7 Polynomial and piecewise polynomial interpolation

Let \( f \) be a function, which is only known at the 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 \( f(x) \) for other values of \( x \).

Example 7.1: Assume that we need to evaluate \( \cos(\pi/6) \), but the trigonometric function-key on your calculator is broken and we do not have access to a computer. We do remember 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 7.2: 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 7.3: 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.

7.1 Polynomial interpolation

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

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

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

Example 7.4: Let \( n = 1 \). Then the interpolation polynomial reduces to the constant \( y_1 \). When \( n = 2 \), the interpolating 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 7.1 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 \( p(\pi/6) \).

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

\[
p(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}.
\]
This is a polynomial of degree at most \( n - 1 \). We would like to determine the coefficients \( a_j \), which multiply powers of \( x \), so that the 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 in matrix form as follows,

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

The above matrix is known as a Vandermonde matrix. It is nonsingular when the nodes \( x_j \) are distinct. For instance, when \( n = 2 \), we have

\[
\det \begin{bmatrix}
1 & x_1 \\
1 & x_2
\end{bmatrix} = x_2 - x_1 \neq 0.
\]

The nonsingularity of the Vandermonde matrix guarantees the existence of a unique interpolation polynomial.

The representation of a polynomial \( p(x) \) in terms of the powers of \( x \), like in (2), 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

\[
p(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 = a_1 + (a_2 + (a_3 + a_4 x) x) x.
\]

The right-hand side is easy to evaluate; only \( O(n) \) arithmetic floating point operations are required, see Exercise 7.2.

However, Vandermonde matrices generally are severely ill-conditioned. This is illustrated in Exercise 7.3. 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 basis for polynomials of degree \( n - 1 \) is given by

\[
\ell_k(x) = \prod_{\substack{j=1 \\text{to} \ n \\text{\&} \ j \neq k}} \frac{x - x_j}{x_k - x_j}, \quad k = 1, 2, \ldots, n.
\]

The \( \ell_k(x) \) are known as Lagrange polynomials. They are of degree \( n - 1 \). It is easy to verify that the Lagrange polynomials satisfy

\[
\ell_k(x_j) = \begin{cases} 
1, & k = j, \\
0, & k \neq j.
\end{cases}
\]

This property makes it possibly to determine the interpolation polynomial in terms solving a linear system of equations. It follows from (5) that the interpolation polynomial is given by

\[
p(x) = \sum_{k=1}^{n} y_k \ell_k(x).
\]
We refer to this expression as the interpolation polynomial in *Lagrange form*.

The only drawback of the representation (6) of the interpolation polynomial is that it is somewhat cumbersome to evaluate; straightforward evaluation of each Lagrange polynomial $\ell_k(x)$ requires $O(n)$ arithmetic floating point operations, which suggests that the evaluation of the sum (6) requires $O(n^2)$ arithmetic floating point operations. The latter operation 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*

$$w_k = \frac{1}{\prod_{j=1 \atop j \neq k}^{n} (x_k - x_j)}.$$  \hspace{1cm} (7)

Then the Lagrange polynomials can be written as

$$\ell_k(x) = \ell(x) \frac{w_k}{x - x_k}, \hspace{0.5cm} k = 1, 2, \ldots, n.$$  

We assume for simplicity that $x \neq x_k$. All terms in the sum (6) contain the factor $\ell(x)$, which is independent of $k$. We therefore can move this factor outside the sum, and obtain

$$p(x) = \ell(x) \sum_{k=1}^{n} y_k \frac{w_k}{x - x_k}.$$  \hspace{1cm} (8)

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 $k$. The expression (8) simplifies to

$$1 = \ell(x) \sum_{k=1}^{n} \frac{w_k}{x - x_k},$$

which shows that

$$\ell(x) = \frac{1}{\sum_{k=1}^{n} \frac{w_k}{x - x_k}}.$$  

Finally, substituting the above expression into (8) yields

$$p(x) = \sum_{k=1}^{n} y_k \frac{w_k}{x - x_k}.$$  \hspace{1cm} (9)

This formula is known as the *barycentric representation* of the Lagrange interpolating polynomial, or simply as the interpolating polynomial in barycentric form. It requires that the weights be computed, e.g., by using
the definition (7). This requires $O(n^2)$ arithmetic floating point operations. Given the weights, $p(x)$ can be determined for each value of $x$ in only $O(n)$ arithmetic floating point operations; see Exercise 7.4 for further details.

The representation (9) can be shown to be quite insensitive to round-off errors and therefore also 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 (9) allows the weights to be multiplied by an arbitrary nonzero constant.

### 7.2 The approximation error

Let the nodes $x_j$ be distinct and live in the real interval $[a, b]$, and let $f(x)$ be a function, which is $n$ times differentiable in $[a, b]$. Let $f^{(n)}(x)$ denote the $n$th derivative. Assume that $y_j = f(x_j), j = 1, 2, \ldots, n$. Then the difference $f(x) - p(x)$ can be expressed as

$$f(x) - p(x) = \prod_{j=1}^{n} (x - x_j) \frac{f^{(n)}(\xi)}{n!},$$

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 $x$ and $x_1, x_2, \ldots, x_n$ are there. One can derive the expression (10) by using a variant of the mean-value theorem from Calculus.

We will not prove the error-formula (10) 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 easily. 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 (10) 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 (10) also raises the question how the nodes $x_j$ should be distributed in the interval $[a, b]$ in order to give a small error $\max_{a \leq x \leq b} |f(x) - p(x)|$. For instance, we may want to choose nodes $x_j$ that solve the minimization problem

$$\min_{x_j} \max_{a \leq x \leq b} \prod_{j=1}^{n} |x - x_j|.$$  \hspace{1cm} (11)

This complicated problem turns out to have a simple solution! Let for the moment $a = -1$ and $b = 1$. Then the solution is given by

$$x_j = \cos \left( \frac{2j - 1}{2n} \pi \right), \quad j = 1, 2, \ldots, n.$$  \hspace{1cm} (12)

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.
For intervals with endpoints $a < b$, the solution of (11) is given by

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

Finally, the presence of a high-order derivative in the error-formula (10) suggests that interpolation polynomials are likely to give small approximation errors when the function has many continuous derivatives that are not very large in magnitude. In the next subsection, we will discuss a modifications of polynomial interpolation, that can work better when the function to be approximated does not have many (or any) continuous derivatives.

**Exercises**

**Exercise 7.1**: Solve the interpolation problem of Example 7.1.

**Exercise 7.2**: Write a MATLAB or Octave function for evaluating the polynomial (2) in nested form (4). The input are the coefficients $a_j$ and $x$; the output is the value $p(x)$. 

**Exercise 7.3**: 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, …, exponentially? Use the MATLAB or Octave functions `vander` and `cond`. Determine the growth experimentally. Describe how you designed the experiments. Show your MATLAB or Octave codes and relevant input and output.

**Exercise 7.4**: Write a MATLAB or Octave function for computing the weights of the barycentric representation (9) of the interpolation polynomial, using the definition (7). The code should avoid overflow and underflow.
Exercise 7.5: (Bonus exercise.) Assume that the weights (7) are available for the barycentric representation of the interpolation polynomial (9) for the interpolation problem (1). Let another data point \( \{x_{n+1}, y_{n+1}\} \) be available. Write a MATLAB or 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 \( \mathcal{O}(n) \) arithmetic floating point operations.  

Exercise 7.6: The pressure in a fully automatic espresso machine varies with size of the coffee grinds. The pressure is measured in bar and the grind size in units, where 1 signifies a coarse grind and 5 a fine one. The relation is tabulated in Table 1.

(a) Determine by linear interpolation, which pressure grind size 2.1 gives rise to. Which data should be used?

(b) Determine by quadratic interpolation, which pressure grind size 2.1 gives rise to. Which data should be used? Are the results for linear and quadratic interpolation close?  

Exercise 7.7: 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 2.

(a) Determine the actual value of \( \Gamma(4.5) \) by interpolation in 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) Also, 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.

---

### Table 1: Grind size (left-hand side column) versus pressure (right-hand side column).

<table>
<thead>
<tr>
<th>Grind Size (unit)</th>
<th>Pressure (bar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.0</td>
</tr>
<tr>
<td>2</td>
<td>12.5</td>
</tr>
<tr>
<td>3</td>
<td>14.5</td>
</tr>
<tr>
<td>4</td>
<td>16.0</td>
</tr>
<tr>
<td>5</td>
<td>18.0</td>
</tr>
</tbody>
</table>

### Table 2: \( n \) and \( \Gamma(n) \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \Gamma(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>120</td>
</tr>
</tbody>
</table>
How do the computed approximations in (a) and (b) compare? Explain! □

Exercise 7.8: (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 measure the difference at 500 equidistant nodes \( t_j \) in \([-1, 1]\). We refer to the quantity

\[
\max_{t_j, j=1,2,\ldots,500} |f(t_j) - p(t_j)|
\]

as the error. Compute the error.

(b) Repeat the above computations with the function \( f(x) = e^x/(1 - 25x^2) \). Plot \( p(t_j) - f(t_j), j = 1, 2, \ldots, 500 \). Where in the interval \([-1, 1]\) is the error the largest?

(c) Repeat the computations in (a) using 20 Chebyshev points (12) as interpolation points. How do the errors compare for equidistant and Chebyshev points? Plot the error.

(d) Repeat the computations in (b) using 20 Chebyshev points (12) as interpolation points. How do the errors compare for equidistant and Chebyshev points? □

Exercise 7.9: Compute an approximation of the integral

\[
\int_{0}^{1} \sqrt{x} \exp(x^2) dx
\]

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

Exercise 7.10: The function \( f(t) \) gives the position of a ball at time \( t \). Table 3 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 \( t = 1 \).

### 7.3 Interpolation by piecewise polynomials

In the above subsection, 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 a few continuous derivatives and interpolation can be carried out at the Chebyshev points (12) or (13).

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.
Example 7.6: 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 (6),

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

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

The MATLAB command `plot([-1,0,1],[0,1,0])` gives the continuous graph of Figure 2. 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 approximation by reducing the lengths of the subintervals.

We conclude that piecewise linear approximations of functions are easy to compute. These approximations display kinks. Many interpolation points may be required to determine a piecewise linear approximant of desired accuracy. The benefits of piecewise linear approximation is the ease of the computations on each subinterval. □

![Figure 2: Example 7.6: Quadratic polynomial $p(x) = 1 - x^2$ (red dashed graph) and piecewise linear approximation (continuous blue graph).](image-url)

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 the next lecture.

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 $a \leq x_1 < x_2 < \ldots < x_n \leq b$. 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. Thus, we would like to determine a polynomial $p_j(x)$, such that

$$p_j(x_j) = y_j, \quad p_j(x_{j+1}) = y_{j+1}, \quad p'_j(x_j) = y'_j, \quad p'_j(x_{j+1}) = y'_{j+1}. \quad (14)$$

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

$$p_j(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3, \quad (15)$$

which 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 (15) into the conditions (14) gives the linear system of equations,

$$
\begin{bmatrix}
1 & x_j & x_j^2 & x_j^3 \\
1 & x_{j+1} & x_{j+1}^2 & x_{j+1}^3 \\
0 & 1 & 2x_j & 3x_j^2 \\
0 & 1 & 2x_{j+1} & 3x_{j+1}^2
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4
\end{bmatrix}
= 
\begin{bmatrix}
y_j \\
y_{j+1} \\
y'_j \\
y'_{j+1}
\end{bmatrix}.
\quad (16)
$$

The last 2 rows impose interpolation of the derivative values. The matrix can be shown to be nonsingular when $x_j \neq x_{j+1}$.

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 approximation with a continuous derivative on $[a, b]$. The latter can be seen as follows: The polynomial $p_j$ is defined and differentiable on the interval $[x_j, x_{j+1}]$ for $j = 1, 2, \ldots, n-1$. What remains to be established is that our approximant also has a continuous derivative at the interpolation points. This, however, follows from the interpolation conditions (14). We have that

$$\lim_{x \searrow x_{j+1}} p'_j(x) = p'_j(x_{j+1}) = y'_{j+1}, \quad \lim_{x \nearrow x_{j+1}} p'_j(x) = p'_j(x_{j+1}) = y'_{j+1}. \quad (16)$$

The existence of the limit follows from the continuity of each polynomial on the interval where it is defined, and the other equalities are the interpolation conditions. Thus, $p'_j(x_{j+1}) = p'_{j+1}(x_{j+1})$, which shows the continuity of the derivative at $x_{j+1}$ of our piecewise cubic polynomial approximant.

The use of piecewise cubic polynomials as described gives attractive approximations. However, the approach discussed requires derivative information be available. When no derivative information is explicitly known, modifications of the scheme outlined can be used. A simple modification is to use estimates the derivative-

Exercise 7.11: Consider the function in Example 7.6. 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 (14). Plot the resulting function. □
**Exercise 7.12:** 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 (14) and compute a piecewise cubic approximation. How does it compare with the one from Exercise 7.11 and with the piecewise linear approximation of Example 7.6.

**Exercise 7.13:** Compute a spline approximant, e.g., by using the function `spline` in MATLAB or Octave.