of continuous functions. We begin, in Section 2.2.1 with a discussion of least squares approximation

for data points, and in Section 2.2.2 we focus on function approximation in the mean square sense.

2.2.1 Least squares approximation of data points

Consider $n+1$ distinct x values $x_0 < \ldots < x_n$, and suppose that we know the values $u_0, \ldots, u_n$ taken by an unknown function u when evaluated at these points. We wish to approximate the function u by a function of the form

$$\widehat{u}(x) = \sum_{i=0}^{m} \alpha_i \varphi_i(x) \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}, \tag{2.27}$$

for some $m < n$. In many cases of practical interest, the basis functions $\varphi_0, \ldots, \varphi_m$ are polynomials. In contrast with interpolation, here we seek a function $\widehat{u}$ in a finite-dimensional function space of dimension m strictly lower than the number of data points. In order for $\widehat{u}$ to be a good approximation, we wish to find coefficients $\alpha_0, \ldots, \alpha_m$ such that the following linear system is approximately satisfied.

$$\mathsf{A}\boldsymbol{\alpha} := \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_m(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_m(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \ldots & \varphi_m(x_2) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_{n-2}) & \varphi_1(x_{n-2}) & \ldots & \varphi_m(x_{n-2}) \\ \varphi_0(x_{n-1}) & \varphi_1(x_{n-1}) & \ldots & \varphi_m(x_{n-1}) \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_m(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix} \approx \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{pmatrix} =: \boldsymbol{b}.$$

In general, since the matrix on the left-hand side has more lines than columns, there does not exist an exact solution to this equation. In order to find an approximate solution, a natural approach is to find coefficients $\alpha_0, \ldots, \alpha_m$ such that the residual vector $\boldsymbol{r} = \mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}$ is small in some vector norm. A particularly popular approach, known as least squares approximation, is to minimize the Euclidean norm of $\boldsymbol{r}$ or, equivalently, the square of the Euclidean norm:

$$\|\boldsymbol{r}\|^2 = \sum_{i=0}^{n} \left| u_i - \widehat{u}(x_i) \right|^2 = \sum_{i=0}^{n} \left(u_i - \sum_{j=0}^{m} \alpha_j \varphi_j(x_i) \right)^2.$$

Let us denote the right-hand side of this equation by $J(\boldsymbol{\alpha})$, which we view as a function of the vector of coefficients $\boldsymbol{\alpha}$. A necessary condition for $\boldsymbol{a}_*$ to be a minimizer is that $\nabla J(\boldsymbol{\alpha}_*) = 0$. The gradient of J, written as a column vector, is given by

$$\begin{aligned} \nabla J(\boldsymbol{\alpha}) &= \nabla \Big((\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b})^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) \Big) \\ &= \nabla \Big(\boldsymbol{\alpha}^T (\mathsf{A}^T \mathsf{A}) \boldsymbol{\alpha} - \boldsymbol{b} \mathsf{A} \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathsf{A}^T \boldsymbol{b} + \boldsymbol{b}^T \boldsymbol{b} \Big)) \\ &= 2(\mathsf{A}^T \mathsf{A}) \boldsymbol{\alpha} - 2\mathsf{A}^T \boldsymbol{b}. \end{aligned}$$

We deduce that $\boldsymbol{\alpha}_*$ solves the linear system

$$\mathsf{A}^T\mathsf{A}\boldsymbol{\alpha}_* = \mathsf{A}^T\boldsymbol{b}, \tag{2.28}$$

where the matrix on the left-hand side is given by:

$$\mathsf{A}^T\mathsf{A} := \begin{pmatrix} \sum_{i=0}^n \varphi_0(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_0(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_0(x_i)\varphi_m(x_i) \\ \sum_{i=0}^n \varphi_1(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_1(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_1(x_i)\varphi_m(x_i) \\ \vdots & \vdots & & \vdots \\ \sum_{i=0}^n \varphi_m(x_i)\varphi_0(x_i) & \sum_{i=0}^n \varphi_m(x_i)\varphi_1(x_i) & \dots & \sum_{i=0}^n \varphi_m(x_i)\varphi_m(x_i) \end{pmatrix}.$$

Equation (2.28) is a system of m equations with m unknowns, which admits a unique solution provided that $\mathsf{A}^T\mathsf{A}$ is full rank or, equivalently, the columns of A are linearly independent. The linear equations (2.28) are known as the *normal equations*. As a side note, we mention that the solution $\boldsymbol{\alpha}_* = (\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T\boldsymbol{b}$ coincides with the maximum likelihood estimator for α under the assumption that the data is generated according to $u_i = u(x_i) + \varepsilon_i$, for some function $u \in \operatorname{Span}\{\varphi_0, \dots, \varphi_m\}$ and random noise $\varepsilon_i \sim \mathcal{N}(0, 1)$.

Remark 2.2. From equation (2.28) we deduce that

$$\mathsf{A}\boldsymbol{\alpha}_* = \mathsf{A}(\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T\boldsymbol{b}.$$

The matrix $\Pi_\mathsf{A} := \mathsf{A}(\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T$ on the right-hand side is the orthogonal projection operator onto $\operatorname{col}(\mathsf{A}) \subset \mathbf{R}^n$, the subspace spanned by the columns of A. Indeed, it holds that $\Pi_\mathsf{A}^2 = \Pi_\mathsf{A}$, which is the defining property of a projection operator.

To conclude this section, we note that the matrix $\mathsf{A}^+ = (\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T$ is a left inverse of the matrix A, because $\mathsf{A}^+\mathsf{A} = \mathsf{I}$. It is also called the Moore–Penrose inverse or pseudoinverse of the matrix A, which generalizes the usual inverse matrix. In Julia, the backslash operator silently uses the Moore–Penrose inverse when employed with a rectangular matrix. Therefore, solving the normal equations (2.28) can be achieved by just writing `α = A\b`.

2.2.2 Mean square approximation of functions

The approach described in Section 2.2.1 can be generalized to the setting where the actual function u, rather than just discrete evaluations of it, is available. In this section, we seek an approximation of the form (2.27) such that the error $\hat{u}(x) - u(x)$, measured in some function norm, is minimized. Of course, the solution to this minimization problem depends in general on the norm employed, and may in some cases not even be unique. Instead of specifying a particular norm, as done in Section 2.2.1, in this section we retain some generality and assume only that the norm is induced by an inner product on the space of real-valued continuous functions:

$$\langle \bullet, \bullet \rangle \colon C([a,b]) \times C([a,b]) \to \mathbf{R}. \tag{2.29}$$

In other words, we seek to minimize

$$J(\boldsymbol{\alpha}) := \|\hat{u} - u\|^2 = \langle \hat{u} - u, \hat{u} - u \rangle.$$

This is again a function of the $m+1$ variables $\alpha_0, \ldots, \alpha_m$. Before calculating its gradient, we rewrite the function $J(\boldsymbol{\alpha})$ in a simpler form:

$$\begin{aligned} J(\boldsymbol{\alpha}) &= \left\langle u - \sum_{j=0}^{m} \alpha_j \varphi_j, u - \sum_{k=0}^{m} \alpha_k \varphi_k \right\rangle \\ &= \sum_{j=0}^{m} \sum_{k=0}^{m} \alpha_j \alpha_k \langle \varphi_j, \varphi_k \rangle - 2 \sum_{j=0}^{m} \alpha_j \langle u, \varphi_j \rangle + \langle u, u \rangle = \boldsymbol{\alpha}^T \mathsf{G} \boldsymbol{\alpha} - 2 \boldsymbol{b}^T \boldsymbol{\alpha} + \langle u, u \rangle, \end{aligned}$$

where we introduced

$$\mathsf{G} := \begin{pmatrix} \langle \varphi_0, \varphi_0 \rangle & \langle \varphi_0, \varphi_1 \rangle & \cdots & \langle \varphi_0, \varphi_m \rangle \\ \langle \varphi_1, \varphi_0 \rangle & \langle \varphi_1, \varphi_1 \rangle & \cdots & \langle \varphi_1, \varphi_m \rangle \\ \vdots & \vdots & & \vdots \\ \langle \varphi_m, \varphi_0 \rangle & \langle \varphi_m, \varphi_1 \rangle & \cdots & \langle \varphi_m, \varphi_m \rangle \end{pmatrix}, \qquad \boldsymbol{b} := \begin{pmatrix} \langle u, \varphi_0 \rangle \\ \langle u, \varphi_1 \rangle \\ \vdots \\ \langle u, \varphi_m \rangle \end{pmatrix}. \tag{2.30}$$

Employing the same approach as in the previous section, we then obtain $\nabla J(\boldsymbol{\alpha}) = \mathsf{G}\boldsymbol{\alpha} - \boldsymbol{b}$, and so the minimizer of $J(\boldsymbol{\alpha})$ is the solution to the equation

$$\mathsf{G}\boldsymbol{\alpha} = \boldsymbol{b}. \tag{2.31}$$

The matrix G, known as the *Gram matrix*, is positive semi-definite and nonsingular provided that the basis functions are linearly independent, see Exercise 2.9. Therefore, the solution $\boldsymbol{\alpha}_*$ exists and is unique. In addition, since the Hessian of J is equal to G, the vector $\boldsymbol{\alpha}_*$ is indeed a minimizer. Note that if $\langle \bullet, \bullet \rangle$ is defined as a finite sum of the form

$$\langle f, g \rangle = \sum_{i=0}^{n} f(x_i) g(x_i), \tag{2.32}$$

then (2.31) coincides with the normal equations (2.28) from the previous section. We remark that (2.32) is in fact not an inner product on the space of continuous functions, but it is an inner product on the space of polynomials of degree less than or equal to n.

In practice, the matrix and right-hand side of the linear system (2.31) can usually not be calculated exactly, because the inner product $\langle \bullet, \bullet \rangle$ is defined through an integral; see (2.33) in the next section.

Remark 2.3. Rewriting the normal equations (2.31) in terms of $\hat{u}$ we obtain

$$\langle \hat{u} - u, \varphi_0 \rangle = 0, \qquad \ldots, \qquad \langle \hat{u} - u, \varphi_m \rangle = 0.$$

Therefore, the optimal approximation $\hat{u} \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}$ satisfies

$$\forall v \in \mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}, \qquad \langle \hat{u} - u, v \rangle = 0.$$

This shows that the optimal approximation $\hat{u}$, in the sense of the norm $\|\bullet\|$, is the orthogonal projection of u onto $\mathrm{Span}\{\varphi_0, \ldots, \varphi_m\}$.

2.2.3 Orthogonal polynomials

The Gram matrix G in (2.31) is equal to the identity matrix when the basis functions are orthonormal for the inner product considered. In this case, the solution to the linear system is

$$\alpha_i = \langle u, \varphi_i \rangle, \qquad i = 0, \ldots, m,$$

and so the best approximation $\hat{u}$ (for the norm induced by the inner product considered!) is simply given by

$$\hat{u} = \sum_{i=0}^{m} \langle u, \varphi_i \rangle \varphi_i.$$

The coefficients $\langle u, \varphi_i \rangle$ of the basis functions in this expansion are called *Fourier coefficients*. Given a finite dimensional subspace $\mathcal{S}$ of the space of continuous functions, an orthonormal basis can be constructed via the Gram–Schmidt process. In this section, we focus on the particular case where $\mathcal{S} = \mathbf{P}(n)$ – the subspace of polynomials of degree less than or equal to n. Another widely used approach, which we do not explore in this course, is to use trigonometric basis functions. We also assume that the inner product (2.29) is of the form

$$\langle f, g \rangle = \int_a^b f(x) g(x) w(x) \, \mathrm{d}\, x, \tag{2.33}$$

where $w(x)$ is a given nonnegative weight function such that

$$\int_a^b w(x) \, \mathrm{d}\, x > 0.$$

Let $\varphi_0(x), \varphi_1(x), \varphi_2(x) \ldots$ denote the orthonormal polynomials obtained by applying the Gram–Schmidt procedure to the monomials $1, x, x^2, \ldots$. These depend in general on the weight $w(x)$ and on the interval $[a, b]$. A few of the popular classes of orthogonal polynomials are presented in the table below:

Name	$w(x)$	$[a, b]$
Legendre	1	$[-1, 1]$
Chebyshev	$\frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$
Hermite	$\exp\left(-\frac{x^2}{2}\right)$	$[-\infty, \infty]$
Laguerre	e^{-x}	$[0, \infty]$

Orthogonal polynomials have a rich structure, and in the rest of this section we prove some of their key properties, one of which will be very useful in the context of numerical integration in Chapter 3. We begin by showing that orthogonal polynomials have distinct real roots.

Proposition 2.8. *Assume for simplicity that $w(x) > 0$ for all $x \in [a, b]$, and let $\varphi_0, \varphi_1, \ldots$ denote orthonormal polynomials of increasing degree for the inner product (2.33). Then for all $n \in \mathbf{N}$, the polynomial φ_n has n distinct roots in the open interval (a, b).*

Proof. Reasoning by contradiction, we assume that φ_n changes sign at only $k < n$ points of the open interval (a, b), which we denote by $x_1, \ldots, x_k$. Then

$$\varphi_n(x) \times (x - x_0)(x - x_1) \ldots (x - x_k)$$

is either everywhere nonnegative or everywhere nonpositive over $[a, b]$. But then

$$\int_a^b \varphi_n(x) \times (x - x_1) \ldots (x - x_k)\, w(x) \,\mathrm{d}\, x$$

is nonzero, which is a contradiction because the product $(x - x_1) \ldots (x - x_k)$ is a polynomial of degree k, which is orthogonal to φ_n by assumption. Indeed, being orthogonal to $\varphi_0, \ldots, \varphi_{n-1}$, the polynomial φ_n is also orthogonal to any linear combination of these polynomials. □

Next, we show that orthogonal polynomials satisfy a three-term recurrence relation.

Proposition 2.9. *Assume that $\varphi_0, \varphi_1, \ldots$ are orthonormal polynomials for some inner product of the form (2.33) such that φ_i is of degree i. Then*

$$\forall n \in \{1, 2, \ldots\}, \qquad \alpha_{n+1}\varphi_{n+1}(x) = (x - \beta_n)\varphi_n(x) - \alpha_n \varphi_{n-1}(x), \tag{2.34}$$

where

$$\alpha_n = \left\langle x\varphi_n, \varphi_{n-1} \right\rangle, \qquad \beta_n = \left\langle x\varphi_n, \varphi_n \right\rangle.$$

In addition, $\alpha_1\varphi_1(x) = (x - \beta_0)\varphi_0(x)$.

Proof. Since $x\varphi_n(x)$ is a polynomial of degree $n + 1$, it may be decomposed as

$$x\varphi_n(x) = \sum_{i=0}^{n+1} \gamma_{n,i}\varphi_i(x). \tag{2.35}$$

Taking the inner product of both sides of this equation with φ_i and employing the orthonormality assumption, we obtain an expression for the coefficients:

$$\gamma_{n,i} = \left\langle x\varphi_n, \varphi_i \right\rangle.$$

From the expression (2.33) of the inner product, it is clear that $\left\langle x\varphi_n, \varphi_i \right\rangle = \left\langle \varphi_n, x\varphi_i \right\rangle$. Since $x\varphi_i$ is a polynomial of degree $i + 1$ and φ_n is orthogonal to all polynomials of degree strictly less than n, we deduce that $\gamma_{n,i} = 0$ if $i < n - 1$. Consequently, we can rewrite the right-hand side of (2.35) as a sum involving only three terms

$$x\varphi_n(x) = \left\langle x\varphi_n, \varphi_{n-1}\right\rangle \varphi_{n-1}(x) + \left\langle x\varphi_n, \varphi_n\right\rangle \varphi_n(x) + \left\langle x\varphi_n, \varphi_{n+1}\right\rangle \varphi_{n+1}(x). \tag{2.36}$$

Since $\left\langle x\varphi_n, \varphi_{n+1}\right\rangle = \left\langle x\varphi_{n+1}, \varphi_n\right\rangle$, we obtain the statement after rearranging. □

Remark 2.4. Notice that the polynomials in Proposition 2.9 are orthonormal by assumption, and so the coefficient α_{n+1} is just a normalization constant. We deduce that

$$\varphi_{n+1}(x) = \frac{(x-\beta_n)\varphi_n(x) - \alpha_n\varphi_{n-1}(x)}{\|(x-\beta_n)\varphi_n(x) - \alpha_n\varphi_{n-1}(x)\|},$$

which enables to calculate the orthogonal polynomials recursively.

2.2.4 Orthogonal polynomials and numerical integration: an introduction

Equation (2.36) may be rewritten in matrix form as follows:

$$\begin{pmatrix} x\varphi_0(x) \\ x\varphi_1(x) \\ x\varphi_2(x) \\ \vdots \\ x\varphi_{m-1}(x) \\ x\varphi_m(x) \end{pmatrix} = \begin{pmatrix} \beta_0 & \alpha_1 & & & & \\ \alpha_1 & \beta_1 & \alpha_2 & & & \\ & \alpha_2 & \beta_2 & \alpha_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \alpha_{m-1} & \beta_{m-1} & \alpha_m \\ & & & & \alpha_m & \beta_m \end{pmatrix} \begin{pmatrix} \varphi_0(x) \\ \varphi_1(x) \\ \varphi_2(x) \\ \vdots \\ \varphi_{m-1}(x) \\ \varphi_m(x) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \alpha_{m+1}\varphi_{m+1}(x) \end{pmatrix}.$$

Let T denote the matrix on the left-hand side of this equation, and let $r_0, \ldots, r_m$ denote the roots of φ_{m+1}. By Proposition 2.8, these are distinct and all belong to the interval (a, b). The second term on the right-hand side cancels out when x is a root of φ_{m+1}, and so

$$\forall r \in \left\{r_0, \ldots, r_m\right\}, \qquad \begin{pmatrix} r\varphi_0(r) \\ r\varphi_1(r) \\ \vdots \\ r\varphi_{m-1}(r) \\ r\varphi_m(r) \end{pmatrix} = \begin{pmatrix} \beta_0 & \alpha_1 & & & \\ \alpha_1 & \beta_1 & \alpha_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \alpha_{m-1} & \beta_{m-1} & \alpha_m \\ & & & \alpha_m & \beta_m \end{pmatrix} \begin{pmatrix} \varphi_0(r) \\ \varphi_1(r) \\ \vdots \\ \varphi_{m-1}(r) \\ \varphi_m(r) \end{pmatrix}.$$

In other words, for any root r of φ_{m+1}, the vector $\begin{pmatrix} \varphi_0(r) & \cdots & \varphi_m(r) \end{pmatrix}^T$ is an eigenvector of the matrix T, with associated eigenvalue equal to r. Since T is a symmetric matrix, the eigenvectors associated with distinct eigenvalues are orthogonal for the Euclidean inner product of $\mathbf{R}^{m+1}$, so given that the eigenvalues of T are distinct, we deduce that

$$\forall i \neq j, \qquad \sum_{i=0}^{m} \varphi_i(r_i)\varphi_i(r_j) = 0. \tag{2.37}$$

Let us construct the matrix

$$\mathsf{P} = \begin{pmatrix} \varphi_0(r_0) & \varphi_1(r_0) & \dots & \varphi_m(r_0) \\ \varphi_0(r_1) & \varphi_1(r_1) & \dots & \varphi_m(r_1) \\ \varphi_0(r_2) & \varphi_1(r_2) & \dots & \varphi_m(r_2) \\ \vdots & \vdots & \dots & \vdots \\ \varphi_0(r_m) & \varphi_1(r_m) & \dots & \varphi_m(r_m) \end{pmatrix}.$$

Equation (2.37) indicates that the rows of P are orthogonal, and so the matrix $\mathsf{D} = \mathsf{P}\mathsf{P}^T$ is diagonal with elements given by

$$d_{ii} = \sum_{j=0}^{m} |\varphi_j(r_i)|^2, \qquad i = 0, \dots, m.$$

(Here we start counting the rows from 0 for convenience.) Since $\mathsf{P}\mathsf{P}^T\mathsf{D}^{-1} = \mathsf{I}$, we deduce that the inverse of P is given by $\mathsf{P}^{-1} = \mathsf{P}^T\mathsf{D}^{-1}$. Consequently,

$$\mathsf{P}^T\mathsf{D}^{-1}\mathsf{P} = \mathsf{P}^{-1}\mathsf{P} = \mathsf{I},$$

which means that the *columns* of P are orthonormal for the inner product $(\boldsymbol{x}, \boldsymbol{y}) \mapsto \boldsymbol{x}^T\mathsf{D}^{-1}\boldsymbol{y}$. In other words, the polynomials $\varphi_1, \dots, \varphi_m$ are orthonormal for the inner product

$$\begin{aligned} \langle \bullet, \bullet \rangle_{m+1} \colon \mathbf{P}(m) \times \mathbf{P}(m) &\to \mathbf{R}; \\ (p, q) &\mapsto \sum_{i=0}^{m} \frac{p(r_i)q(r_i)}{d_{ii}}. \end{aligned}$$

We have thus shown that, if $\varphi_0, \varphi_1, \varphi_2, \dots$ is a family of orthonormal polynomials for an inner product $\langle \bullet, \bullet \rangle$, then these are also orthonormal for the inner product $\langle \bullet, \bullet \rangle_{m+1}$. We reformulate our findings in the following result where, since m was arbitrary in the previous reasoning, we add a superscript to indicate when the quantities involved depend on m.

Theorem 2.10. *Orthonormal polynomials $\varphi_0, \dots, \varphi_m$ for the inner product*

$$\langle f, g \rangle = \int_a^b f(x)g(x)w(x) \,\mathrm{d}\, x$$

are also orthonormal for the inner product

$$\langle f, g \rangle_{m+1} = \sum_{i=0}^{m} f\Big(r_i^{(m+1)}\Big) g\Big(r_i^{(m+1)}\Big) w_i^{(m+1)},$$

where $r_0^{(m+1)}, \dots, r_m^{(m+1)}$ are the roots of φ_{m+1} and the weights $w_i^{(m+1)}$ are given by

$$w_i^{(m+1)} = \frac{1}{\sum_{j=0}^{m} \left|\varphi_j\Big(r_i^{(m+1)}\Big)\right|^2}, \qquad i = 0, \dots, m.$$

As an immediate corollary, we deduce that

$$\forall (p, q) \in \mathbf{P}(m) \times \mathbf{P}(m), \qquad \langle p, q \rangle = \langle p, q \rangle_{m+1}, \tag{2.38}$$

Indeed, denoting by $p = \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m$ and $q = \beta_0\varphi_0 + \cdots + \beta_m\varphi_m$ the expansions of the polynomials p and q in the orthonormal basis, we have

$$\begin{aligned}\langle p, q\rangle &= \langle \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m, \beta_0\varphi_0 + \cdots + \beta_m\varphi_m\rangle \\ &= \sum_{i=0}^{m}\sum_{j=0}^{m} \alpha_i\beta_j\langle\varphi_i, \varphi_j\rangle = \alpha_0\beta_0 + \cdots + \alpha_m\beta_m = \sum_{i=0}^{m}\sum_{j=0}^{m} \alpha_i\beta_j\langle\varphi_i, \varphi_j\rangle_{m+1} \\ &= \langle \alpha_0\varphi_0 + \cdots + \alpha_m\varphi_m, \beta_0\varphi_0 + \cdots + \beta_m\varphi_m\rangle_{m+1} = \langle p, q\rangle_{m+1}.\end{aligned}$$

To conclude this section, we prove the following statement, which is another consequence of Theorem 2.10 and has applications to numerical integration.

Theorem 2.11. *It holds that*

$$\forall p \in \mathbf{P}(2m+1), \qquad \int_a^b p(x)\, w(x)\, \mathrm{d}\, x = \sum_{i=0}^{m} p\Big(r_i^{(m+1)}\Big) w_i^{(m+1)}. \tag{2.39}$$

Proof. Taking $q = 1$ in (2.38) and employing the definitions of $\langle \bullet, \bullet\rangle$ and $\langle \bullet, \bullet\rangle_{m+1}$, we have that (2.38) is satisfied for any $p \in \mathbf{P}(m)$. Next, any polynomial $p \in \mathbf{P}(2m+1)$ may be decomposed as $p(x) = \varphi_{m+1}(x)q(x) + \rho(x)$, for some polynomial q of degree m (the quotient of the polynomial division of p by φ_{m+1}) and some polynomial ρ of degree lower than or equal to m (the remainder of the polynomial division). Therefore, since (2.39) was already shown to hold for polynomials of degree up to m, we obtain

$$\begin{aligned}\int_a^b p(x)w(x)\, \mathrm{d}\, x &= \int_a^b \varphi_{m+1}(x)q(x)w(x)\, \mathrm{d}\, x + \int_a^b \rho(x)w(x)\, \mathrm{d}\, x \\ &= 0 + \int_a^b \rho(x)w(x)\, \mathrm{d}\, x = 0 + \sum_{i=0}^{m} \rho(r_i)w_i \\ &= \sum_{i=0}^{m} \varphi_{m+1}(r_i)q(r_i)w_i + \sum_{i=0}^{m} \rho(r_i)w_i = \sum_{i=0}^{m} p(r_i)w_i,\end{aligned}$$

where we dropped the $^{(m+1)}$ superscript for conciseness and we used, in the penultimate inequality, the fact that $r_0, \ldots, r_m$ are the roots of the polynomial φ_{m+1}. □

Since the left-hand side of (2.39) is an integral and the right-hand side is a sum, we have just constructed an integration formula, which enjoys a very nice property: it is exact for polynomials of degree up to $2m+1$! A formula of this type is called a *quadrature formula*, with $m+1$ *nodes* $r_0^{(m+1)}, \ldots, r_m^{(m+1)}$ and associated *weights* $w_0^{(m+1)}, \ldots, w_m^{(m+1)}$. Note that the nodes and weights of the quadrature depend on the weight $w(x)$ and on the degree m. We will revisit this subject in Chapter 3.

2.3 Exercises

Exercise 2.1. *Find the polynomial $p(x) = ax + b$ (a straight line) that goes through the points (x_0, u_0) and (x_1, u_1).*

Exercise 2.2. *Find the polynomial $p(x) = ax^2 + bx + c$ (a parabola) that goes through the points $(0, 1)$, $(1, 3)$ and $(2, 7)$.*

Exercise 2.3. *Prove the following recurrence relation for Chebyshev polynomials:*

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x), \qquad i = 1, 2, \ldots.$$

Exercise 2.4. *Show by recursion that*

$$[u_0, u_1, \ldots, u_n] = \sum_{j=0}^{n} \frac{u_j}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)}. \tag{2.40}$$

Deduce from this identity that

$$[u_0, u_1, \ldots, u_n] = [u_{\sigma_1}, u_{\sigma_2}, \ldots, u_{\sigma_n}],$$

for any permutation $\boldsymbol{\sigma}$ of $(0, 1, 2, \ldots, n)$.

Solution. The first statement (2.40) is clear when $n = 0$. Reasoning by induction, we assume that the statement is true up to $n-1$ and prove that it then also holds for n. Using the definition (2.10) and the induction hypothesis, we obtain that

$$\begin{aligned}[u_0, u_1, \ldots, u_n] &= \frac{[u_1, \ldots, u_n] - [u_0, \ldots, u_{n-1}]}{x_n - x_0} \\ &= \frac{1}{x_n - x_0}\left(\sum_{j=\mathbf{1}}^{n} \frac{u_j}{\prod_{k\in\{\mathbf{1},\ldots,n\}\setminus\{j\}}(x_j - x_k)} - \sum_{j=0}^{\mathbf{n-1}} \frac{u_j}{\prod_{k\in\{0,\ldots,\mathbf{n-1}\}\setminus\{j\}}(x_j - x_k)}\right)\end{aligned}$$

Rewriting the fractions with a common denominator leads to

$$[u_0, u_1, \ldots, u_n] = \frac{1}{x_n - x_0}\sum_{j=0}^{n} \frac{u_j\big((x_j - x_0) - (x_j - x_n)\big)}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)} = \sum_{j=0}^{n} \frac{u_j}{\prod_{k\in\{0,\ldots,n\}\setminus\{j\}}(x_j - x_k)},$$

which concludes the proof of the first statement. The second statement then follows immediately, because the right-hand side of (2.40) is invariant under permutations. △

Exercise 2.5. *Using the Gregory–Newton formula, find an expression for*

$$\sum_{i=1}^{n} i^4.$$

Exercise 2.6. *Let $(f_0, f_1, f_2, \ldots) = (1, 1, 2, \ldots)$ denote the Fibonacci sequence. Prove that there does not exist a polynomial p such that*

$$\forall n \in \mathbf{N}, \qquad f_n = p(n). \tag{2.41}$$

Solution. Assume by contradiction that $p\colon \mathbf{R} \to \mathbf{R}$ is a polynomial such that (2.41) is satisfied, and let n be the degree of this polynomial. Then it holds that $\Delta^{n+1}p = 0$ (2.5), where both sides are

viewed as functions from $\mathbf{R}$ to $\mathbf{R}$. On the other hand, since $p(n) = f_n$ for all $n \in \mathbf{N}$, we can calculate explicitly the values of taken by the function $\Delta^m p$ when evaluated at all the natural numbers, for all $m \in \mathbf{N}$. We collate a few values in the following table.

n	0	1	2	3	4	5	6
$\Delta^0 p(n)$	**1**	**1**	**2**	**3**	**5**	**8**	**13**
$\Delta^1 p(n)$	0	**1**	**1**	**2**	**3**	**5**	**8**
$\Delta^2 p(n)$	1	0	**1**	**1**	**2**	**3**	**5**
$\Delta^3 p(n)$	-1	1	0	**1**	**1**	**2**	**3**

It appears from these calculations that the Fibonacci sequence is shifted one position to the right with each additional application of Δ. In other words, our calculations suggest that

$$\forall (m, n) \in \mathbf{N} \times \mathbf{N}, \qquad \Delta^m p(m+n) = f_n, \tag{2.42}$$

which is a contradiction. To conclude, let us prove (2.42) rigorously. This equation is obvious for $m = 0$ by assumption. Now, reasoning by contradiction, we assume that (2.42) is true up to m. Then by definition of the difference operator Δ, we have

$$\begin{aligned} \Delta^{m+1} p(m+n+1) &= \Delta^m p(m+n+2) - \Delta^m p(m+n+1) \\ &= f_{n+2} - f_{n+1} = f_n. \end{aligned}$$

Here, we used the induction hypothesis (2.42) in the second equality, and the definition of the Fibonacci series in the third one. △

Exercise 2.7. *Using the Gregory–Newton formula, show that*

$$\forall n \in \mathbf{N}, \qquad 2^n = 1 + n + \frac{n^{\underline{2}}}{2!} + \frac{n^{\underline{3}}}{3!} + \frac{n^{\underline{4}}}{4!} + \cdots \tag{2.43}$$

Solution. Equation (2.43) is a particular case of the following more general statement: for any function $f \in \mathbf{R} \to \mathbf{R}$, it holds that

$$\forall n \in \mathbf{N}, \qquad f(n) = f(0) + \Delta f(0) n + \Delta^2 f(0) \frac{n^{\underline{2}}}{2!} + \Delta^3 f(0) \frac{n^{\underline{3}}}{3!} + \Delta^4 f(0) \frac{n^{\underline{4}}}{4!} + \cdots \tag{2.44}$$

In order to show this equation, it is sufficient to prove that for any $n_* \in \mathbf{N}$, the two sides of (2.44) coincide for every $n \in \{0, \ldots, n_*\}$. Since $n^{\underline{p}} = 0$ for all $n \in \{0, \ldots, p-1\}$ by definition (2.4) of the falling powers, the right-hand side of (2.44) coincides for all $n \in \{0, \ldots, n_*\}$ with

$$g(n) = f(0) + \Delta f(0) n + \Delta^2 f(0) \frac{n^{\underline{2}}}{2!} + \cdots + \Delta^{n_*} f(0) \frac{n^{\underline{n_*}}}{n_*!}.$$

We recognize on the right-hand side Newton's expression of the interpolating polynomial through the points $\big(0, f(0)\big), \ldots, \big(n_*, f(n_*)\big)$, and so $g(n) = f(n)$ for all $n \in \{0, \ldots, n_*\}$, which concludes the proof. △

Remark 2.5. Remarkably, equation (2.43) holds in fact for *any* $n \in \mathbf{R}_{>0}$. However, showing this more general statement is beyond the scope of this course.

Exercise 2.8. *Prove Theorem 2.7.*

Exercise 2.9. *Show that the matrix* G *in (2.30) is positive definite if the basis functions* $\varphi_0, \ldots, \varphi_m$ *are linearly independent.*

Exercise 2.10. *Write a Julia code for interpolating the following function using a polynomial of degree 20 over the interval* $[-1, 1]$.

$$f(x) = \tanh\left(\frac{x+1/2}{\varepsilon}\right) + \tanh\left(\frac{x}{\varepsilon}\right) + \tanh\left(\frac{x-1/2}{\varepsilon}\right), \qquad \varepsilon = .01.$$

Use equidistant and then Chebyshev nodes, and compare the two approaches in terms of accuracy. Plot the function f *together with the approximating polynomials.*

Exercise 2.11. *Write from scratch a function to obtain the polynomial interpolating the data points*

$$(x_0, u_0), \ldots, (x_n, u_n).$$

Your function should return the values taken by the interpolating polynomial when evaluated at the points $X_0, \ldots, X_m$. *You may use the following code to test your function*

```
import Plots
function interp(X, x, u)
    # Your code comes here
end
n, m = 10, 100
f(t) = cos(2π * t)
x = LinRange(0, 1, n)
X = LinRange(0, 1, m)
u = f.(x)
U = interp(X, x, u)
Plots.plot(X, f.(X), label="Original function")
Plots.plot!(X, U, label="Interpolation")
Plots.scatter!(x, u, label="Data")
```

Exercise 2.12. *We wish to use interpolation to approximate the following parametric function, called an* epitrochoid*:*

$$x(\theta) = (R+r)\cos\theta + d\cos\left(\frac{R+r}{r}\theta\right) \tag{2.45}$$

$$y(\theta) = (R+r)\sin\theta - d\sin\left(\frac{R+r}{r}\theta\right), \tag{2.46}$$

with $R = 5$, $r = 2$ *and* $d = 3$, *and for* $\theta \in [0, 4\pi]$. *Write a Julia program to interpolate* $x(\theta)$ *and* $y(\theta)$ *using 40 equidistant points. Use the* `BigFloat` *format in order to reduce the impact*

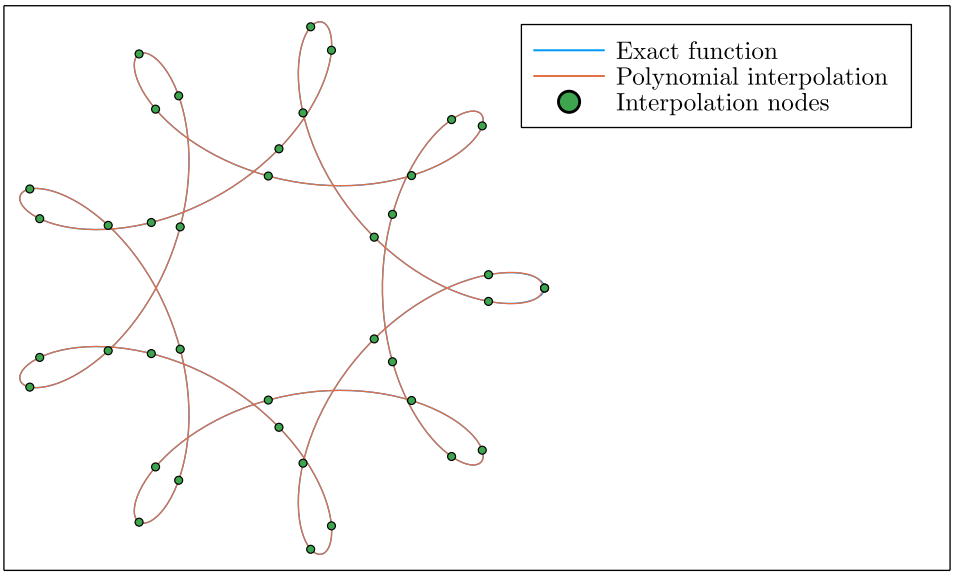

Figure 2.5: Solution for Exercise 2.12.

of round-off errors. After constructing the polynomial interpolations $\hat{x}(\theta)$ and $\hat{y}(\theta)$, plot the parametric curve $\theta \mapsto \big(\hat{x}(\theta), \hat{y}(\theta)\big)$. Your plot should look similar to Figure 2.5.

Exercise 2.13 (Solving the Laplace equation using a spectral method)**.** *The classical Laplace equation with homogeneous Dirichlet boundary conditions in dimension 1 reads*

$$\text{Find } u \in C^2\big([0,1]\big) \text{ such that } \quad \begin{cases} -u''(x) = f(x) & \forall x \in (0,1), \\ u(0) = u(1) = 0. \end{cases} \tag{2.47}$$

Our goal in this exercise is to approximate the exact solution $u(x)$ using interpolation. Specifically, we propose to proceed in two steps:

- *Interpolate the right-hand side using a polynomial with equidistant nodes. That is, find a polynomial $\hat{f} \in \mathbf{P}(n)$ such that*

$$\forall i \in \{0, \dots, n\}, \qquad \hat{f}(x_i) = f(x_i), \qquad x_i = \frac{i}{n}.$$

- *Solve (2.47) with $\hat{f}$ instead of f. Since $\hat{f}$ is a polynomial, this can be achieved analytically.*

Implement this program in the case where

$$f(x) = \exp\big(\sin(2\pi x)\big)\cos(2\pi x)^2 - \exp\big(\sin(2\pi x)\big)\sin(2\pi x),$$

and compare for various values of n the approximate solution you obtain with the exact solution to (2.47), which is given by $u(x) = (2\pi)^{-2}\exp\Big(\big(\sin(2\pi x)\big) - 1\Big)$ in this case.

Exercise 2.14 (Modeling the vapor pressure of mercury)**.** *The dataset loaded through the following Julia commands contains data on the vapor pressure of mercury as a function of the temperature.*

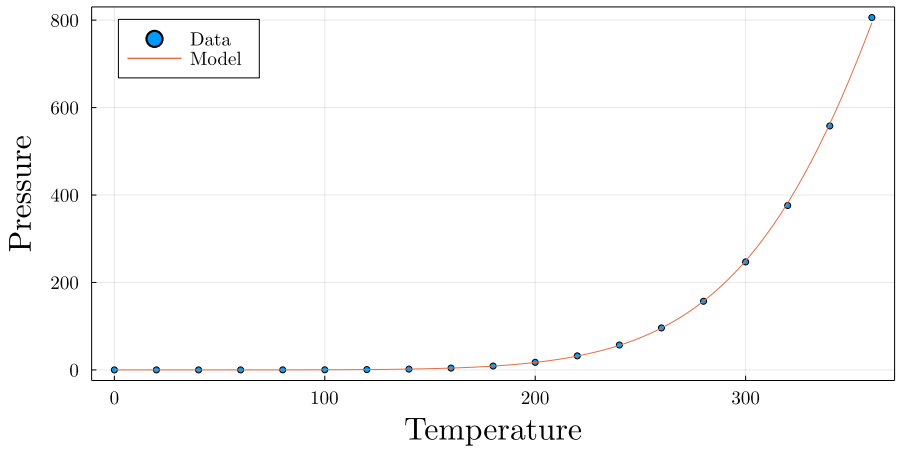

Figure 2.6: Solution for Exercise 2.14.

```
import RDatasets
data = RDatasets.dataset("datasets", "pressure")
```

Find a low-dimensional mathematical model of the form

$$p(T) = \exp\bigl(\alpha_0 + \alpha_1 T + \alpha_2 T^2 + \alpha_3 T^3\bigr) \tag{2.48}$$

for the pressure as a function of the temperature. Plot the approximation together with the data. An example solution is given in Figure 2.6.

Exercise 2.15. *Let* $u\colon [0, 2\pi] \to \mathbf{R}$ *and*

$$x_k = \left(\frac{2k\pi}{2n+1}\right), \qquad k = 0, \ldots, 2n. \tag{2.49}$$

We wish to interpolate u *at these nodes using complex exponentials:*

$$\hat{u} = \sum_{k=-n}^{n} a_k \mathrm{e}^{\mathrm{i}\,kx}.$$

Write a function

```
function fourier_interpolate(u, x, X)
    # Your code comes here ...
end
```

which takes three arguments:

- `u` *is the function to interpolate;*
- `x` *are the interpolation nodes, given by (2.49) in the test code below; you can assume that this array contains an odd number of elements.*
- `X` *is a one-dimensional array of values on the* x *axis.*

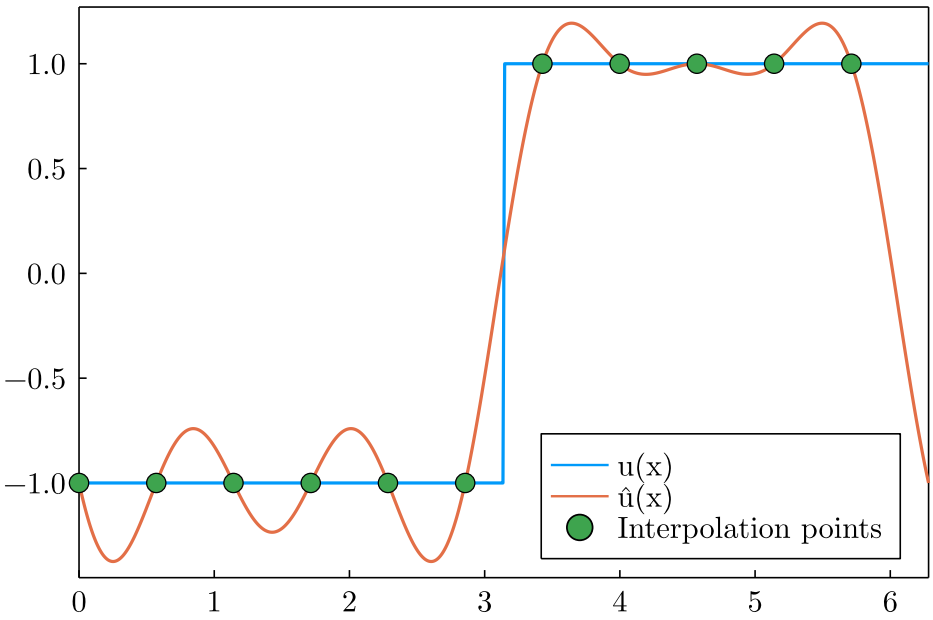

Figure 2.7: Example solution for Exercise 2.15.

The function should return a one-dimensional array containing the values that $\hat{u}$ takes when evaluated at the points contained in `X`*. You can use the following code to test your function:*

```
import Plots
n, m = 5, 1000
x = 2π/(2n+1) * (0:2n)
X = 2π/m * (0:m)

u(x) = sign(x - π)
# u(x) = exp(sin(x) + cos(5x))
# u(x) = x^2 * (x - 2π)^2 / π^4

@time U = fourier_interpolate(u, x, X)
Plots.plot(X, u.(X), label="u(x)", legend=:bottomright)
Plots.plot!(X, U, label="û(x)")
Plots.scatter!(x, u.(x), label="Interpolation points")
Plots.xlims!(0, 2π)
```

An example of the output plot is illustrated in Figure 2.7.

2.4 Discussion and bibliography

A comprehensive study of approximation theory would require to cover the L^∞ setting as well as other functional settings. A pillar of L^∞ approximation theorem is Chebyshev's equioscillation theorem, which we alluded to in Remark 2.1. An excellent introductory reference on approximation theory is [8] (in French). See also [10, Chapter 10] and the references therein.