Chapter 10

Polynomial Interpolation

Polynomial interpolants are rarely the end product of a numerical process. Their importance is more as building blocks for other, more complex algorithms in differentiation, integration, solution of differential equations, approximation theory at large, and other areas. Hence, polynomial interpolation arises frequently; indeed, it is one of the most ubiquitous tasks, both within the design of numerical algorithms and in their analysis. Its importance and centrality help explain the considerable length of the present chapter.

Section 10.1 starts the chapter with a general description of approximation processes in one independent variable, arriving at polynomial interpolation as one such fundamental family of techniques. In Sections 10.2, 10.3, and 10.4 we shall see no less than three different forms (different bases) of interpolating polynomials. They are all of fundamental importance and are used extensively in the practical construction of numerical algorithms.

Estimates and bounds for the error in polynomial interpolation are derived in Section 10.5. If the choice of locations for the interpolation data is up to the user, then a special set of abscissae (nodes) called Chebyshev points is an advantageous choice, and this is discussed in Section 10.6. Finally, Section 10.7 considers the case where not only function values but also derivative values are available for interpolation.

10.1 General approximation and interpolation

Interpolation is a special case of approximation. In this section we consider different settings in which approximation problems arise, explain the need for finding approximating functions, describe a general form for interpolants and important special cases, and end up with polynomial interpolation.

Discrete and continuous approximation in one dimension

It is possible to distinguish between approximation techniques for two types of problems:

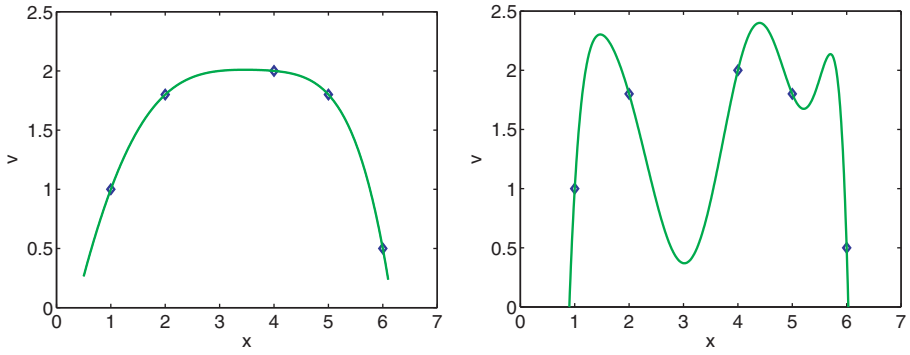
1. **Data fitting** (Discrete approximation problem):

   Given a set of data points $\{(x_i, y_i)\}_{i=0}^{n}$, find a *reasonable* function $v(x)$ that fits the data points.

   If the data are accurate it might make sense to require that $v(x)$ **interpolate** the data, i.e., that the curve pass through the data exactly, satisfying

   $$v(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

   See Figure 10.1.



(a) Reasonable. (b) Unreasonable.

Figure 10.1. *Different interpolating curves through the same set of points.*

2. **Approximating functions**:

   For a complicated function $f(x)$ (which may be given explicitly, or only implicitly), find a simpler function $v(x)$ that approximates $f(x)$.

   For instance, suppose we need to quickly find an approximate value for sin(1.2) (that's 1.2 in radians, not degrees) with only a primitive calculator at hand. From basic trigonometry we know the values of $\sin(x)$ for $x = 0, \pi/6, \pi/4, \pi/3$, and $\pi/2$: how can we use these to estimate sin(1.2)?

   For another instance, suppose we have a complex, expensive program that calculates the final point (say, the landing location) of the trajectory of a space shuttle for each given value of a certain control parameter. We perform this calculation, i.e., invoke the program, for several parameter values. But then we may want to use this computed information to have an idea of the resulting landing location for other values of the control parameter—without resorting to the full calculation for each parameter value.

   Interpolation techniques for such function approximation are *identical* to those of data fitting once we specify the data points $\{(x_i, y_i = f(x_i))\}_{i=0}^{n}$. The *difference* between function interpolation and data fitting interpolation is that in the former we

   - have some freedom to choose x_i cleverly, and
   - may be able to consider the *global* interpolation error.

The need for interpolation

Why do we want to find an approximating function $v(x)$ in general?

- For *prediction*: we can use $v(x)$ to find approximate values of the underlying function at locations x other than the data abscissae, $x_0, \ldots, x_n$.

  If x is inside the smallest interval containing all the data abscissae, then this is called *interpolation*; if x is outside that interval, then we have *extrapolation*. For instance, we may have data regarding the performance of some stock at each trading week's end during the past year. Interpolating for the value of the stock during other days of last year is a relatively safe undertaking. Extrapolating for the value of the stock sometime next year is much more risky (although potentially more interesting).

- For *manipulation*: an instance is finding approximations for derivatives and integrals of the underlying function.

The interpolating function must be not only easy to evaluate and manipulate, but also "reasonable"; i.e., it must resemble a curve that we would actually draw through the points—see Figure 10.1. Just exactly what qualifies as reasonable is context-dependent and hard to define in precise terms for general purposes.

Interpolants and their representation

We generally assume a *linear form* for all interpolating (or, more generally, approximating) functions $v(x)$.[38] Thus we write

$$v(x) = \sum_{j=0}^{n} c_j \phi_j(x) = c_0\phi_0(x) + \cdots + c_n\phi_n(x),$$

where $\{c_j\}_{j=0}^n$ are *unknown coefficients*, or **parameters** determined from the data, and $\{\phi_j(x)\}_{j=0}^n$ are predetermined **basis functions**. These basis functions are further assumed to be *linearly independent*, which means that if coefficients $\{c_j\}_{j=0}^n$ are found such that $v(x) = 0$ for all x, then all these coefficients themselves must vanish: $c_j = 0,\ j = 0, 1, \ldots, n$.

Notice our default assumption that the number of basis functions equals the number of data points $n+1$. If there are fewer basis functions than data, then we cannot hope for interpolation of all data values and we typically end up resorting to a least squares approach as in Chapter 6. Other techniques of approximating functions which do not fit the data exactly are considered in Chapters 12 and 13.

In general, the interpolation conditions can be written as $n+1$ linear relations for the $n+1$ unknown coefficients. The resulting linear system of equations is

$$\begin{pmatrix} \phi_0(x_0) & \phi_1(x_0) & \phi_2(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & \vdots & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_n(x_n) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{10.1}$$

We may not actually directly form and solve the system (10.1) in a given situation, but it is always there as a result of the assumption of linear representation of $v(x)$ in terms of the basis functions.

Here are some common examples of interpolants:

- In the present chapter we consider *polynomial interpolation*

$$v(x) = \sum_{j=0}^{n} c_j x^j = c_0 + c_1 x^1 + \cdots + c_n x^n.$$

This simplest and most familiar form of representing a polynomial implies the choice of a *monomial* basis

$$\phi_j(x) = x^j, \quad j = 0, 1, \ldots, n,$$

but we will see other choices as well.

[38] Note that the form of the interpolating function is linear in the sense that it is a linear combination of basis functions in some appropriate space, *not* that $v(x)$ itself is a linear function of x.

- In the next chapter we discuss *piecewise polynomial interpolation*, which is based on performing polynomial interpolation in "pieces," rather than on the entire given interval.
- *Trigonometric interpolation* is also extremely useful, especially in signal processing and for describing wave and other periodic phenomena. For instance, consider

$$\phi_j(x) = \cos(jx), \quad j = 0, 1, \ldots, n.$$

We elaborate on more general choices in this spirit in Chapter 13.

In general, it is important to distinguish two stages in the interpolation process:

1. **Constructing** the interpolant. These are operations that are independent of where we would then evaluate $v(x)$. An instance of this is determining the coefficients $c_0, c_1, \ldots, c_n$ for a given basis $\phi_0(x), \phi_1(x), \ldots, \phi_n(x)$.
2. **Evaluating** the interpolant at a given point x.

The interpolant construction is done once for a given set of data. After that, the evaluation may be applied many times.

Polynomial interpolation

Note: The rest of this chapter is devoted exclusively to polynomial interpolation.

The main reason polynomial approximation is desirable is its simplicity. Polynomials

- are easy to construct and evaluate (recall also the nested form from Example 1.4);
- are easy to sum and multiply (and the result is also a polynomial);
- are easy to differentiate and integrate (and the result is also a polynomial); and
- have widely varying characteristics despite their simplicity.

10.2 Monomial interpolation

Let us denote a polynomial interpolant of degree at most n by

$$p(x) = p_n(x) = \sum_{j=0}^{n} c_j x^j = c_0 + c_1 x + \cdots + c_n x^n.$$

For $n+1$ data points

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n),$$

we want to find $n+1$ coefficients[39] $c_0, c_1, \ldots, c_n$ such that

$$p(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

We will assume, until Section 10.7, that the abscissae of the data points are distinct, meaning

$$x_i \neq x_j \quad \text{whenever} \quad i \neq j.$$

[39] Remember that a polynomial of degree n has $n+1$, not n, coefficients.

Example 10.1. Let $n = 1$ and let our two data points be $(x_0, y_0) = (1, 1)$ and $(x_1, y_1) = (2, 3)$. We want to fit a polynomial of degree at most 1 of the form

$$p_1(x) = c_0 + c_1 x$$

through these two points.

The interpolating conditions are

$$p_1(x_0) = c_0 + 1c_1 = 1,$$
$$p_1(x_1) = c_0 + 2c_1 = 3.$$

These are two linear equations for the two unknowns c_0 and c_1, which can be solved using high school algebra techniques. We obtain $c_0 = -1$ and $c_1 = 2$, so

$$p_1(x) = 2x - 1.$$

This linear interpolant is depicted in Figure 10.2.

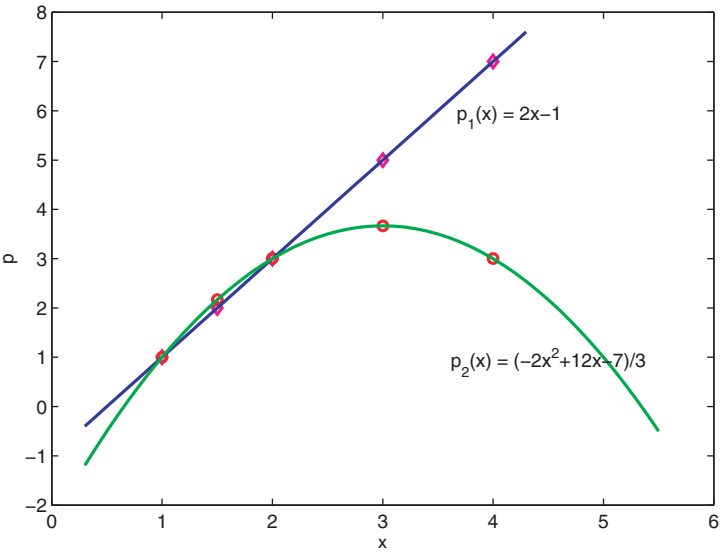


Figure 10.2. *Quadratic and linear polynomial interpolation.*

Next, let $n = 2$ and let our three data points be $(1, 1)$, $(2, 3)$, and $(4, 3)$. The first two are the same as above, but the third data pair specifies a significantly different value at $x = 4$ than what $p_1(x)$ predicts: while $p_1(4) = 7$, here we seek a polynomial whose value at $x = 4$ equals 3. Thus, we want to fit a polynomial of degree at most 2 of the form

$$p_2(x) = c_0 + c_1 x + c_2 x^2$$

through these three points. Note that the coefficients c_0 and c_1 in $p_2(x)$ are not expected to be the same as for $p_1(x)$, in general. The interpolating conditions are

$$p_2(x_0) = c_0 + 1c_1 + 1c_2 = 1,$$
$$p_2(x_1) = c_0 + 2c_1 + 4c_2 = 3,$$
$$p_2(x_2) = c_0 + 4c_1 + 16c_2 = 3.$$

This is a 3×3 linear system for the unknown coefficients c_j, which in matrix form reads

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}.$$

The MATLAB commands

```
A = [1 1 1; 1 2 4; 1 4 16];
y = [1; 3; 3];
c = A \ y
```

yield (of course, up to a rounding error)

$$c_0 = -\frac{7}{3}, \quad c_1 = 4, \quad c_2 = -\frac{2}{3}.$$

This completes the construction of the quadratic interpolant. The desired interpolating polynomial p_2 is

$$p_2(x) = (-2x^2 + 12x - 7)/3,$$

and this can be evaluated for any given value of x. For instance, at $x = 3$ we have

$$p_2(3) = \frac{11}{3},$$

which is quite lower than the value $p_1(3) = 5$. Observe also the different values of the two polynomials at $x = 1.5$ in Figure 10.2, as the pair of close but not coinciding magenta symbols illustrates. ∎

Unique interpolating polynomial

Generalizing the example above to the case of $n+1$ data points, the interpolation conditions lead to a system of $(n+1)$ linear equations in the $(n+1)$ unknowns $c_0, c_1, \ldots, c_n$ given by

$$\begin{pmatrix} 1 & {x_0}^1 & {x_0}^2 & \cdots & {x_0}^n \\ 1 & {x_1}^1 & {x_1}^2 & \cdots & {x_1}^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & {x_n}^1 & {x_n}^2 & \cdots & {x_n}^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The coefficient matrix X is called a *Vandermonde* matrix; in particular, it is known from an introductory linear algebra text that

$$\det(X) = \prod_{i=0}^{n-1} \left[\prod_{j=i+1}^{n} (x_j - x_i) \right],$$

i.e., this determinant is the product of all possible differences in the x_i. So, as long as the abscissae are distinct, $\det(X) \neq 0$ and hence X is nonsingular.[40] This argument provides a simple proof to

[40] It also verifies our assumption that the basis functions, in this case the monomials $\phi_j(x) = x^j$, $j = 0, 1, \ldots, n$, are linearly independent, because the matrix is nonsingular for an arbitrary choice of distinct points $x_0, x_1, \ldots, x_n$.

Product Notation.
Let $z_1, z_2, \ldots, z_n$ be n scalar values. We are used to the notation for their sum, given by

$$\Sigma_{i=1}^{n} z_i = z_1 + z_2 + \cdots + z_n.$$

Analogously, the notation for their product is

$$\Pi_{i=1}^{n} z_i = z_1 \cdot z_2 \cdots z_n.$$

Theorem: Polynomial Interpolant Unique Existence.
For any real data points $\{(x_i, y_i)\}_{i=0}^{n}$ with distinct abscissae x_i there exists a unique polynomial $p(x)$ of degree at most n which satisfies the interpolation conditions

$$p(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

the theorem given on this page that there exists a *unique interpolating polynomial* p. The uniqueness of polynomial interpolation is particularly important because of the different forms that this polynomial can take: the same result is obtained, no matter which method or basis is used to obtain the interpolating polynomial.

Later on, in Section 10.7, we will generalize this existence and uniqueness result for the case when data abscissae are not necessarily distinct.

Using the monomial basis for constructing interpolants

Our discussion so far not only shows uniqueness but also suggests a general way for obtaining the interpolating polynomial $p(x)$: form the Vandermonde matrix and solve a linear system of equations. The big advantage of this approach is its intuitive simplicity and straightforwardness. However, if we consider a general-purpose use of polynomial interpolation, then this approach has disadvantages:

1. The calculated coefficients c_j are not directly indicative of the interpolated function, and they may completely change if we wish to slightly modify the interpolation problem; more on this in Sections 10.3 and 10.4.

2. The Vandermonde matrix X is often ill-conditioned (see Section 5.8), so the coefficients thus determined are prone to inaccuracies.

3. This approach requires about $\frac{2}{3}n^3$ operations (flops) to carry out Gaussian elimination (see Section 5.1) for the construction stage; another method exists which requires only about n^2 operations. The evaluation stage, however, is as quick as can be; using the nested form, it requires about $2n$ flops per evaluation point.

There are situations in which the last two disadvantages are not important. First, the higher computational cost, if there is any, is important only when n is "large," not when it equals 2 or 3. Also, the ill-conditioning, i.e., the claim that the basis $\phi_j(x) = x^j$, $j = 0, 1, \ldots, n$, is "not good" in the sense that roundoff error gets unreasonably magnified, occurs mostly when the interval of interpolation is wide or n is not small. If all points of concern, including all data points x_i and evaluation points x, are in a small interval near, say, some point $\hat{x}$, then the basis

$$\{1, (x - \hat{x}), (x - \hat{x})^2, (x - \hat{x})^3\}$$

is perfectly reasonable for a cubic polynomial. More on this in Section 11.2. The polynomial

$$p(x) = c_0 + c_1(x - \hat{x}) + \cdots + c_n(x - \hat{x})^n$$

is, in fact, reminiscent of a *Taylor series* expansion (see page 5)

$$f(x) = f(\hat{x}) + f'(\hat{x})(x - \hat{x}) + \cdots + \frac{f^{(n)}(\hat{x})}{n!}(x - \hat{x})^n + R_n(x),$$

where the remainder term can be written as

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - \hat{x})^{n+1}$$

for some ξ between x and $\hat{x}$.

Throughout most of this chapter we consider values of n in the single decimal digit territory. Larger polynomial degrees appear only in Section 10.6, where the monomial basis is indeed inadequate.

There are several additional desirable features, often far more important than the efficiency considerations mentioned above, which the simple monomial basis $\{\phi_j(x) = x^j\}$ does not have. These are introduced in the next two sections, together with bases that do possess such features and which allow a more intuitive understanding in context of both the problems and their resolutions.

Specific exercises for this section: Exercises 1–3.

10.3 Lagrange interpolation

The coefficients c_j of the polynomials $p_1(x)$ and $p_2(x)$ in Example 10.1 do not relate directly to the given data values y_j. It would be nice if a polynomial basis is found such that $c_j = y_j$, giving

$$p(x) = p_n(x) = \sum_{j=0}^{n} y_j \phi_j(x).$$

Such a representation would be particularly easy to manipulate, for instance, when we seek formulas for differentiation or integration. This is shown in later chapters, particularly Chapters 14 through 16.

Such a polynomial basis is provided by the Lagrange interpolation process. For this we define the **Lagrange polynomials**, $L_j(x)$, which are polynomials of degree n that satisfy

$$L_j(x_i) = \begin{cases} 0, & i \neq j, \\ 1, & i = j. \end{cases}$$

Given data y_i at abscissae x_i as before, the unique polynomial interpolant of degree at most n can now be written as

$$p(x) = \sum_{j=0}^{n} y_j L_j(x).$$

Indeed, p is of degree at most n (being the linear combination of polynomials of degree n), and it satisfies the interpolation conditions because

$$p(x_i) = \sum_{j=0}^{n} y_j L_j(x_i) = 0 + \cdots + 0 + y_i L_i(x_i) + 0 + \cdots + 0 = y_i.$$

Example 10.2. Let us use the same three data pairs of Example 10.1, namely, $(1,1)$, $(2,3)$, and $(4,3)$, to demonstrate the construction of Lagrange polynomials. To make $L_0(x)$ vanish at $x=2$ and $x=4$ we write

$$L_0(x) = a(x-2)(x-4).$$

Requiring $L_0(1)=1$ then determines $a(1-2)(1-4)=1$, i.e., $a=\frac{1}{3}$, and thus

$$L_0(x) = \frac{1}{3}(x-2)(x-4).$$

Similarly we determine that

$$L_1(x) = -\frac{1}{2}(x-1)(x-4), \quad L_2(x) = \frac{1}{6}(x-1)(x-2).$$

These Lagrange polynomials are depicted in Figure 10.3. We thus obtain the interpolant

$$\begin{aligned} p_2(x) &= \frac{y_0}{3}(x-2)(x-4) - \frac{y_1}{2}(x-1)(x-4) + \frac{y_2}{6}(x-1)(x-2) \\ &= \frac{1}{3}(x-2)(x-4) - \frac{3}{2}(x-1)(x-4) + \frac{3}{6}(x-1)(x-2). \end{aligned}$$

Despite the different form, this is precisely the same quadratic interpolant as the one in Example 10.1, so in fact, $p_2(x) = (-2x^2+12x-7)/3$. It is also easy to verify that here, too, $p_2(3) = \frac{11}{3}$. All this should not come as a surprise—it's an illustration of the uniqueness of polynomial interpolation; see the theorem on page 301. ∎

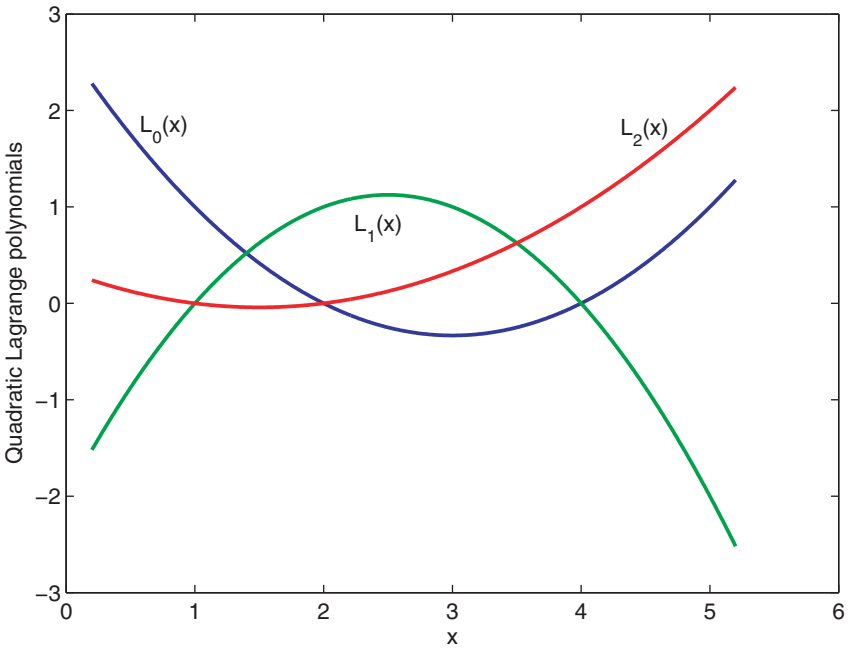


Figure 10.3. *The quadratic Lagrange polynomials $L_0(x)$, $L_1(x)$, and $L_2(x)$ based on points $x_0=1$, $x_1=2$, $x_2=4$, used in Example 10.2.*

Properties of Lagrange polynomials

What properties do Lagrange polynomials have? What do they look like?

In the general case where $n+1$ data abscissae x_i are specified, the Lagrange polynomials uniquely exist, because they are really nothing other than polynomial interpolants for special data.[41] This is also straightforward to verify directly, by explicitly specifying

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

Indeed, the polynomial of degree n, written in terms of its roots as

$$(x-x_0)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n),$$

clearly interpolates the 0-values at all data abscissae other than x_j, and dividing by its value at x_j normalizes the expression to yield $L_j(x_j) = 1$. Another picture of a Lagrange polynomial is provided in Figure 10.4.

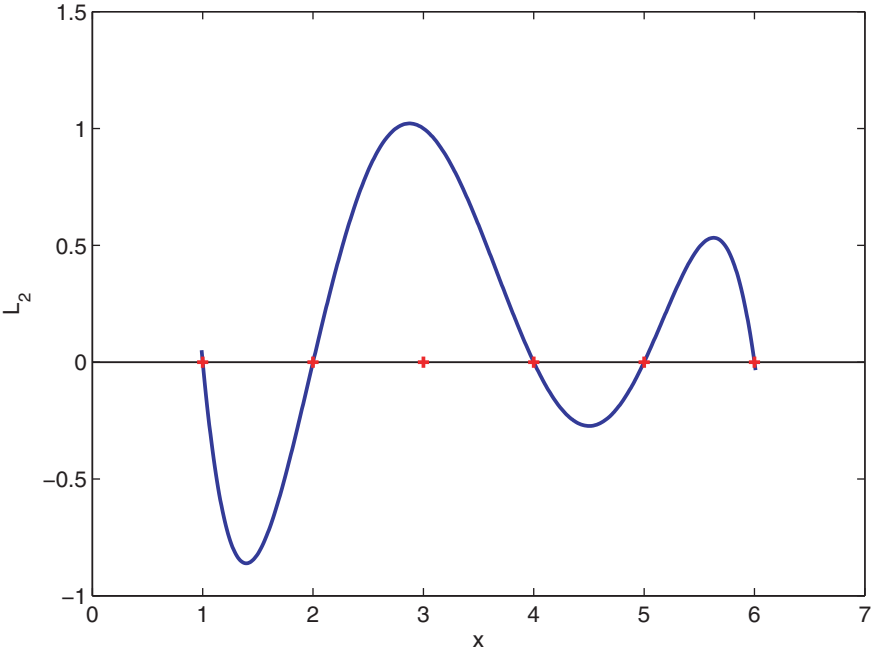


Figure 10.4. *The Lagrange polynomial $L_2(x)$ for $n = 5$. Guess what the data abscissae x_i are.*

The Lagrange polynomials form an ideally conditioned basis $\phi_j(x) = L_j(x)$ for all polynomials of degree at most n. In the case of Lagrange interpolation, the matrix elements in the system (10.1) on page 297 are $\phi_j(x_i) = L_j(x_i)$, and we see that what was a potentially problematic Vandermonde matrix for the monomial basis in Section 10.2 is now an *identity* matrix. Thus, the system (10.1) (which is never formed as such) yields the solution $c_j = y_j,\ j = 1,\ldots,n$.

Note that L_j has n zeros and thus $n-1$ extrema (alternating minima and maxima). Incidentally, $L_j(x_j) = 1$ need not be a maximum for $L_j(x)$.

[41] For each j, set $y_i \leftarrow 1$ if $i = j$, $y_i \leftarrow 0$ if $i \neq j$, $i = 0, 1, \ldots, n$.

Algorithm: Lagrange Polynomial Interpolation.

1. *Construction*: Given data $\{(x_i, y_i)\}_{i=0}^n$, compute barycentric weights $w_j = 1/\prod_{i\neq j}(x_j - x_i)$, and also the quantities $w_j y_j$, for $j = 0, 1, \ldots n$.
2. *Evaluation*: Given an evaluation point x not equal to one of the data points $\{x_i\}_{i=0}^n$, compute
$$p(x) = \frac{\sum_{j=0}^n \frac{w_j y_j}{(x-x_j)}}{\sum_{j=0}^n \frac{w_j}{(x-x_j)}}.$$

Construction and evaluation

The discussion so far should make it clear what the Lagrange polynomials are and how they work for interpolation. Next, let us devise a way to carry out the construction and evaluation stages efficiently. This would be important if n is large.

The construction stage involves what does not depend on an evaluation point x. Here this means constructing the denominators of the $n+1$ Lagrange polynomials. Let us then define

$$\rho_j = \prod_{i\neq j}(x_j - x_i), \quad w_j = \frac{1}{\rho_j}, \quad j = 0, 1, \ldots, n.$$

This construction requires about n^2 flops. The quantities w_j are called **barycentric weights**.

Moving on to the evaluation stage, for a given point x different from the abscissae at which the value of $p(x)$ is required, note that all numerators of the $L_j(x)$ involve small variations of the function

$$\psi(x) = (x - x_0)\cdots(x - x_n) = \prod_{i=0}^n (x - x_i).$$

The interpolant is then given by

$$p(x) = \psi(x) \sum_{j=0}^n \frac{w_j y_j}{(x - x_j)}.$$

Clearly, for each argument value x the evaluation costs $\mathcal{O}(n)$ flops (about $5n$, to be more precise).

A slight further modification, essentially for the purpose of beauty, can be made by observing that for the particular function $f(x)$ that equals 1 everywhere we have $y_j = 1$ for all j. Moreover, the interpolant is also identically 1, so

$$1 = \psi(x) \sum_{j=0}^n \frac{w_j \cdot 1}{(x - x_j)}$$

for any x. This determines $\psi(x)$ in terms of quantities that are computed anyway. The resulting formula is called **barycentric interpolation**, leading to the algorithm given on this page. It would be worth your while to apply this algorithm to Example 10.2 and check what intermediate quantities arise.

Specific exercises for this section: Exercises 4–6.

10.4 Divided differences and Newton's form

In this section we continue to consider polynomial interpolation at $n+1$ data points with distinct abscissae. Yet another representation for polynomial interpolation is introduced (it's the last one: promise!), because our previous representations do not cover two important aspects. The first is the desirability of introducing interpolation data (x_i, y_i) one pair at a time, rather than all at once from the start. The other aspect that we have avoided thus far is estimating the error in the interpolation approximation, and the developments here help in setting up the discussion in Section 10.5.

The Newton polynomial basis

We have seen the standard monomial basis, $\{\phi_j(x) = x^j\}_{j=0}^n$. This has led to a relatively inferior procedure for constructing $p = p_n$ but an easy procedure for evaluating $p_n(x)$ at a given argument x. On the other hand, with the Lagrange polynomial basis, $\{\phi_j(x) = \prod_{i\neq j, i=0}^n \frac{(x-x_i)}{(x_j-x_i)}\}_{j=0}^n$, the construction stage is easy, but the evaluation of $p_n(x)$ is relatively involved conceptually. The Newton polynomial basis can be viewed as a useful compromise: we set

$$\phi_j(x) = \prod_{i=0}^{j-1}(x - x_i), \quad j = 0, 1, \ldots, n.$$

The discussion that follows illustrates the merits of this choice.

Example 10.3. For a quadratic interpolant we have the basis functions

$$\phi_0(x) = 1, \ \phi_1(x) = x - x_0, \ \phi_2(x) = (x - x_0)(x - x_1).$$

Consider again the same data set as in Examples 10.1 and 10.2, namely, $(1,1)$, $(2,3)$, and $(4,3)$. Then the interpolation condition at $x_0 = 1$ yields

$$1 = p(1) = c_0\phi_0(1) + c_1\phi_1(1) + c_2\phi_2(1) = c_0 \cdot 1 + c_1 \cdot 0 + c_2 \cdot 0 = c_0.$$

Next, with $c_0 = 1$, the interpolation condition at $x_1 = 2$ reads

$$3 = p(2) = 1 + c_1(2-1) + c_2 \cdot 0,$$

yielding $c_1 = 2$.

Finally, the third interpolation condition gives

$$3 = p(4) = 1 + 2(4-1) + c_2(4-1)(4-2),$$

yielding $c_2 = -\frac{4}{6}$. Therefore, the interpolating polynomial is

$$p_2(x) = 1 + 2(x-1) - \frac{2}{3}(x-1)(x-2).$$

At $x = 3$ this evaluates, of course, to $p_2(3) = \frac{11}{3}$, which is the same value as in Examples 10.1 and 10.2, because the interpolating polynomial is unique: only its form changes with the representation. As in our previous examples, we encourage the skeptics (and a healthy dose of skepticism is a trait of a good scientist!) to open the brackets and verify that the same polynomial as in those previous examples is obtained, yet again.

The important point to take away from this example is that to evaluate c_0 we used only the first data point, and to evaluate c_1 we needed only the first two. ∎

Existence of an adaptive interpolant

The most important feature of the Newton representation is that it is evolutionary, or recursive: having determined $p_{n-1}(x)$ that interpolates at the first n data points, we use it to cheaply construct $p_n(x)$ which interpolates all previous data as well as (x_n, y_n). Thus, we do not need to know all data points, or determine the degree n, ahead of time. Rather, we can do this *adaptively*. This feature may be of great value when working with, say, laboratory measurements where not all data become available at once. See also Exercises 9 and 10.

The present form of $p_n(x)$ is

$$p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}).$$

Is it always possible to determine the coefficients c_j? And if yes, is this particular representation unique?

The (affirmative) answers can be obtained upon considering the general system (10.1) on page 297. Given the form of the basis functions, we clearly have in the present case

$$\phi_j(x_i) = 0,\ i = 0, 1, \ldots, j-1, \quad \phi_j(x_j) \neq 0.$$

Hence, the matrix in (10.1) is *lower triangular* and the elements on its diagonal are nonzero. Since the determinant of a triangular matrix is the product of its diagonal elements we obtain that for arbitrary (distinct) data abscissae the matrix is nonsingular. Hence there is a unique solution for the unknown coefficients $c_0, \ldots, c_n$ in terms of the data $y_0, \ldots, y_n$.

The argument above relies on basic notions of linear algebra. In fact, an efficient *forward substitution* algorithm given on page 96 exists as well. Thus, we could form the system (10.1) and apply the algorithm of Section 5.1 to end the present discussion right here. However, we proceed instead to carry out the forward substitution algorithm symbolically, without forming (10.1) and without relying on knowledge from Chapter 5, because this results in important additional information regarding approximation processes as well as a more direct, efficient algorithm.

Representation in terms of divided differences

For ease of notation let us switch below from y_i to $f(x_i)$: we want to keep track of which data is used through the recursive construction process. Thus we proceed as follows.

- Determine c_0 using the condition $p_n(x_0) = f(x_0)$:

$$\begin{aligned} f(x_0) = p_n(x_0) &= c_0 + 0 + \cdots + 0 = c_0 \\ \Rightarrow c_0 &= f(x_0). \end{aligned}$$

- Next, determine c_1 using the condition $p_n(x_1) = f(x_1)$:

$$\begin{aligned} f(x_1) = p_n(x_1) &= c_0 + c_1(x_1 - x_0) + 0 + \cdots + 0 = c_0 + c_1(x_1 - x_0) \\ \Rightarrow c_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}. \end{aligned}$$

- Next, impose also the condition $p_n(x_2) = f(x_2)$. With c_0 and c_1 already determined this yields a condition involving c_2 alone. Please verify (you may use the statement of Exercise 13 for this purpose) that the result can be arranged as

$$c_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}.$$

- Continue this process along until all the coefficients c_j have been determined.

The coefficient c_j of the interpolating polynomial in Newton's form is called the jth *divided difference*, denoted $f[x_0,x_1,\ldots,x_j]$. So we write

$$f[x_0] = c_0,\ f[x_0,x_1] = c_1,\ \ldots,\ f[x_0,x_1,\ldots,x_n] = c_n.$$

This explicitly indicates which of the data points each coefficient c_j depends upon.

Note: The notation of divided differences is rather detailed, perhaps in an unappealing way at first. But please hang on, as several important ideas soon make an appearance.

Thus, the *Newton divided difference interpolation formula* in its full glory is given by

$$\begin{aligned} p_n(x) &= f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) \\ &\quad + \cdots + f[x_0,x_1,\ldots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1}) \\ &= \sum_{j=0}^{n} \left(f[x_0,x_1,\ldots,x_j] \prod_{i=0}^{j-1} (x-x_i) \right). \end{aligned}$$

The divided difference coefficients satisfy the recursive formula

$$f[x_0,x_1,\ldots,x_j] = \frac{f[x_1,x_2,\ldots,x_j] - f[x_0,x_1,\ldots,x_{j-1}]}{x_j - x_0}.$$

See Exercise 7. More generally, x_0 can be replaced by x_i for any $0 \le i < j$. The resulting formula is given on this page. This recursion means that we do not have to build the coefficients from scratch each time we add a new interpolation point.

Divided Differences.
Given points $x_0,x_1,\ldots,x_n$, for arbitrary indices $0 \le i < j \le n$, set

$$\begin{aligned} f[x_i] &= f(x_i), \\ f[x_i,\ldots,x_j] &= \frac{f[x_{i+1},\ldots,x_j] - f[x_i,\ldots,x_{j-1}]}{x_j - x_i}. \end{aligned}$$

Divided difference table and interpolant evaluation

How do we use all of these recurrences and formulas in practice? Note that in order to compute $\gamma_{n,n} = f[x_0,x_1,\ldots,x_n]$ we must compute all of

$$\gamma_{j,l} = f[x_{j-l},x_{j-l+1},\ldots,x_j], \quad 0 \le l \le j \le n.$$

Thus, we construct a *divided difference table*, which is a lower triangular array depicted next.

i	x_i	$f[x_i]$	$f[x_{i-1},x_i]$	$f[x_{i-2},x_{i-1},x_i]$	$\cdots$	$f[x_{i-n},\ldots,x_i]$
0	x_0	$f(x_0)$				
1	x_1	$f(x_1)$	$\frac{f[x_1]-f[x_0]}{x_1-x_0}$			
2	x_2	$f(x_2)$	$\frac{f[x_2]-f[x_1]}{x_2-x_1}$	$f[x_0,x_1,x_2]$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
n	x_n	$f(x_n)$	$\frac{f[x_n]-f[x_{n-1}]}{x_n-x_{n-1}}$	$f[x_{n-2},x_{n-1},x_n]$	$\cdots$	$f[x_0,x_1,\ldots,x_n]$

Extracting the diagonal entries yields the coefficients $c_j = \gamma_{j,j} = f[x_0,\ldots,x_j]$ for the Newton interpolation polynomial.

Example 10.4. For the same problem as in Example 10.3 we have

$$f[x_0,x_1] = \frac{3-1}{2-1} = 2, \quad f[x_1,x_2] = \frac{3-3}{4-2} = 0, \quad f[x_0,x_1,x_2] = \frac{0-2}{4-1} = -\frac{2}{3}.$$

The corresponding divided difference table is

i	x_i	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$
0	1	1		
1	2	3	2	
2	4	3	0	$-\frac{2}{3}$

Thus, the interpolating polynomial $p_2(x)$ is as given in Example 10.3.

Note that the first two terms of $p_2(x)$ form the linear interpolant $p_1(x)$ obtained in Example 10.1. This demonstrates the specialty of the Newton basis. Also, in nested form we have

$$p_2(x) = 1 + (x-1)\left(2 - \frac{2}{3}(x-2)\right).$$

If we wish to add another data point, say, $(x_3, f(x_3)) = (5,4)$, we need only add another row to the divided difference table. We calculate

$$f[x_3] = 4, \quad f[x_2,x_3] = \frac{4-3}{5-4} = 1, \quad f[x_1,x_2,x_3] = \frac{1-0}{5-2} = \frac{1}{3},$$
$$f[x_0,x_1,x_2,x_3] = \frac{(1/3)-(-2/3)}{5-1} = \frac{1}{4}.$$

The corresponding table is

i	x_i	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$
0	1	1			
1	2	3	2		
2	4	3	0	$-\frac{2}{3}$	
3	5	4	1	$\frac{1}{3}$	$\frac{1}{4}$

Thus, for p_3 we have the expression

$$\begin{aligned} p_3(x) &= p_2(x) + f[x_0,x_1,x_2,x_3](x-x_0)(x-x_1)(x-x_2) \\ &= 1 + (x-1)\left(2 - \frac{2}{3}(x-2)\right) + \frac{1}{4}(x-1)(x-2)(x-4). \end{aligned}$$

In nested form, this reads

$$p_3(x) = 1 + (x-1)\left(2 + (x-2)\left(-\frac{2}{3} + \frac{1}{4}(x-4)\right)\right).$$

Obtaining a higher degree approximation is simply a matter of adding a term.

Figure 10.5 displays the two polynomials p_2 and p_3. Note that p_2 predicts a rather different value for $x = 5$ than the additional datum $f(x_3)$ later imposes, which explains the significant difference between the two interpolating curves. ∎

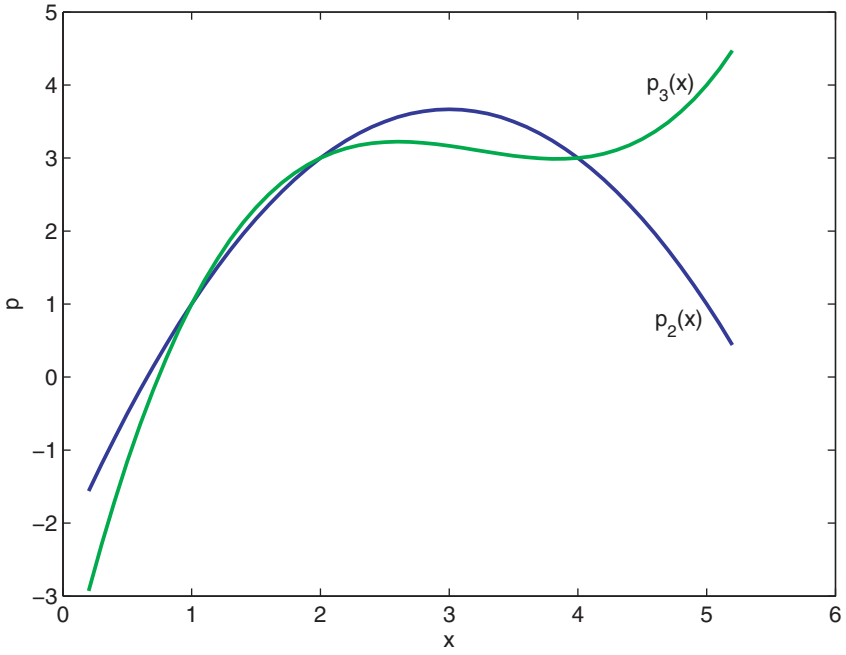


Figure 10.5. *The interpolating polynomials p_2 and p_3 for Example* 10.4.

Here are three MATLAB functions that implement the method described above. The first constructs the divided difference table and returns the coefficients of the polynomial interpolant.

```
function [coef,table] = divdif (xi, yi)
%
% function [coef,table] = divdif (xi, yi)
%
% Construct a divided difference table based on data points (xi,yi).
% Upon return, the Newton interpolation coefficients are in coef

np1 = length(xi); n = np1-1;
table = zeros(np1,np1); xi = shiftdim(xi); yi = shiftdim(yi);
% construct divided difference table one column at a time
table(1:np1,1) = yi;
for k = 2:np1
  table(k:np1,k) = (table(k:np1,k-1) - table(k-1:n,k-1)) ./ ...
                   (xi(k:np1) - xi(1:np1-k+1));
end
coef = diag(table);  % the diagonal elements of table
```

Note that the variable `table` is a two-dimensional array, and you can safely ignore the zero values in its strictly upper triangular part. In fact, you should be able to see from the code that it is not strictly necessary to store this entire table as such, but we do this anyway for illustration purposes.

Next, the following function uses nested evaluation to evaluate the polynomial interpolant in Newton's form:

```
function p = evalnewt (x, xi, coef)
%
%  function p = evalnewt (x, xi, coef)
%
%  evaluate at x the interpolating polynomial in Newton form
%  based on interpolation points xi and coefficients coef

np1 = length(xi);
p = coef(np1)*ones(size(x));
for j=np1-1:-1:1
  p = p.*(x - xi(j)) + coef(j);
end
```

An algorithm for a more general case (in which derivative values are also involved) is provided on page 321 in Section 10.7.

Example 10.4 also demonstrates the process of adding just one more data pair $(x_{n+1}, f(x_{n+1}))$ to an existing interpolant p_n of the first $n+1$ data pairs, obtaining

$$p_{n+1}(x) = p_n(x) + f[x_0, x_1, \ldots, x_n, x_{n+1}] \prod_{i=0}^{n} (x - x_i).$$

This is implemented in the following routine:

```
function [coef,table] = divdifadd (xi, yi, table)
%
% function [coef,table] = divdifadd (xi, yi, table)
%
% Construct one more row of an existing divided
% difference table, and add interpolation coefficient, based
% on one additional (last in xi and yi) data point

np1 = length(xi); n = np1 - 1;
table = [table zeros(n,1); yi(np1) zeros(1,n)];
for k=2:np1
  table(np1,k) = (table(np1,k-1) - table(n,k-1)) / ...
                 (xi(np1) - xi(np1-k+1));
end
coef = diag(table);
```

Here is the script (minus some plotting instructions) used to generate Figure 10.5:

```
x = .2:.01:5.2; % evaluation mesh
%quadratic interpolant
xi = [1,2,4]; yi = [1,3,3];
[coef2,table] = divdif(xi,yi);
%evaluate quadratic at x
y2 = evalnewt(x,xi,coef2);
%add data point
xi = [xi,5]; yi = [yi,4];
%cubic interpolant
[coef3,table] = divdifadd(xi,yi,table);
%evaluate cubic at x
y3 = evalnewt(x,xi,coef3);
plot (x,y2,'b',x,y3,'g')
```

Algorithms comparison

Although a precise operation count is not awfully important in practice (for one thing because n is not expected to be large, except perhaps in Section 10.6) we note that the construction of the divided difference table requires $n^2/2$ divisions and n^2 additions/subtractions. Moreover, nested evaluation of the Newton form requires roughly $2n$ multiplications and additions: these operation counts at each stage are at least as good as for the previous interpolants we saw.

Having described no less than three bases for polynomial interpolation, and in considerable detail at that, let us pause and summarize their respective properties in a table. The costs of construction and evaluation at one point are given to leading term in flop count.

Basis name	$\phi_j(x)$	Construction cost	Evaluation cost	Selling feature
Monomial	x^j	$\frac{2}{3}n^3$	$2n$	simple
Lagrange	$L_j(x)$	n^2	$5n$	$c_j = y_j$ most stable
Newton	$\prod_{i=0}^{j-1}(x - x_i)$	$\frac{3}{2}n^2$	$2n$	adaptive

Divided differences and derivatives

Note: The important connection between divided differences and derivatives, derived below, is given in the theorem on the current page. If you are prepared to believe it, then you can skip the rather technical discussion that follows.

The divided difference function can be viewed as an extension of the concept of function derivative. Indeed, if we consider a differentiable function $f(x)$ applied at two points z_0 and z_1, and imagine z_1 approaching z_0, then

$$f[z_0, z_1] = \frac{f(z_1) - f(z_0)}{z_1 - z_0} \to f'(z_0).$$

But even if z_0 and z_1 remain distinct, the Mean Value Theorem given on page 10 asserts that there is a point ζ between them such that

$$f[z_0, z_1] = f'(\zeta).$$

This directly relates the first divided difference with the first derivative of f, in case the latter exists.

A similar connection exists between the kth divided difference of f and its kth derivative. Let us show this in detail.

Theorem: Divided Difference and Derivative.
Let the function f be defined and have k bounded derivatives in an interval $[a,b]$ and let $z_0, z_1, \ldots, z_k$ be $k+1$ distinct points in $[a,b]$. Then there is a point $\zeta \in [a,b]$ such that

$$f[z_0, z_1, \ldots, z_k] = \frac{f^{(k)}(\zeta)}{k!}.$$

Thus, let $z_0, z_1, \ldots, z_k$ be $k+1$ distinct points contained in an interval on which a function f is defined and has k bounded derivatives. Note before we start that divided difference coefficients are symmetric in the arguments. That is, if $(\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_k)$ is a permutation of the abscissae $(z_0, z_1, \ldots, z_k)$, then

$$f[\hat{z}_0, \hat{z}_1, \ldots, \hat{z}_k] = f[z_0, z_1, \ldots, z_k];$$

see Exercise 13. Thus, we may as well assume that the points z_i are sorted in increasing order and write

$$a = z_0 < z_1 < \cdots < z_k = b.$$

Without computing anything let p_k be the interpolating polynomial of degree at most k satisfying $p_k(z_i) = f(z_i)$, and denote the error $e_k(x) = f(x) - p_k(x)$. Then $e_k(z_i) = 0$, $i = 0, 1, \ldots, k$, i.e., e_k has $k+1$ zeros (roots).

Applying Rolle's Theorem (see page 10) we observe that between each two roots z_{i-1} and z_i of $e_k(x)$ there is a root of the derivative error, $e_k'(x)$, for a total of k roots in the interval $[a, b]$. Repeatedly applying Rolle's Theorem further we obtain that the function $e_k^{(k-l)}(x)$ has $l+1$ roots in $[a, b]$, $l = k, k-1, \ldots, 1, 0$. In particular, $e_k^{(k)}$ has one such root, denoted ζ, where

$$0 = e_k^{(k)}(\zeta) = f^{(k)}(\zeta) - p_k^{(k)}(\zeta).$$

Next, we must characterize $p_k^{(k)}$, the kth derivative of the interpolating polynomial of degree k. This task is straightforward if the Newton interpolation form is considered, because using this form we can write the interpolating polynomial as

$$p_k(x) = f[z_0, z_1, \ldots, z_k]\, x^k + q_{k-1}(x),$$

with $q_{k-1}(x)$ a polynomial of degree $< k$.

But the kth derivative of $q_{k-1}(x)$ vanishes everywhere: $q_{k-1}^{(k)} \equiv 0$. Hence, the only element that remains is the kth derivative of $f[z_0, z_1, \ldots, z_k]x^k$. The kth derivative of x^k is the constant $k!$ (why?), and hence $p_k^{(k)} = f[z_0, z_1, \ldots, z_k](k!)$. Substituting in the expression for $e_k^{(k)}(\zeta)$ we obtain the important formula

$$f[z_0, z_1, \ldots, z_k] = \frac{f^{(k)}(\zeta)}{k!}.$$

Thus we have proved the theorem stated on the facing page which methodically connects between divided differences and derivatives even when the data abscissae are not close to one another. ♦

This formula stands in line with the view that the Newton interpolation form is an extension of sorts of the Taylor series expansion. It proves useful in Section 10.7 as well as in estimating the interpolation error, a task to which we proceed without further ado.

Specific exercises for this section: Exercises 7–14.

10.5 The error in polynomial interpolation

In our discussion of divided differences and derivatives at the end of the previous section we have focused on data ordinates $f(x_i) = y_i$, $i = 0, 1, \ldots, n$, and paid no attention to any other value of $f(x)$. The function f that presumably gave rise to the interpolated data could even be undefined at other argument values. But now we assume that $f(x)$ is defined on an interval $[a, b]$ containing the interpolation points, and we wish to assess how large the difference between $f(x)$ and $p_n(x)$ may be at any point of this interval. For this we will also assume that some derivatives of $f(x)$ exist and are bounded in the interval of interest.

An expression for the error

Let us define the *error function* of our constructed interpolant as

$$e_n(x) = f(x) - p_n(x).$$

We can obtain an expression for this error using a simple trick: at any given point $x \in [a,b]$ where we want to evaluate the error in $p_n(x)$, pretend that x is another interpolation point! Then, from the formula in Section 10.4 connecting p_{n+1} to p_n, we obtain

$$f(x) = p_{n+1}(x) = p_n(x) + f[x_0, x_1, \ldots, x_n, x]\psi_n(x),$$

where

$$\psi_n(x) = \prod_{i=0}^{n}(x - x_i),$$

defined also on page 305. Therefore, the error is

$$e_n(x) = f(x) - p_n(x) = f[x_0, x_1, \ldots, x_n, x]\psi_n(x).$$

However, simple as this formula for the error is, it depends explicitly both on the data and on an individual evaluation point x. Hence we want a more general, qualitative handle on the error. So, we proceed to obtain further forms of error estimates and bounds.

Error estimates and bounds

The first thing we do is to replace the divided difference in the error formula by the corresponding derivative, assuming f is smooth enough. Thus, we note that if the approximated function f has $n+1$ bounded derivatives, then by the theorem given on page 312 there is a point $\xi = \xi(x)$, where $a \le \xi \le b$, such that $f[x_0, x_1, \ldots, x_n, x] = \frac{f^{(n+1)}(\xi)}{(n+1)!}$. This relationship is obtained upon identifying k with $n+1$ and $(z_0, z_1, \ldots, z_k)$ with $(x_0, \ldots, x_n, x)$ in that theorem. It yields the *error estimate*

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\psi_n(x).$$

Theorem: Polynomial Interpolation Error.
If p_n interpolates f at the $n+1$ points $x_0, \ldots, x_n$ and f has $n+1$ bounded derivatives on an interval $[a,b]$ containing these points, then for each $x \in [a,b]$ there is a point $\xi = \xi(x) \in [a,b]$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i).$$

Furthermore, we have the error bound

$$\max_{a \le x \le b} |f(x) - p_n(x)| \le \frac{1}{(n+1)!} \max_{a \le t \le b} |f^{(n+1)}(t)| \max_{a \le s \le b} \prod_{i=0}^{n} |s - x_i|.$$

Although we do not know $\xi = \xi(x)$, we can get a bound on the error at all evaluation points x in some interval $[a,b]$ containing the data abscissae $x_0, \ldots, x_n$ if we have an upper bound on

$$\|f^{(n+1)}\| = \max_{a \le t \le b} |f^{(n+1)}(t)|.$$

Likewise, we can get a bound on the interpolation error if in addition we maximize $|\psi_n(x)|$ for the given abscissae over the evaluation interval. Moreover, since a similar error expression holds for each evaluation point $x \in [a,b]$, the maximum error magnitude on this interval is also bounded by the same quantity. We obtain the Polynomial Interpolation Error Theorem given on the facing page.

Note that although the error expression above was derived using Newton's form, it is independent of the basis used for the interpolating polynomial. There is only one such polynomial, regardless of method of representation!

Practical considerations

In practice, f is usually unknown, and so are its derivatives. We can sometimes do the following:

- Gauge the accuracy of such an approximation by computing a sequence of polynomials $p_k(x), p_{k+1}(x), \ldots$ using different subsets of the points and comparing results to see how well they agree. (If they agree well, then this would suggest convergence, or sufficient accuracy.) This should work when f does not vary excessively and the interval $[a,b]$ containing the abscissae of the data as well as the evaluation points is not too large.

- Hold the data points in reserve to use for construction of higher order polynomials to estimate the error. Thus, if we have more than the $n+1$ data pairs used to construct $p_n(x)$, then we can evaluate p_n at these additional abscissae and compare against the given values to obtain an underestimate of the maximum error in our interpolant. We may need many extra points to appraise realistically, though.

Example 10.5. The following maximum daily temperatures (in Celsius) were recorded every third day during one August month in a Middle Eastern city:

Day	3	6	9	12	15	18	21	24	27
Temperature (C)	31.2	32.0	35.3	34.1	35.0	35.5	34.1	35.1	36.0

Note that these data values are *not* monotonically decreasing or increasing. Suppose we wish to estimate the maximum temperature at day $x = 13$ of that month.

It is useful to practice some judgment here: just because there is a table available with lots of data does not necessarily imply that they should *all* be used to get an idea of what is happening around a specific point. What we could do, for example, is construct a cubic ($n = 3$) polynomial using four specific points near $x = 13$. Following this strategy, we choose $x_0 = 9, x_1 = 12, x_2 = 15, x_3 = 18$. Doing so, we get $p_3(13) = 34.29$. If on the other hand we go for linear interpolation of the two closest neighbors, at $x = 12$ and $x = 15$, we get $p_1(13) = 34.4$. This provides some confidence that our calculations are probably on the mark.

In summary, it was hot on that day, too. ∎

How should you choose the data abscissae, if possible, for lower degree approximations?

To keep $|\psi_n(x)|$ small, try to cluster data near where you want to make the approximations. Also, avoid extrapolation if there is that option. Last but not least, treat polynomial interpolation as a

device for *local* approximation; for *global* interpolation of given data, use piecewise polynomials—see Chapter 11.

Specific exercises for this section: Exercises 15–16.

10.6 Chebyshev interpolation

Suppose that we are interested in a good quality interpolation of a given smooth function $f(x)$ on the entire interval $[a,b]$. We are free to choose the $n+1$ data abscissae, $x_0, x_1, \ldots, x_n$: how should we choose these points?

Consider the expression for the interpolation error from the previous section, specifically given on page 314. Let us further assume absence of additional information about ξ or even f itself, other than the assurance that it can be sampled anywhere and that its bounded $(n+1)$st derivative exists: this is a common situation in many applications. Then the best we can do to minimize the error $\max_{a\le x\le b} |f(x) - p_n(x)|$ is to choose the data abscissae so as to minimize the quantity $\max_{a\le x\le b} |\psi_n(x)|$.

The latter minimization task leads to the choice of **Chebyshev points**. These points are defined on the interval $[-1,1]$ by

$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0,\ldots,n.$$

For a general interval $[a,b]$ we apply the affine transformation that maps $[-1,1]$ onto $[a,b]$ to shift and scale the Chebyshev points. So, by the transformation

$$x = a + \frac{b-a}{2}(t+1), \quad t \in [-1,1],$$

we redefine the interpolation abscissae as

$$x_i \longleftarrow a + \frac{b-a}{2}(x_i+1), \quad i = 0,\ldots,n.$$

Interpolation error using Chebyshev points

Let us stay with the interval $[-1,1]$, then. The Chebyshev points are zeros (roots) of the *Chebyshev polynomial*, defined and discussed in detail in Section 12.4. Thus, the monic Chebyshev polynomial (i.e., the Chebyshev polynomial that is scaled so as to have its leading coefficient equal 1) is given by $\psi_n(x)$, where the x_i are the Chebyshev points. As explained in detail in Section 12.4, the maximum absolute value of this polynomial over the interpolation interval is 2^{-n}. Thus, the $n+1$ Chebyshev points defined above solve the **min-max** problem

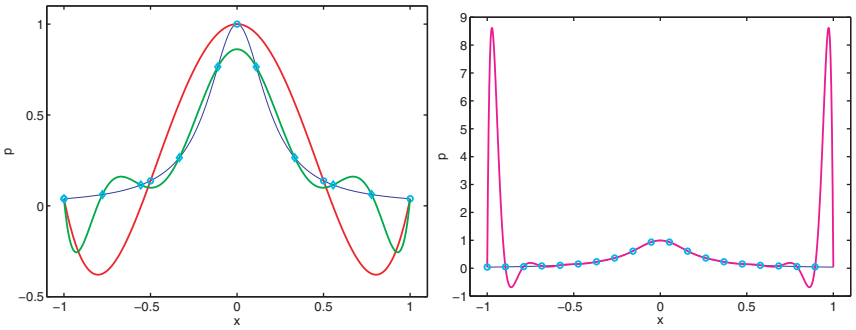
$$\beta = \min_{x_0,x_1,\ldots,x_n} \max_{-1\le x\le 1} |(x-x_0)(x-x_1)\cdots(x-x_n)|,$$

yielding the value $\beta = 2^{-n}$. This leads to the interpolation error bound

$$\max_{-1\le x\le 1} |f(x) - p_n(x)| \le \frac{1}{2^n(n+1)!} \max_{-1\le t\le 1} |f^{(n+1)}(t)|.$$

Example 10.6. A long time ago, C. Runge gave the innocent-looking example

$$f(x) = \frac{1}{1+25x^2}, \quad -1 \le x \le 1.$$



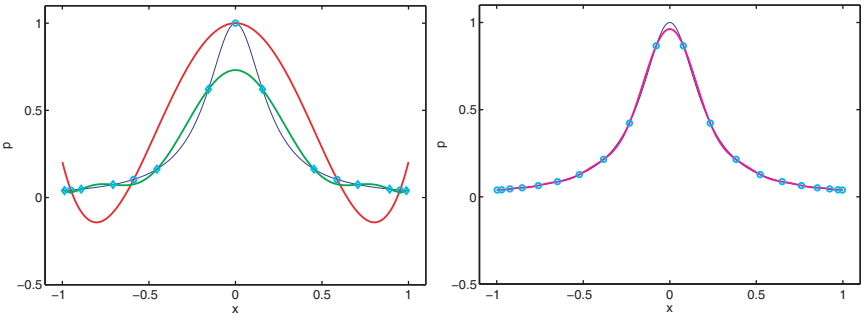
(a) $n = 4, 9$. (b) $n = 19$.

Figure 10.6. *Global polynomial interpolation at uniformly spaced abscissae can be bad. Here the blue curve with one maximum point is the Runge function of Example* 10.6*, and the other curves are polynomial interpolants of degree n.*

Results of polynomial interpolation at 5, 10 and 20 *equidistant* points are plotted in Figure 10.6. Note the different scaling on the y-axis between the two graphs: the approximation gets worse for larger n!

Calculating $\frac{f^{(n+1)}}{(n+1)!}$ we can see growth in the error term near the interval ends, explaining the fact that the results do not improve as the degree of the polynomial is increased.[42]

Next, we repeat the experiment, this time using the Chebyshev points for abscissae of the data points; nothing else changes. The much better results are plotted in Figure 10.7. ∎



(a) $n = 4, 9$. (b) $n = 19$.

Figure 10.7. *Polynomial interpolation at Chebyshev points, Example* 10.6*. Results are much improved as compared to Figure* 10.6*, especially for larger n.*

The improvement in Example 10.6 upon employing interpolation at Chebyshev points over polynomial interpolation at uniformly spaced points is rather remarkable. You may think that it

[42] Although there is a classical theorem by Weierstrass stating that it is possible to find arbitrarily close polynomial approximations to any continuous function.

is a bit "special," in that the Chebyshev points concentrate more near the interval ends, which is precisely where $\frac{f^{(n+1)}}{(n+1)!}$ gets large for this particular example, while our working assumption was that we don't know much about f. It is not difficult to construct examples where Chebyshev interpolation would not do well. Nonetheless, interpolation at Chebyshev points yields very good results in many situations, and important numerical methods for solving differential equations are based on it.

Note: A fuller explanation of the remarkable behavior of Chebyshev interpolation goes much deeper into approximation theory than we will in this book. A practical outcome is that this is the only place in the present chapter where it is natural to allow n to grow large. One conclusion from the latter is that the Lagrange (barycentric) interpolation (see page 305) should generally be employed for accurate Chebyshev interpolation.

Example 10.7. Suppose we wish to obtain a really accurate polynomial approximation for the function

$$f(x) = e^{3x}\sin(200x^2)/(1+20x^2), \quad 0 \le x \le 1.$$

A plot of this function is given in the top panel of Figure 10.8. It certainly does not look like a low degree polynomial.

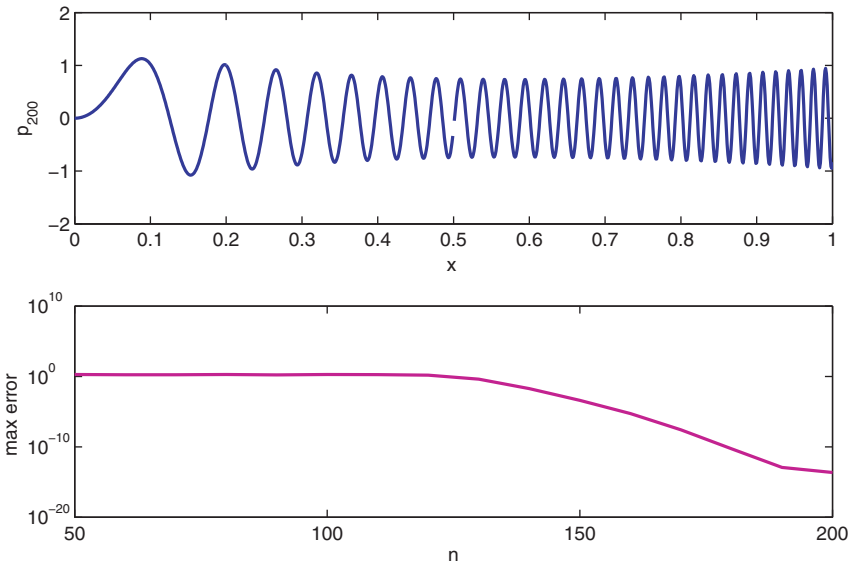


Figure 10.8. *Top panel: the function of Example* 10.7 *is indistinguishable from its polynomial interpolant at* 201 *Chebyshev points. Bottom panel: the maximum polynomial interpolation error as a function of the polynomial degree. When doubling the degree from* $n = 100$ *to* $n = 200$ *the error decreases from unacceptable* (> 1) *to almost rounding unit level.*

We have interpolated this function at $n+1$ Chebyshev points for n from 50 to 200 in increments of 10. The Lagrange representation was employed. The overall maximum error (measured over the uniform evaluation mesh `x = 0:.001:1`) is graphed in Figure 10.8 as well. Notice that for polynomial degrees as high as 100 and more the interpolation error is rather lousy. But as n is increased further the error eventually goes down, and rather fast: the error then looks like $\mathcal{O}(q^{-n})$

for some $q > 1$. This is called **spectral accuracy**. For $n = 200$ the error is already at rounding unit level.

For the Runge function in Example 10.6 things look even better: spectral accuracy is observed already for moderate values of n (see Exercise 19). But then that function is somewhat special.

In many applications extreme accuracy is not required, but reasonable accuracy is, and the lack of initial improvement in the bottom panel of Figure 10.8 can be unnerving: only at about $n = 150$ results are visually acceptable and the fact that for even larger n we start getting phenomenal accuracy is not always crucial, although no one would complain about too much accuracy. ∎

Specific exercises for this section: Exercises 17–20.

10.7 Interpolating also derivative values

Often, an interpolating polynomial $p_n(x)$ is desired that interpolates not only values of a function but also values of its *derivatives* at given points.

Example 10.8. A person throws a stone at an angle of 45°, and the stone lands after flying for five meters. The trajectory of the stone, $y = f(x)$, obeys the equations of motion, taking gravity and perhaps even air-drag forces into account. But we are after a quick approximation, and we know right away about f that $f(0) = 1.5$, $f'(0) = 1$, and $f(5) = 0$. Thus, we pass a quadratic through these points.

Using a monomial basis we write

$$p_2(x) = c_0 + c_1 x + c_2 x^2, \quad \text{hence } p_2'(x) = c_1 + 2c_2 x.$$

Substituting the data gives three equations for the coefficients, namely, $1.5 = p_2(0) = c_0$, $1 = p_2'(0) = c_1$, and

$$0 = p_2(5) = c_0 + 5c_1 + 25c_2.$$

The resulting interpolant (please verify) is

$$p_2(x) = 1.5 + x - 0.26x^2,$$

and it is plotted in Figure 10.9. ∎

The general problem

The general case involves more notation, but there is not much added complication beyond that. Suppose that there are $q + 1$ distinct abscissae

$$t_0, t_1, t_2, \ldots, t_q$$

and $q + 1$ nonnegative integers

$$m_0, m_1, m_2, \ldots, m_q,$$

and consider finding the unique **osculating polynomial**[43] of lowest degree satisfying

$$p_n^{(k)}(t_i) = f^{(k)}(t_i) \quad (k = 0, \ldots, m_i),\ i = 0, 1, \ldots, q.$$

By counting how many conditions there are to satisfy, clearly the degree of our interpolating polynomial is at most

$$n = \sum_{k=0}^{q} m_k + q.$$

[43] The word "osculating" is derived from Latin for "kiss."

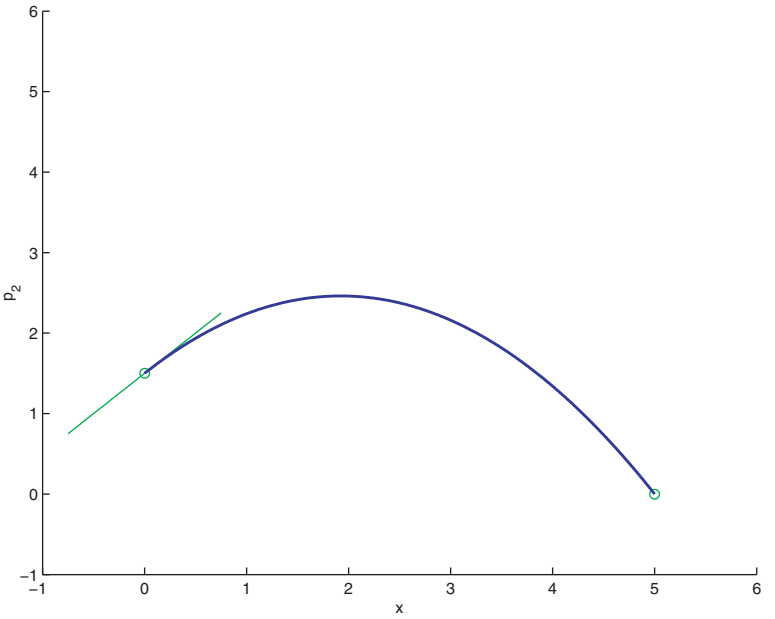


Figure 10.9. *A quadratic interpolant* $p_2(x)$ *satisfying* $p_2(0) = 1.5$, $p_2'(0) = 1$, *and* $p_2(5) = 0$.

Thus, the values of p_n and its first m_i derivatives coincide with the corresponding values of the first m_i derivatives of f at each of the interpolation points. This generalizes what we have seen before and more. Instances include the following:

1. If $q = n$, $m_i = 0$ for each i, then we have the polynomial interpolation at function values as we have seen before.
2. If $q = 0$, then we have the *Taylor polynomial* of degree m_0 at t_0.
3. If $n = 2q + 1$, $m_i = 1$ for each i, then we have **Hermite interpolation**.

This list provides the most useful instances, although the above formulation is *not* restricted to these choices.

Hermite cubic interpolation

The most popular osculating polynomial interpolant is the *Hermite cubic*, obtained by setting $q = m_0 = m_1 = 1$. Using the monomial basis we can write the interpolation conditions as

$$c_0 + c_1 t_0 + c_2 t_0^2 + c_3 t_0^3 = f(t_0), \quad c_1 + 2c_2 t_0 + 3c_3 t_0^2 = f'(t_0),$$
$$c_0 + c_1 t_1 + c_2 t_1^2 + c_3 t_1^3 = f(t_1), \quad c_1 + 2c_2 t_1 + 3c_3 t_1^2 = f'(t_1),$$

and solve these four linear equations for the coefficients c_j.

Constructing the osculating polynomial

A *general method* of constructing the osculating polynomial can be easily devised by extending Newton's form and the divided differences of Section 10.4. We define the set of abscissae by repeat-

ing each of the points t_i in the sequence $m_i + 1$ times, resulting in the sequence

$$(x_0, x_1, x_2, \ldots, x_n) = \left(\underbrace{t_0, t_0, \ldots, t_0}_{m_0+1}, \underbrace{t_1, \ldots, t_1}_{m_1+1}, \ldots, \underbrace{t_q, \ldots, t_q}_{m_q+1} \right).$$

The corresponding data values are

$$\begin{aligned}&(y_0, y_1, y_2, \ldots, y_n)\\ &= \left(f(t_0), f'(t_0), \ldots, f^{(m_0)}(t_0), f(t_1), \ldots, f^{(m_1)}(t_1), \ldots, f(t_q), \ldots, f^{(m_q)}(t_q) \right).\end{aligned}$$

Then use the Newton interpolating form

$$p_n(x) = \sum_{j=0}^{n} f[x_0, x_1, \ldots, x_j] \prod_{i=0}^{j-1} (x - x_i),$$

where

$$f[x_k, \ldots, x_j] = \begin{cases} \frac{f[x_{k+1}, \ldots, x_j] - f[x_k, \ldots, x_{j-1}]}{x_j - x_k}, & x_k \neq x_j, \\ \frac{f^{(j-k)}(x_k)}{(j-k)!}, & x_k = x_j, \end{cases}$$

for $0 < k \leq j \leq n$. Note that repeated values must be consecutive in the sequence $\{x_i\}_{i=0}^n$. The resulting algorithm is given on this page.

Algorithm: Polynomial Interpolation in Newton Form.

1. *Construction*: Given data $\{(x_i, y_i)\}_{i=0}^n$, where the abscissae are not necessarily distinct,

$$\begin{aligned}&\text{for } j = 0, 1, \ldots, n\\ &\quad \text{for } l = 0, 1, \ldots, j\\ &\qquad \gamma_{j,l} = \begin{cases} \frac{\gamma_{j,l-1} - \gamma_{j-1,l-1}}{x_j - x_{j-l}} & \text{if } x_j \neq x_{j-l}, \\ \frac{f^{(l)}(x_j)}{l!} & \text{otherwise.} \end{cases}\end{aligned}$$

2. *Evaluation*: Given an evaluation point x,

$$\begin{aligned}&p = \gamma_{n,n}\\ &\text{for } j = n-1, n-2, \ldots, 0,\\ &\quad p = p\,(x - x_j) + \gamma_{j,j}\end{aligned}$$

With this method definition, the expression for the error in polynomial interpolation derived in Section 10.5 also extends seamlessly to the case of osculating interpolation.

Example 10.9. For the function $f(x) = \ln(x)$ we have the values $f(1) = 0$, $f'(1) = 1$, $f(2) = .693147$, $f'(2) = .5$. Let us construct the corresponding Hermite cubic interpolant.

Using the simple but not general monomial basis procedure outlined above, we readily obtain the interpolant

$$p(x) = -1.53426 + 2.18223x - 0.761675x^2 + 0.113706x^3.$$

Using instead the general algorithm given on the previous page yields the alternative representation

$$p(x) = (x-1) - .30685(x-1)^2 + .113706(x-1)^2(x-2).$$

The function and its interpolant are depicted in Figure 10.10.

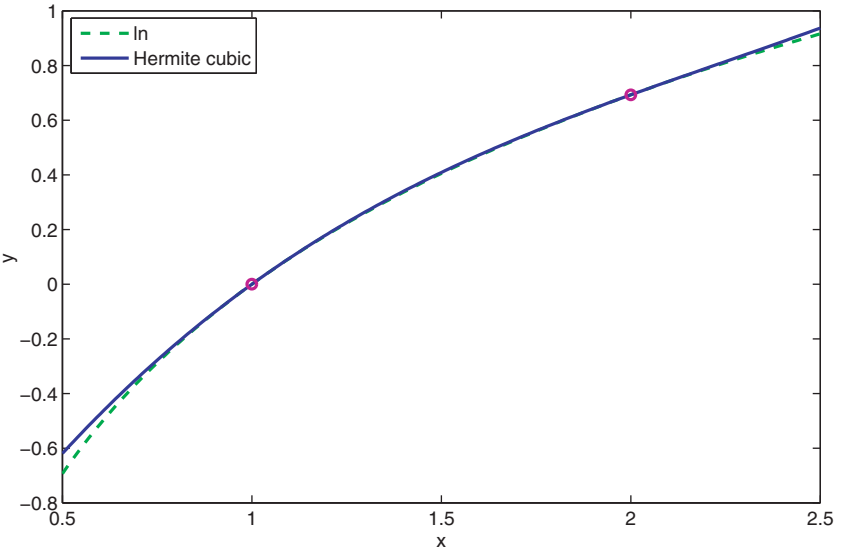


Figure 10.10. *The osculating Hermite cubic for* $\ln(x)$ *at the points* 1 *and* 2.

For example, $p_3(1.5) = .409074$, whereas $\ln(1.5) = .405465$.

To estimate the error at $x = 1.5$, pretending we cannot evaluate f there, we use the same expression as before: since $|f''''(x)| = |6x^{-4}| \le 6$, we have that the error is bounded by

$$\frac{1}{4}(1.5-1)^2(1.5-2)^2 = .015625.$$

This turns out to be about 4 times larger than the actual error. ∎

Example 10.10. Suppose we are given the five data values

t_i	$f(t_i)$	$f'(t_i)$	$f''(t_i)$
8.3	17.564921	3.116256	0.120482
8.6	18.505155	3.151762	

We set up the divided difference table

$$(x_0,x_1,x_2,x_3,x_4) = \left(\underbrace{8.3,8.3,8.3}_{m_0=2},\underbrace{8.6,8.6}_{m_1=1}\right),$$

$$f[x_0,x_1] = \frac{f'(t_0)}{1!} = f'(8.3), \quad f[x_1,x_2] = \frac{f'(t_0)}{1!},$$
$$f[x_0,x_1,x_2] = \frac{f''(t_0)}{2!} = \frac{f''(8.3)}{2}, \quad f[x_3,x_4] = \frac{f'(t_1)}{1!} = f'(8.6),$$

and so on. In the table below the originally given derivative values are underlined:

x_i	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot,\cdot]$
8.3	17.564921				
8.3	17.564921	3.116256			
8.3	17.564921	3.116256	0.060241		
8.6	18.505155	3.134113	0.059524	-0.002389	
8.6	18.505155	3.151762	0.058829	-0.002319	0.000233

The resulting quartic interpolant is

$$\begin{aligned} p_4(x) &= \sum_{k=0}^{4} f[x_0,\ldots,x_k]\prod_{j=0}^{k-1}(x-x_j) \\ &= 17.564921 + 3.116256(x-8.3) + 0.060241(x-8.3)^2 \\ &\quad - 0.002389(x-8.3)^3 + 0.000233(x-8.3)^3(x-8.6). \end{aligned}$$ ∎

Specific exercises for this section: Exercises 21–25.

10.8 Exercises

0. **Review questions**

   (a) Distinguish between the terms data fitting, interpolation, and polynomial interpolation.

   (b) Distinguish between (discrete) data fitting and approximating a given function.

   (c) What are basis functions? Does an approximant $v(x)$ that is written as a linear combination of basis functions have to be linear in x?

   (d) An interpolating polynomial is unique regardless of the choice of the basis. Explain why.

   (e) State one advantage and two disadvantages of using the monomial basis for polynomial interpolation.

   (f) What are Lagrange polynomials? How are they used for polynomial interpolation?

   (g) What are barycentric weights?

(h) State the main advantages and the main disadvantage for using the Lagrange representation.

(i) What is a divided difference table and how is it constructed?

(j) Write down the formula for polynomial interpolation in Newton form.

(k) State two advantages and two disadvantages for using the Newton representation for polynomial interpolation.

(l) Describe the linear systems that are solved for the monomial basis, the Lagrange representation, and the Newton representation.

(m) Describe the connection between the kth divided difference of a function f and its kth derivative.

(n) Provide an expression for the error in polynomial interpolation as well as an error bound expression.

(o) How does the smoothness of a function and its derivatives affect the quality of polynomial interpolants that approximate it, in general?

(p) Give an example where the error bound is attained.

(q) When we interpolate a function f given only data points, i.e., we do not know f or its derivatives, how can we gauge the accuracy of our approximation?

(r) What are Chebyshev points and why are they important?

(s) Describe osculating interpolation. How is it different from the usual polynomial interpolation?

(t) What is a Hermite cubic interpolant?

1. Derive the linear interpolant through the two data points $(1.0, 2.0)$ and $(1.1, 2.5)$. Derive also the quadratic interpolant through these two pairs as well as $(1.2, 1.5)$. Show that the situation can be depicted as in Figure 10.11.

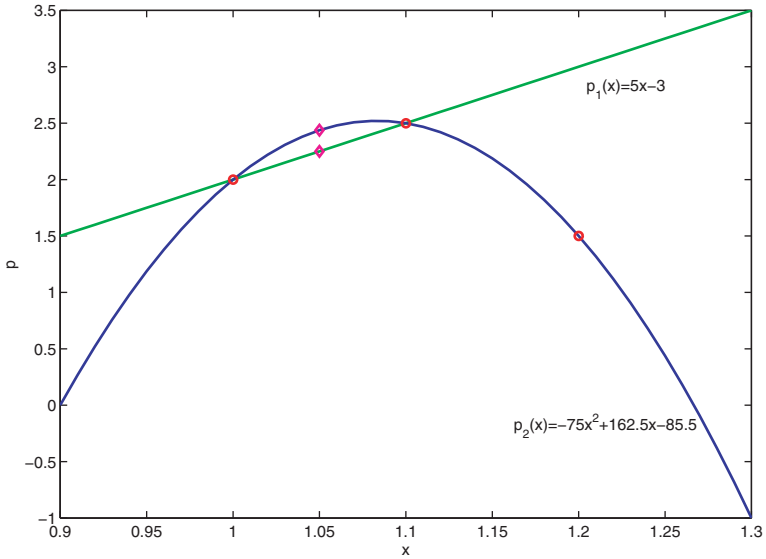


Figure 10.11. *Quadratic and linear polynomial interpolation.*

2. Some modeling considerations have mandated a search for a function

$$u(x) = \gamma_0 e^{\gamma_1 x + \gamma_2 x^2},$$

where the unknown coefficients γ_1 and γ_2 are expected to be nonpositive. Given are data pairs to be interpolated, (x_0, z_0), (x_1, z_1), and (x_2, z_2), where $z_i > 0$, $i = 0, 1, 2$. Thus, we require $u(x_i) = z_i$.

The function $u(x)$ is not linear in its coefficients, but $v(x) = \ln(u(x))$ is linear in its.

(a) Find a quadratic polynomial $v(x)$ that interpolates appropriately defined three data pairs, and then give a formula for $u(x)$ in terms of the original data.

[This is a pen-and-paper item; the following one should consume much less of your time.]

(b) Write a script to find u for the data $(0, 1)$, $(1, .9)$, $(3, .5)$. Give the coefficients γ_i and plot the resulting interpolant over the interval $[0, 6]$. In what way does the curve behave qualitatively differently from a quadratic?

3. Use the known values of the function $\sin(x)$ at $x = 0, \pi/6, \pi/4, \pi/3$ and $\pi/2$ to derive an interpolating polynomial $p(x)$. What is the degree of your polynomial?

What is the interpolation error magnitude $|p(1.2) - \sin(1.2)|$?

4. Given $n + 1$ data pairs $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, define for $j = 0, 1, \ldots, n$ the functions $\rho_j = \prod_{i \neq j}(x_j - x_i)$, and let also $\psi(x) = \prod_{i=0}^{n}(x - x_i)$.

(a) Show that

$$\rho_j = \psi'(x_j).$$

(b) Show that the interpolating polynomial of degree at most n is given by

$$p_n(x) = \psi(x) \sum_{j=0}^{n} \frac{y_j}{(x - x_j)\psi'(x_j)}.$$

5. Construct a polynomial $p_3(t)$ of degree at most three *in Lagrange form* that interpolates the data

t	−1.1	1.1	2.2	0.0
y	0.0	6.75	0.0	0.0

6. Given the four data points $(-1, 1), (0, 1), (1, 2), (2, 0)$, determine the interpolating cubic polynomial

- using the monomial basis;
- using the Lagrange basis;
- using the Newton basis.

Show that the three representations give the same polynomial.

7. For the Newton basis, prove that $c_j = f[x_0, x_1, \ldots, x_j]$ satisfies the recursion formula as stated on page 308 in Section 10.4.

[This proof is short but may be challenging.]

8. A secret formula for eternal youth, $f(x)$, was discovered by Dr. Quick, who has been working in our biotech company. However, Dr. Quick has disappeared and is rumored to be negotiating with a rival organization.

   From the notes that Dr. Quick left behind in his hasty exit it is clear that $f(0) = 0$, $f(1) = 2$, and that $f[x_0, x_1, x_2] = 1$ for *any* three points x_0, x_1, x_2. Find $f(x)$.

9. Joe had decided to buy stocks of a particularly promising Internet company. The price per share was \$100, and Joe subsequently recorded the stock price at the end of each week. With the abscissae measured in days, the following data were acquired:
   $(0,100), (7,98), (14,101), (21,50), (28,51), (35,50)$.

   In attempting to analyze what happened, it was desired to approximately evaluate the stock price a few days before the crash.

   (a) Pass a linear interpolant through the points with abscissae 7 and 14. Then add to this data set the value at 0 and (separately) the value at 21 to obtain two quadratic interpolants. Evaluate all three interpolants at $x = 12$. Which one do you think is the most accurate? Explain.

   (b) Plot the two quadratic interpolants above, together with the data (without a broken line passing through the data) over the interval $[0, 21]$. What are your observations?

10. Suppose we are given 137 uniformly spaced data pairs at distinct abscissae: $(x_i, y_i), i = 0, 1, \ldots, 136$. These data are thought to represent a function which is piecewise smooth; that is, the unknown function $f(x)$ which gave rise to these data values has many bounded derivatives everywhere except for a few points where it jumps discontinuously. (Imagine drawing a curve smoothly from left to right, mixed with lifting the pen and moving it vertically a few times.) For each subinterval $[x_{i-1}, x_i]$ we want to pass the hopefully best cubic $p_3(x)$ for an accurate interpolation of $f(x)$ at points $x_{i-1} < x < x_i$. This involves choosing good neighbors to interpolate at.

    Propose an algorithm for this task. Justify.

11. Given a sequence $y_0, y_1, y_2, \ldots$, define the *forward difference operator* Δ by

$$\Delta y_i = y_{i+1} - y_i.$$

    Powers of Δ are defined recursively by

$$\begin{aligned} \Delta^0 y_i &= y_i, \\ \Delta^j y_i &= \Delta(\Delta^{j-1} y_i), \quad j = 1, 2, \ldots. \end{aligned}$$

    Thus, $\Delta^2 y_i = \Delta(y_{i+1} - y_i) = y_{i+2} - 2y_{i+1} + y_i$, etc.

    Consider polynomial interpolation at equispaced points, $x_i = x_0 + ih$, $i = 0, 1, \ldots, n$.

    (a) Show that

$$f[x_0, x_1, \ldots, x_j] = \frac{1}{j! h^j} \Delta^j f(x_0).$$

    [Hint: Use mathematical induction.]

(b) Show that the interpolating polynomial of degree at most n is given by the *Newton forward difference* formula

$$p_n(x) = \sum_{j=0}^{n} \binom{s}{j} \Delta^j f(x_0),$$

where $s = \frac{x-x_0}{h}$ and $\binom{s}{j} = \frac{s(s-1)\cdots(s-j+1)}{j!}$ (with $\binom{s}{0} = 1$).

12. Given a sequence $y_0, y_1, y_2, \ldots$, define the *backward difference operator* ∇ by

$$\nabla y_i = y_i - y_{i-1}.$$

Powers of ∇ are defined recursively by

$$\begin{aligned} \nabla^0 y_i &= y_i, \\ \nabla^j y_i &= \nabla(\nabla^{j-1} y_i), \quad j = 1, 2, \ldots. \end{aligned}$$

Thus, $\nabla^2 y_i = \nabla(y_i - y_{i-1}) = y_i - 2y_{i-1} + y_{i-2}$, etc.

Consider polynomial interpolation at equispaced points, $x_i = x_0 + ih$, $i = 0, 1, \ldots, n$.

(a) Show that

$$f[x_n, x_{n-1}, \ldots, x_{n-j}] = \frac{1}{j!h^j} \nabla^j f(x_n).$$

[Hint: Use mathematical induction.]

(b) Show that the interpolating polynomial of degree at most n is given by the *Newton backward difference* formula

$$p_n(x) = \sum_{j=0}^{n} (-1)^j \binom{s}{j} \nabla^j f(x_n),$$

where $s = \frac{x_n - x}{h}$ and $\binom{s}{j} = \frac{s(s-1)\cdots(s-j+1)}{j!}$ (with $\binom{s}{0} = 1$).

13. Let $(\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_k)$ be a permutation of the abscissae $(x_0, x_1, \ldots, x_k)$. Show that

$$f[\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_k] = f[x_0, x_1, \ldots, x_k].$$

[Hint: Consider the kth derivative of the unique polynomial of degree k interpolating f at these $k+1$ points, regardless of how they are ordered.]

14. Let the points $x_0, x_1, \ldots, x_n$ be fixed and consider the divided difference $f[x_0, x_1, \ldots, x_n, x]$ as a function of x. (This function appears as part of the expression for the error in polynomial interpolation.)

Suppose next that $f(x)$ is a polynomial of degree m. Show that

- if $m \le n$, then $f[x_0, x_1, \ldots, x_n, x] \equiv 0$;
- otherwise $f[x_0, x_1, \ldots, x_n, x]$ is a polynomial of degree $m - n - 1$.

[Hint: If $m > n$, show it first for the case $n = 0$. Then proceed by induction, examining the function $g(x) = f[x_1, \ldots, x_n, x]$.]

15. Suppose we want to approximate the function e^x on the interval $[0,1]$ by using polynomial interpolation with $x_0 = 0, x_1 = 1/2$ and $x_2 = 1$. Let $p_2(x)$ denote the interpolating polynomial.

    (a) Find an upper bound for the error magnitude

    $$\max_{0 \le x \le 1} |e^x - p_2(x)|.$$

    (b) Find the interpolating polynomial using your favorite technique.

    (c) Plot the function e^x and the interpolant you found, both on the same figure, using the commands `plot`.

    (d) Plot the error magnitude $|e^x - p_2(x)|$ on the interval using logarithmic scale (the command `semilogy`) and verify by inspection that it is below the bound you found in part (a).

16. For the problem of Exercise 3, find an error bound for the polynomial interpolation on $[0, \pi/2]$.

    Compare your error bound to the actual interpolation error at $x = 1.2$.

17. Construct two simple examples for any positive integer n, one where interpolation at $n+1$ equidistant points is more accurate than interpolation at $n+1$ Chebyshev points and one where Chebyshev interpolation is more accurate.

    Your examples should be convincing without the aid of any computer implementation.

18. (a) Interpolate the function $f(x) = \sin(x)$ at 5 Chebyshev points over the interval $[0, \pi/2]$. Compare your results to those of Exercises 3 and 16.

    (b) Repeat the interpolation, this time using 5 Chebyshev points over the interval $[0, \pi]$. Plot $f(x)$ as well as the interpolant. What are your conclusions?

19. Interpolate the Runge function of Example 10.6 at Chebyshev points for n from 10 to 170 in increments of 10. Calculate the maximum interpolation error on the uniform evaluation mesh `x = -1:.001:1` and plot the error vs. polynomial degree as in Figure 10.8 using `semilogy`. Observe spectral accuracy.

20. The **Chebyshev extremum points** are close cousins of the Chebyshev points. They are defined on the interval $[-1, 1]$ by

    $$x_i = \xi_i = \cos\left(\frac{i}{n}\pi\right), \quad i = 0, 1, \ldots, n.$$

    See Section 12.4 for more on these points. Like the Chebyshev points, the extremum Chebyshev points tend to concentrate more near the interval's ends.

    Repeat the runs of Exercises 18 and 19, as well as those reported in Example 10.7, interpolating at the $n+1$ Chebyshev extremum points by a polynomial of degree at most n.

    Compare against the corresponding results using the Chebyshev points and show that although the Chebyshev extremum points are slightly behind, it's never by much. In fact, the famed Chebyshev stability and spectral accuracy are also observed here!

21. Interpolate the function $f(x) = \ln(x)$ by passing a cubic through the points $x_i = (0.1, 1, 2, 2.9)$. Evaluate your interpolant at $x = 1.5$ and compare the result against the exact value and against the value of the osculating Hermite cubic through the points $x_i = (1, 1, 2, 2)$, given in Example 10.9.

    Explain your observations by looking at the error terms for both interpolating cubic polynomials.

22. For some function f, you have a table of extended divided differences of the form

i	z_i	$f[\cdot]$	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$
0	5.0	$f[z_0]$			
1	5.0	$f[z_1]$	$f[z_0,z_1]$		
2	6.0	4.0	5.0	-3.0	
3	4.0	2.0	$f[z_2,z_3]$	$f[z_1,z_2,z_3]$	$f[z_0,z_1,z_2,z_3]$

    Fill in the unknown entries in the table.

23. For the data in Exercise 22, what is the osculating polynomial $p_2(x)$ of degree at most 2 that satisfies

$$p_2(5.0) = f(5.0),\ p_2'(5.0) = f'(5.0),\ p_2(6.0) = f(6.0)?$$

24. (a) Write a script that interpolates $f(x) = \cosh(x) = \frac{e^x + e^{-x}}{2}$ with an osculating polynomial that matches both $f(x)$ and $f'(x)$ at abscissae $x_0 = 1$ and $x_1 = 3$. Generate a plot (with logarithmic vertical axis) comparing $f(x)$ and the interpolating polynomial and another plot showing the error in your interpolant over this interval. Use the command `semilogy` for generating your graphs.

    (b) Modify the code to generate another interpolant that matches $f(x)$ and $f'(x)$ at the abscissae $x_0 = 1$, $x_1 = 2$, and $x_2 = 3$. Generate two more plots: comparison of function to interpolant, and error. Compare the quality of this new polynomial to the quality of the polynomial of part (a).

    (c) Now, modify the code to generate an interpolant that matches $f(x)$, $f'(x)$ and $f''(x)$ at the abscissae $x_0 = 1$ and $x_1 = 3$. Generate two more plots and comment on the quality of this approximation compared to the previous two interpolants.

25. A popular technique arising in methods for minimizing functions in several variables involves a *weak line search*, where an approximate minimum x^* is found for a function in one variable, $f(x)$, for which the values of $f(0)$, $f'(0)$, and $f(1)$ are given. The function $f(x)$ is defined for all nonnegative x, has a continuous second derivative, and satisfies $f(0) < f(1)$ and $f'(0) < 0$. We then interpolate the given values by a quadratic polynomial and set x^* as the minimum of the interpolant.

    (a) Find x^* for the values $f(0) = 1,\ f'(0) = -1,\ f(1) = 2$.

    (b) Show that the quadratic interpolant has a unique minimum satisfying $0 < x^* < 1$. Can you show the same for the function f itself?

26. Suppose we have computed an interpolant for data points $\{(x_i, y_i)\}_{i=0}^n$ using each of the three polynomial bases that we have discussed in this chapter (monomial, Lagrange, and Newton), and we have already constructed any necessary matrices, vectors, interpolation coefficients,

and/or basis functions. Suddenly, we realize that the final data point was wrong. Consider the following situations:

(a) The final data point should have been $(\tilde{x}_n, y_n)$ (that is, y_n is the same as before).

(b) The final data point should have been $(x_n, \tilde{y}_n)$ (x_n is the same as before).

For each of these two scenarios, determine the computational cost of computing the modified interpolants. Give your answers in the $\mathcal{O}$ notation (see page 7).

[A couple of the answers for parts of this question may require some "extra" knowledge of linear algebra.]

10.9 Additional notes

The fundamental role that polynomial interpolation plays in many different aspects of numerical computation is what gave rise to such a long chapter on what is essentially a relatively simple process. Many textbooks on numerical methods and analysis devote considerable space for this topic, and we mention Burden and Faires [11], Cheney and Kincaid [12], and the classical Conte and de Boor [13] for a taste.

Lagrange polynomials are used in the design of methods for numerical integration and solution of differential equations. We will see some of this in Chapters 14, 15, and 16. For much more see, e.g., Davis and Rabinowitz [17], Ascher and Petzold [5], and Ascher [3].

For polynomial interpolation at uniformly spaced (or equispaced) abscissae, where there is a spacing value h such that $x_i - x_{i-1} = h$ for all $i = 1, \ldots, n$, the divided difference table acquires a simpler form. Special names from the past are attached to this: there are the Newton forward and backward difference formulas, as well as Stirling's centered difference formula. See Exercises 11 and 12.

The use of divided differences and the Newton form interpolation in designing methods for complex problems is also prevalent. As elaborated also in some of the exercises, this is natural especially if we add interpolation points one at a time, for instance, because we wish to stay on one side of a discontinuity in the interpolated function (rather than crossing the discontinuity with an infinitely differentiable polynomial). Certain such methods for solving problems with shocks are called essentially nonoscillatory (ENO) schemes; see, e.g., [3].

Chebyshev polynomial interpolation is the most notable exception to the practical rule of keeping the degree n of interpolating polynomials low and using them locally, switching for higher accuracy to the methods of Chapter 11. At least for infinitely smooth functions $f(x)$, high degree Chebyshev polynomial interpolation can yield very high accuracy at a reasonable cost, as Example 10.7 and Exercise 19 indicate. This allows one to relate to smooth functions as objects, internally replacing them by their Chebyshev interpolants in a manner that is invisible to a user. Thus, "symbolic" operations such as differentiation, integration, and even solution of differential equations in one variable, are enabled. For work on this in a MATLAB context by Trefethen and coauthors, see http://www.mathworks.com/matlabcentral/fx_files/23972/10/content/chebfun/guide/html/guide4.html.

Chebyshev polynomials often arise in the numerical analysis of seemingly unrelated methods because of their min-max property. For instance, they feature prominently in the analysis of convergence rates for the famous conjugate gradient method introduced in Section 7.4; see, e.g., Greenbaum [32] or LeVeque [50]. They are also applicable for efficiently solving certain partial differential equations using methods called *spectral collocation*; see, e.g., Trefethen [69].