11 Polynomial and Piecewise Polynomial Interpolation

Let $f$ be a function, which is only known at the real nodes $x_1, x_2, \ldots, x_n$ i.e., all we know about the function $f$ are its values $y_j = f(x_j)$, $j = 1, 2, \ldots, n$. For instance, we may have obtained these values through measurements and now would like to determine approximations of $f(x)$ for other values of $x$.

**Example 11.1**

Assume that we need to evaluate $\cos(\pi/6)$, but the trigonometric function-key on our calculator is broken and we do not have access to a computer. We recall that $\cos(0) = 1$, $\cos(\pi/4) = 1/\sqrt{2}$, and $\cos(\pi/2) = 0$. How can we use this information about the cosine function to determine an approximation of $\cos(\pi/6)$? □

**Example 11.2**

Let $x$ represent time (in hours) and $f(x)$ be the amount of rain falling at time $x$. Assume that $f(x)$ is measured once an hour at a weather station. We would like to determine the total amount of rain fallen during a 24-hour period, i.e., we would like to compute

$$\int_{0}^{24} f(x) \, dx.$$ 

How can we determine an estimate of this integral? □

**Example 11.3**

Let $f(x)$ represent the position of a car at time $x$ and assume that we know $f(x)$ at the times $x_1, x_2, \ldots, x_n$. How can we determine the velocity at time $x$? Can we also find out the acceleration? □

Interpolation by polynomials or piecewise polynomials provide approaches to solving the problems in the above examples. We first discuss polynomial interpolation and then turn to interpolation by piecewise polynomials. Polynomial least-squares approximation is another technique for computing a polynomial that approximates given data. Least-squares approximation was discussed and illustrated in Lecture 6 and will be considered further in this lecture.
11.1 Polynomial interpolation

Given \( n \) distinct real nodes \( x_1, x_2, \ldots, x_n \) and associated real numbers \( y_1, y_2, \ldots, y_n \), determine the polynomial \( p(x) \) of degree at most \( n - 1 \), such that

\[
p(x_j) = y_j, \quad j = 1, 2, \ldots, n.
\]

(1)

The polynomial is said to **interpolate** the values \( y_j \) at the nodes \( x_j \), and is referred to as the **interpolation polynomial**.

**Example 11.4**

Let \( n = 1 \). Then the interpolation polynomial reduces to the constant \( y_1 \). When \( n = 2 \), the interpolation polynomial is linear and can be expressed as

\[
p(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1} (x - x_1).
\]

**Example 11.1 cont’d**

We may seek to approximate \( \cos(\pi/6) \) by first determining the polynomial \( p \) of degree at most 2, which interpolates \( \cos(x) \) at \( x = 0, x = \pi/4, \) and \( x = \pi/2 \), and then evaluating \( p(\pi/6) \). These computations yield \( p(\pi/6) = 0.851 \), which is fairly close to the exact value \( \cos(\pi/6) = \sqrt{3}/2 \approx 0.866 \).

Before dwelling more on applications of interpolation polynomials, we have to establish that they exist and investigate whether they are unique. We will consider several representations of interpolation polynomials starting with the power form

\[
p(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1},
\]

(2)

which may be the most natural representation. This polynomial is of degree at most \( n - 1 \). It has \( n \) coefficients \( a_1, a_2, \ldots, a_n \). We would like to determine these coefficients so that the \( n \) interpolation conditions (1) are satisfied. This gives the equations

\[
a_1 + a_2 x_j + a_3 x_j^2 + \cdots + a_n x_j^{n-1} = y_j, \quad j = 1, 2, \ldots, n.
\]

They can be expressed as a linear system of equations with a Vandermonde matrix,

\[
\begin{bmatrix}
1 & x_1 & x_1^2 & \cdots & x_1^{n-2} & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-2} & x_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \\
1 & x_n & x_n^2 & \cdots & x_n^{n-2} & x_n^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_{n-1} \\
a_n
\end{bmatrix}
= \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_{n-1} \\
y_n
\end{bmatrix}.
\]

(3)
As noted in Section 7.5, Vandermonde matrices are nonsingular when the nodes $x_j$ are distinct. This secures the existence of a unique interpolation polynomial.

The representation (2) of the interpolation polynomial $p(x)$ in terms of the powers of $x$ is convenient for many applications, because this representation easily can be integrated or differentiated. Moreover, the polynomial (2) easily can be evaluated by nested multiplication without explicitly computing the powers $x^j$. For instance, pulling out common powers of $x$ from the terms of a polynomial of degree three gives

$$p(x) = a_1 + a_2x + a_3x^2 + a_4x^3 = a_1 + (a_2 + (a_3 + a_4x)x)x.$$  

The right-hand side can be evaluated without explicitly form the powers $x^2$ and $x^3$. Similarly, polynomials of degree at most $n - 1$ can be expressed as

$$p(x) = a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1} = a_1 + (a_2 + (a_3 + (\ldots + a_{n-1})x\ldots)x.  \tag{4}$$

The right-hand side can be evaluated rapidly; only $O(n)$ arithmetic floating point operations (flops) are required; see Exercise 11.2. Here $O(n)$ stands for an expression bounded by $cn$ as $n \to \infty$, where $c > 0$ is a constant independent of $n$.

However, Vandermonde matrices generally are severely ill-conditioned. This is illustrated in Exercise 11.3. When the function values $y_j$ are obtained by measurements, and therefore are contaminated by measurement errors, the ill-conditioning implies that the computed coefficients $a_j$ may differ significantly from the coefficients that would have been obtained with error-free data. Moreover, round-off errors introduced during the solution of the linear system of equations (3) also can give rise to a large propagated error in the computed coefficients. We are therefore interested in investigating other polynomial bases than the power basis.

The Lagrange polynomials of degree $n - 1$ associated with the $n$ distinct points $x_1, x_2, \ldots, x_n$ are given by

$$\ell_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j}, \quad k = 1, 2, \ldots, n.  \tag{5}$$

These polynomials are of degree $n - 1$; they form a basis for all polynomials of degree at most $n - 1$.

It is easy to verify that the Lagrange polynomials satisfy

$$\ell_k(x_j) = \begin{cases} 
1, & k = j, \\
0, & k \neq j.
\end{cases} \tag{6}$$

This property makes it possible to determine the interpolation polynomial without solving a linear system of equations. It follows from (6) that the polynomial

$$p(x) = \sum_{k=1}^{n} y_k \ell_k(x)  \tag{7}$$
satisfies (1). Moreover, since the Lagrange polynomials (5) are of degree \( n - 1 \), the interpolation polynomial (7) is of degree at most \( n - 1 \). It is of degree strictly smaller than \( n - 1 \) when the leading coefficients of the polynomials (5) cancels in (7); see Exercise 11.4 for an example.

We refer to the expression (7) as the interpolation polynomial in Lagrange form. This representation establishes the existence of an interpolation polynomial without using properties of Vandermonde matrices. Unicity also can be shown without using Vandermonde matrices: assume that there are two polynomials \( p(x) \) and \( q(x) \) of degree at most \( n - 1 \), such that

\[
p(x_j) = q(x_j) = y_j, \quad 1 \leq j \leq n.
\]

Then the polynomial \( r(x) = p(x) - q(x) \) is of degree at most \( n - 1 \) and vanishes at the \( n \) distinct nodes \( x_j \). According to the Fundamental Theorem of Algebra, a polynomial of degree \( n - 1 \) has at most \( n - 1 \) distinct zeros or vanishes identically. Hence, \( r(x) \) vanishes identically, and it follows that \( p \) and \( q \) are the same polynomial. Thus, the interpolation polynomial is unique.

We remark that this proof based on Lagrange polynomials of the existence and unicity of the interpolation polynomial \( p \) of degree at most \( n - 1 \) that satisfies (1) shows that the Vandermonde matrix in (3) is nonsingular for any set of \( n \) distinct nodes \( \{x_j\}_{j=1}^n \). Assume that the Vandermonde matrix is singular. Then, for a right-hand side \( [y_1, y_2, \ldots, y_n]^T \) in (3) in the range of the Vandermonde matrix, the linear system of equations (3) has infinitely many solutions. This means that there are infinitely many interpolation polynomials of degree at most \( n - 1 \). But we have established that the interpolation polynomial of degree at most \( n - 1 \) is unique. It follows that the Vandermonde matrix is nonsingular.

While the Lagrange form of the interpolation polynomial (7) can be written up without any computations, there are, nevertheless, some drawback of the this representation. The antiderivative and derivative of the Lagrange form are cumbersome to determine. Therefore this representation is not well suited for polynomial interpolation problems that require the derivative or integral of the interpolation polynomial be evaluated; see Exercises 11.10 and 11.11 for examples. Moreover, the evaluation of the representation (7) at \( x \) by determining the value of each Lagrange polynomial (5) is much more expensive than evaluating the interpolation polynomial in power form (2) using nested multiplication (4). We noted above that the latter requires \( O(n) \) flops to evaluate. The evaluation of each Lagrange polynomial \( \ell_k(x) \) at a point \( x \) also requires \( O(n) \) flops. This suggests that the evaluation of the sum (7) of \( n \) Lagrange polynomials requires \( O(n^2) \) flops. We will now see how the latter flop count can be reduced by expressing the Lagrange polynomials in a different way.

Introduce the nodal polynomial

\[
\ell(x) = \prod_{j=1}^{n} (x - x_j)
\]
and define the weights
\[ w_k = \frac{1}{\prod_{j=1, j \neq k}^{n} (x_k - x_j)}. \] (8)

Then, for \( x \) different from all interpolation points, the Lagrange polynomials can be written as
\[ \ell_k(x) = \ell(x) \frac{w_k}{x - x_k}, \quad k = 1, 2, \ldots, n. \]

All terms in the sum (7) contain the factor \( \ell(x) \), which is independent of \( k \). We therefore can move this factor outside the sum and obtain
\[ p(x) = \ell(x) \sum_{k=1}^{n} y_k \frac{w_k}{x - x_k}. \] (9)

We noted above that the interpolation polynomial is unique. Therefore, interpolation of the constant function \( f(x) = 1 \), which is a polynomial, gives the interpolation polynomial \( p(x) = 1 \). Since \( f(x) = 1 \), we have \( y_k = 1 \) for all \( k \), and the expression (9) simplifies to
\[ 1 = \ell(x) \sum_{k=1}^{n} \frac{w_k}{x - x_k}. \]

It follows that
\[ \ell(x) = \frac{1}{\sum_{k=1}^{n} \frac{w_k}{x - x_k}}. \]

Finally, substituting this expression into (9) yields
\[ p(x) = \frac{\sum_{k=1}^{n} y_k \frac{w_k}{x - x_k}}{\sum_{k=1}^{n} \frac{w_k}{x - x_k}}. \] (10)

This formula is known as the barycentric representation of the interpolation polynomial, or simply as the interpolation polynomial in barycentric form. It requires that the weights be computed, e.g., by using the definition (8). This requires \( \mathcal{O}(n^2) \) arithmetic floating point operations. Given the weights, \( p(x) \) can be evaluated at any point \( x \) in only \( \mathcal{O}(n) \) arithmetic floating point operations. Exercises 11.5 and 11.6 are concerned with these computations. The barycentric representation is particularly attractive when the interpolation polynomial is to be evaluated many times.
The representation (10) can be shown to be quite insensitive to round-off errors and therefore can be used to represent polynomials of high degree, provided that overflow and underflow is avoided during the computation of the weights \( w_k \). This easily can be achieved by rescaling all the weights when necessary; note that the formula (10) allows all weights to be multiplied by an arbitrary nonzero constant.

### 11.2 The approximation error

Let the nodes \( x_j \) be distinct in the real interval \([a, b]\), and let \( f(x) \) be an \( n \) times differentiable function in \([a, b]\) with continuous \( n \)th derivative \( f^{(n)}(x) \). Assume that \( y_j = f(x_j), j = 1, 2, \ldots, n \), and let the polynomial \( p(x) \) of degree at most \( n - 1 \) satisfy the interpolation conditions (1). Then the difference \( f(x) - p(x) \) can be expressed as

\[
 f(x) - p(x) = \prod_{j=1}^{n} (x - x_j) \frac{f^{(n)}(\xi)}{n!}, \quad a \leq x \leq b, \tag{11}
\]

where \( \xi \) is a function of the nodes \( x_1, x_2, \ldots, x_n \) and \( x \). The exact value of \( \xi \) is difficult to pin down, however, it is known that \( \xi \) is in the interval \([a, b]\) when \( x_1, x_2, \ldots, x_n \) and \( x \) are there. One can derive the expression (11) by using a variant of the mean-value theorem from Calculus.

We will not prove the error-formula (11) in this course. Instead, we will use the formula to learn about some properties of the polynomial interpolation problem. Usually, the \( n \)th derivative of \( f \) is not available and only the product over the nodes \( x_j \) can be studied. It is remarkable how much useful information can be gained by investigating this product! First we note that the interpolation error \( \max_{a \leq x \leq b} |f(x) - p(x)| \) is likely to be larger when the interval \([a, b]\) is long than when it is short. We can see this by doubling the size of the interval, i.e., we multiply \( a, b, x \) and the \( x_j \) by \( 2 \). Then the product in the right-hand side of (11) is replaced by

\[
 \prod_{j=1}^{n} (2x - 2x_j) = 2^n \prod_{j=1}^{n} (x - x_j),
\]

which shows that the interpolation error might be multiplied by \( 2^n \) when doubling the size of the interval. Actual computations show that, indeed, the error typically increases with the length of the interval when other relevant quantities remain unchanged.

The error-formula (11) also raises the question how the nodes \( x_j \) should be distributed in the interval \([a, b]\) in order to give a small error \( \max_{a \leq x \leq b} |f(x) - p(x)| \). Since we do not know the derivative \( f^{(n)}(\xi) \), we cannot choose the nodes \( x_j \) to minimize the right-hand side of (11). Instead, we may want to choose the nodes to minimize the magnitude of the product in the right-hand side.

This leads us to the minimization problem

\[
 \min_{a \leq x_1 < x_2 < \ldots < x_n \leq b} \max_{a \leq x \leq b} \prod_{j=1}^{n} |x - x_j|. \tag{12}
\]
The numerical solution of this minimization problem would be quite complicated, because the minimum depends in a nonlinear way on the nodes. Fortunately, this complicated problem has a simple solution! Let for the moment $a = -1$ and $b = 1$. Then the solution is given by

$$x_j = \cos\left(\frac{2j - 1}{2n} \pi\right), \quad j = 1, 2, \ldots, n. \quad (13)$$

These points are the projection of $n$ equidistant points on the upper half of the unit circle onto the $x$-axis; see Figure 1. The $x_j$ are known as Chebyshev points.

![Figure 1: Upper half of the unit circle with 8 equidistant points marked in red, and their projection onto the $x$-axis marked in magenta. The latter are, from left to right, the points $x_1, x_2, \ldots, x_8$ defined by (13) for $n = 8$.](image)

For intervals with endpoints $a < b$, the solution of (12) is given by

$$x_j = \frac{1}{2} (b + a) + \frac{1}{2} (b - a) \cos\left(\frac{2j - 1}{2n} \pi\right), \quad j = 1, 2, \ldots, n. \quad (14)$$

Example 11.5

We would like to approximate the Range function $f(x) = 1/(1 + 25x^2)$ by a polynomial in the interval $[-1, 1]$ and seek to determine a suitable polynomial of degree at most $n$ by interpolating $f$ at the $n + 1$ equidistant nodes $x_j = -1 + j/n$, $j = 0, 1, \ldots, n$. Note that $f$ can be differentiated arbitrarily many times on the interval $[-1, 1]$. Therefore, the error formula (11) holds. However, the right-hand side of (11) cannot be evaluated, because we do not know $\xi$. 

7
Figure 2: Example 11.5: The function \( f \) (in magenta) and interpolation polynomials \( p \) (in black) of different degrees. The equidistant interpolation points are marked by red circles. The maximal degrees of the interpolation polynomials are 5 for graph (a), 10 for (b), 15 for (c), and 20 for (d). Note that the scaling of the horizontal axes differ for the graphs. The graphs show that the approximation error \( \max_{-1 \leq x \leq 1} |f(x) - p(x)| \) increases with the number of interpolation points.

Figure 2 shows a few computed examples. The function \( f \) is plotted in magenta, the interpolation polynomials \( p \) in black, and the interpolation points \((x_j, f(x_j))\) are marked by red circles. Figure 2(a) shows \( f \) and \( p \) when the latter is determined by interpolation at 6 equidistant nodes; it is of degree at most 5. The approximation error is

\[
\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 4.3 \cdot 10^{-1}.
\]

In order to reduce this error, we determine the polynomial \( p \) of degree at most 10 that interpolates \( f \) at 11 equidistant nodes. This polynomial and the interpolation points are shown in
Figure 2(b). The polynomial can be seen to oscillate more at the endpoints of the interval than the polynomial of Figure 2(a). We obtain the approximation error

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 1.9.$$ 

Thus, the error increased by using more interpolation points! Figure 2(c) displays the polynomial of degree at most 15 that interpolates $f$ at 16 equidistant nodes; the approximation error is

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 2.1.$$ 

Finally, Figure 2(c) shows the polynomial of degree at most 20 that interpolates $f$ at 21 equidistant nodes; the approximation error now is

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 6.0 \cdot 10^1.$$ 

We conclude that polynomial interpolation at equidistant nodes does not always give useful polynomial approximants, even when the function has arbitrarily many continuous derivatives. □

**Example 11.6**

This example differs from Example 11.5 in that the interpolation points are Chebyshev points (13). Figure 3(a) shows the function $f(x) = 1/(1 + 25x^2)$ (in magenta) and the polynomial $p$ of degree at most 5 that interpolates $f$ at the nodes (13) with $n = 6$. The approximation error is

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 5.6 \cdot 10^{-1}.$$ 

This error is larger than for the polynomial determined by interpolation at 6 equidistant nodes; cf. Example 11.5. Figure 3(b) shows the polynomial $p$ of degree at most 10 that interpolates $f$ at 11 Chebyshev points. The approximation error is

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 1.1 \cdot 10^{-1}.$$ 

The polynomials of degrees at most 15 and 20 that interpolate at 16 and 21 Chebyshev points are displayed in Figures 3(c) and 3(d), respectively. The corresponding approximation errors are $8.9 \cdot 10^{-2}$ (16 nodes) and $1.5 \cdot 10^{-2}$ (21 nodes). Thus, the approximation error decreases when we increase the number of Chebyshev points. □

**Example 11.7**

Polynomial approximation by interpolation at Chebyshev points is attractive if the function to be approximated can be evaluated at these points. However, there are many situations when the
function is known at equidistant points. In this situation, we may consider reducing the degree of the approximating polynomial by fitting the polynomial in the least-squares sense. A polynomial of low degree is likely to oscillate less than a polynomial of high degree. Therefore, a least-squares polynomial of low degree might furnish a better approximation of the function than an interpolating polynomial of high degree.

Given $n$ distinct real nodes $x_1 < x_2 < \ldots < x_n$, we may determine a polynomial approximant

$$p(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_{m+1} x^m,$$
We may solve this problem by evaluating the QR factorization of the Vandermonde matrix. This approach has previously been discussed in Example 6.4 of Lecture 6 and Exercise 7.13 of Lecture 7.

Figure 4 shows polynomial approximants of the Runge function obtained with this approach. Graph (a) displays the Runge function (in magenta) and the polynomial approximant $p$ (in black) of degree at most $m = 9$. It was computed using $n = 21$ equidistant nodes marked with red circle. Note that $p$ does not interpolate $f$ at these nodes. The approximation error

$$\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 1.5 \cdot 10^{-1}.$$  

This error is smaller than for any of the interpolating polynomials of Figure 2.

Graph (b) of Figure 4 shows the corresponding polynomial approximant $p$ of degree at most $m = 20$ obtained by a least-squares fit of $f$ at 100 equidistant nodes. The approximation error now
is somewhat smaller:
\[
\max_{-1 \leq x \leq 1} |f(x) - p(x)| = 1.4 \cdot 10^{-1}.
\]

One can show that if the degree \(m\) of the least-squares polynomial \(p\) is chosen about \(2\sqrt{n}\), where \(n\) is the number of equidistant nodes, then the approximation error converges to zero as \(n\) is increased. However, as this example illustrates, the rate of convergence is not very fast. □

Despite the difficulties to determine an accurate polynomial approximant of the Runge function on \([-1, 1]\) by interpolation at equidistant nodes, for some functions, including \(e^x\), \(\sin(x)\), and \(\cos(x)\), the interpolation polynomials converge to the function on \([-1, 1]\) for all distributions of distinct interpolation points on the interval \([-1, 1]\) as the number of points is increased. For definiteness, consider \(f(x) = e^x\) and let \(x_1 < x_2 < \ldots < x_n\) be arbitrary distinct points in \([-1, 1]\). We obtain from (11) that
\[
\max_{-1 \leq x \leq 1} |f(x) - p(x)| = \max_{-1 \leq x \leq 1} \left| \prod_{j=1}^{n} (x - x_j) \right| \frac{|f^{(n)}(\xi)|}{n!} \\
\leq \max_{-1 \leq x \leq 1} \left| \prod_{j=1}^{n} |x - x_j| \right| \max_{-1 \leq x \leq 1} \frac{|f^{(n)}(\xi)|}{n!}. \tag{16}
\]

Since \(x, x_j \in [-1, 1]\), we have \(|x - x_j| \leq 2\). Therefore,
\[
\max_{-1 \leq x \leq 1} \left| \prod_{j=1}^{n} |x - x_j| \right| \leq 2^n.
\]

Moreover, \(f^{(n)}(\xi) = e^\xi\) with \(-1 \leq \xi \leq 1\). Therefore, \(|f^{(n)}(\xi)| \leq e\). Substituting these bounds into (16) shows that
\[
\max_{-1 \leq x \leq 1} |f(x) - p(x)| = \max_{-1 \leq x \leq 1} \left| \prod_{j=1}^{n} (x - x_j) \right| \frac{|f^{(n)}(\xi)|}{n!} \leq e \frac{2^n}{n!}.
\]

The right-hand side converges to zero as \(n\) increases, because \(n!\) grows much faster with \(n\) than \(2^n\). This shows that we can determine an arbitrarily accurate polynomial approximation of \(e^x\) on the interval \([-1, 1]\) by polynomial interpolation at arbitrary distinct nodes in the interval.

The fact that we obtain convergence for arbitrary distinct interpolation points, does not imply that all distributions of interpolation points the same approximation error. This will be explored further in Exercise 11.9.

The presence of a high-order derivative in the error-formula (11) indicates that interpolation polynomials are likely to give small approximation errors when the function has many continuous derivatives that are not very large in magnitude. Conversely, equation (11) suggests that interpolating a function with few or no continuous derivatives in \([a, b]\) by a polynomial of small to moderate
degree might not yield an accurate approximation of \( f(x) \) on \([a, b]\). This, indeed, often is the case. We therefore in the next section discuss an extension of polynomial interpolation which typically gives more accurate approximations than standard polynomial interpolation when the function to be approximated is not smooth.

**Exercise 11.1**

Solve the interpolation problem of Example 11.1.

**Exercise 11.2**

Write a MATLAB/Octave function for evaluating the polynomial in nested form (4). The input are the coefficients \( a_j \) and \( x \); the output is the value \( p(x) \).

**Exercise 11.3**

Let \( V_n \) be an \( n \times n \) Vandermonde matrix determined by \( n \) equidistant nodes in the interval \([-1, 1]\). How quickly does the condition number of \( V_n \) grow with \( n \)? Linearly, quadratically, cubically, \ldots, exponentially? Use the MATLAB/Octave functions `vander` and `cond`. Determine the growth experimentally. Describe how you designed the experiments. Show your MATLAB/Octave codes and relevant input and output.

**Exercise 11.4**

Show that the Lagrange polynomials (5) satisfy

\[
\sum_{j=1}^{n} \ell_j(x) = 1
\]

for all \( x \). Hint: Which function does the sum interpolate? Use the unicity of the interpolation polynomial.

**Exercise 11.5**

Write a MATLAB/Octave function for computing the weights of the barycentric representation (10) of the interpolation polynomial, using the definition (8). The code should avoid overflow and underflow.

**Exercise 11.6**

Given the weights (8), write a MATLAB/Octave function for evaluating the polynomial (10) at a point \( x \).
Exercise 11.7

(Bonus exercise.) Assume that the weights (8) are available for the barycentric representation of the interpolation polynomial (10) for the interpolation problem (1). Let another data point \( \{x_{n+1}, y_{n+1}\} \) be available. Write a MATLAB/Octave function for computing the barycentric weights for the interpolation problem (1) with \( n \) replaced by \( n + 1 \). The computations can be carried out in only \( O(n) \) arithmetic floating point operations. □

Exercise 11.8

The \( \Gamma \)-function is defined by

\[
\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt.
\]

Direct evaluation of the integral yields \( \Gamma(1) = 1 \) and integration by parts shows that \( \Gamma(x + 1) = x\Gamma(x) \). In particular, for integer-values \( n > 1 \), we obtain that

\[
\Gamma(n + 1) = n\Gamma(n)
\]

and therefore \( \Gamma(n + 1) = n(n - 1)(n - 2) \cdots 1 \). We would like to determine an estimate of \( \Gamma(4.5) \) by using the tabulated values of Table 1.

(a) Determine an approximation of the value of \( \Gamma(4.5) \) by interpolation at 3 and 5 nodes. Which 3 nodes should be used? Determine the actual value of \( \Gamma(4.5) \). Are the computed approximations close? Which one is more accurate?

(b) Investigate the following approach. Instead of interpolating \( \Gamma(x) \), interpolate \( \ln(\Gamma(x)) \) by polynomials at 3 and 5 nodes. Evaluate the computed polynomial at 4.5 and exponentiate.

How do the computed approximations in (a) and (b) compare? Explain! □

Exercise 11.9

(a) Interpolate the function \( f(x) = e^x \) at 20 equidistant nodes in \([-1, 1]\). This gives an interpolation polynomial \( p \) of degree at most 19. Measure the approximation error \( f(x) - p(x) \) by measuring the
difference at 1000 equidistant nodes \( t_j \) in \([-1, 1]\). We refer to the quantity

\[
\max_{t_j, j=1,2,...,1000} |f(t_j) - p(t_j)|
\]
as the error. Compute this error. Hint: Equidistant nodes can be generated with the MATLAB function \texttt{linspace}. The interpolation polynomial can be computed with the MATLAB function \texttt{polyfit}. This polynomial can be evaluated with the MATLAB function \texttt{polyval}.

(b) Repeat the computations in (a) using 20 Chebyshev points (13) as interpolation points. How do the errors compare for equidistant and Chebyshev points? Plot the error.

(c) The MATLAB function \texttt{polyfit} gives a warning message when the polynomial is of degree larger than about 20. This depends on that the monomial polynomial basis is used. Repeat the computations for problem (b) using the barycentric representation of the interpolation polynomial. Use the MATLAB functions for Exercises 11.5 and 11.6. Determine the approximation error for the interpolation polynomial of degree 40. □

**Exercise 11.10**

Compute an approximation of the integral

\[
\int_0^1 \sqrt{x} \exp(x^2) dx
\]

by first interpolating the integrand by a polynomial of degree at most 3 and then integrating the polynomial. Which representation of the polynomial is most convenient to use? Specify which interpolation points you use. □

**Exercise 11.11**

The function \( f(t) \) gives the position of a ball at time \( t \). Table 2 displays a few values of \( f \) and \( t \). Interpolate \( f \) by a quadratic polynomial and estimate the velocity and acceleration of the ball at time \( t = 1 \). □

<table>
<thead>
<tr>
<th>( t )</th>
<th>( f(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: \( t \) and \( f(t) \).
11.3 Interpolation by piecewise polynomials

In the above section, we sought to determine one polynomial that approximates a function on a specified interval. This works well if either one of the following conditions hold:

- The polynomial required to achieve desired accuracy is of fairly low degree.
- The function has several continuous derivatives and interpolation can be carried out at the Chebyshev points (13) or (14).

A quite natural and different approach to approximate a function on an interval is to first split the interval into subintervals and then approximate the function by a polynomial of fairly low degree on each subinterval. The present section discusses this approach.

Example 11.8

We would like to approximate a function on the interval \([-1, 1]\). Let the function values \(y_j = f(x_j)\) be available, where \(x_1 = -1, x_2 = 0, x_3 = 1\), and \(y_1 = y_3 = 0, y_2 = 1\). It is easy to approximate \(f(x)\) by a linear function on each subinterval \([x_1, x_2]\) and \([x_2, x_3]\). We obtain, using the Lagrange form (7),

\[
p(x) = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} = x + 1, \quad -1 \leq x \leq 0,
\]

\[
p(x) = y_2 \frac{x - x_3}{x_2 - x_3} + y_3 \frac{x - x_2}{x_3 - x_2} = 1 - x, \quad 0 \leq x \leq 1.
\]

The MATLAB command `plot([-1,0,1],[-1,0,0])` gives the continuous graph of Figure 5. This is a piecewise linear approximation of the unknown function \(f(x)\). If \(f(x)\) indeed is a piecewise linear function with a kink at \(x = 0\), then the computed approximation is appropriate. On the other hand, if \(f(x)\) displays the trajectory of a baseball, then the smoother function \(p(x) = 1 - x^2\), which is depicted by the dashed curve, may be a more suitable approximation of \(f(x)\), since baseball trajectories do not exhibit kinks - even if some players occasionally may wish they do.

Piecewise linear functions give better approximations of a smooth function if more interpolation points \(\{x_j, y_j\}\) are used. We can increase the accuracy of the piecewise linear approximant by reducing the lengths of the subintervals and thereby increasing the number of subintervals.

We conclude that piecewise linear approximations of functions are easy to compute. However, piecewise linear approximants display kinks. Therefore, many subintervals may be required to determine a piecewise linear approximant of high accuracy. □

There are several ways to modify piecewise linear functions to give them a more pleasing look. Here we will discuss how to use derivative information to obtain smoother approximants. A different approach, which uses Bézier curves, is described in Lecture 12.

We consider the task of approximating a function on the interval \([a, b]\) and first assume that not only the function values \(y_j = f(x_j)\), but also the derivative values \(y'_j = f'(x_j)\), are available at the
nodes $a = x_1 < x_2 < \ldots < x_n = b$. We can then on each subinterval, say $[x_j, x_{j+1}]$, approximate $f(x)$ by a polynomial that interpolates both $f(x)$ and $f'(x)$ at the endpoints of the subinterval. Thus, we would like to determine a polynomial $p_j(x)$, such that

$$p_j(x_j) = y_j, \quad p_j(x_{j+1}) = y_{j+1}, \quad p'_j(x_j) = y'_j, \quad p'_j(x_{j+1}) = y'_{j+1}. \quad (17)$$

These are 4 conditions, and we seek to determine a polynomial of degree 3,

$$p_j(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3, \quad (18)$$

that satisfies these conditions. Our reason for choosing a polynomial of degree 3 is that it has 4 coefficients, one for each condition. Substituting the polynomial (18) into the conditions (17) gives the linear system of equations,

$$\begin{bmatrix} 1 & x_j & x_j^2 & x_j^3 \\ 1 & x_{j+1} & x_{j+1}^2 & x_{j+1}^3 \\ 0 & 1 & 2x_j & 3x_j^2 \\ 0 & 1 & 2x_{j+1} & 3x_{j+1}^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} y_j \\ y_{j+1} \\ y'_j \\ y'_{j+1} \end{bmatrix}. \quad (19)$$

The last two rows impose interpolation of the derivative values. The matrix can be shown to be nonsingular when $x_j \neq x_{j+1}$. Matrices of the form (19) are referred to as confluent Vandermonde matrices.

The polynomials $p_1(x), p_2(x), \ldots, p_{n-1}(x)$ provide a piecewise cubic polynomial approximation of $f(x)$ on the whole interval $[a, b]$. They can be computed independently and yield an approximant.
that is continuous and has a continuous derivative on \([a, b]\). The latter can be established as
as follows: The polynomial \(p_j\) is defined and differentiable on the subinterval \([x_j, x_{j+1}]\) for \(j = 1, 2, \ldots, n-1\). What remains to be shown is that our approximant is continuous and has a continuous derivative at the interpolation points. This follows from the interpolation conditions (17). We have

\[
\lim_{x \to x_{j+1}} p_j(x) = p_j(x_{j+1}) = y_{j+1}, \quad \lim_{x \to x_{j+1}} p_{j+1}(x) = p_{j+1}(x_{j+1}) = y_{j+1}
\]

and

\[
\lim_{x \to x_{j+1}} p_j'(x) = p_j'(x_{j+1}) = y_{j+1}', \quad \lim_{x \to x_{j+1}} p_{j+1}'(x) = p_{j+1}'(x_{j+1}) = y_{j+1}'.
\]

The existence of the limits follows from the continuity of each polynomial and its derivative on the subinterval where it is defined. Thus,

\[
p_j(x_{j+1}) = p_{j+1}(x_{j+1}), \quad p_j'(x_{j+1}) = p_{j+1}'(x_{j+1}).
\]

This shows the continuity of our piecewise cubic polynomial approximant and its derivative at \(x_{j+1}\).

A popular approach to determine a smooth piecewise polynomial approximant is to use splines. Again consider the problems of approximation a function \(f\) on the interval \([a, b]\) and assume that function values \(y_j = f(x_j)\) are known at the nodes \(a = x_1 < x_2 < \ldots < x_{n-1} < x_n = b\), but that no derivative information is available. We then impose the conditions \(p_1(x_1) = y_1\) and

\[
p_j(x_j) = y_j, \quad p_j(x_{j+1}) = y_{j+1}, \quad p_j'(x_j) = p_j'(x_{j+1}), \quad p_j''(x_j) = p_j''(x_{j+1}),
\]

for \(j = 2, 3, \ldots, n-1\). Thus, at the subinterval boundaries at \(x_2, x_3, \ldots, x_{n-1}\), we require the piecewise cubic polynomial to have continuous first and second derivatives. However, these derivatives are not required to take on prescribed values. This kind of piecewise cubic polynomials are known as (cubic) splines. They are popular design tools in industry. Their determination requires the solution of a linear system of equations, which is somewhat complicated to derive. We will therefore omit its derivation. Extra conditions at the endpoints \(a\) and \(b\) have to be imposed in order to make the resulting linear system of equations uniquely solvable. For instance, we may require that the second derivative of the spline vanishes at \(a\) and \(b\). These are referred to as “natural” boundary conditions. Alternatively, if the derivative of the function to be approximated is known at \(a\) and \(b\), then we may require that the derivative of the spline takes on these values. If the function \(f\) is periodic with period \(b - a\), then it can be attractive to require the first and second derivative also be periodic.
**Exercise 11.12**

Consider the function in Example 11.8. Assume that we also know the derivative values \( y'_1 = 2 \), \( y'_2 = 0 \), and \( y'_3 = -2 \). Determine a piecewise polynomial approximation on \([-1, 1]\) by using the interpolation conditions (17). Plot the resulting function. □

**Exercise 11.13**

Assume the derivative values in the above exercise are not available. How can one determine estimates of these values? Use these estimates in the interpolation conditions (17) and compute a piecewise cubic approximation. How does it compare with the one from Exercise 11.12 and with the piecewise linear approximation of Example 11.8. Plot the computed approximant. □

**Exercise 11.14**

Compute a spline approximant of the function of Example 11.8, e.g., by using the function `spline` in MATLAB or Octave. Plot the computed spline. □

**Exercise 11.15**

Compute a spline approximant of the function of Example 11.5 using 20 equidistant nodes e.g., by using the function `spline` in MATLAB or Octave. Plot the computed spline. Compute the approximation error similarly as in Example 11.5. □