CHAPTER 13

Interpolation

CONTENTS

SO far we have derived methods for *analyzing* functions f, e.g., finding their minima and roots. Evaluating $f(\vec{x})$ at a particular $\vec{x} \in \mathbb{R}^n$ might be expensive, but a fundamental assumption of the methods we developed in previous chapters is that we can obtain $f(\vec{x})$ when we want it, regardless of $\vec{x}$.

There are many contexts in which this assumption is unrealistic. For instance, if we take a photograph with a digital camera, we receive an $n \times m$ grid of pixel color values sampling the continuum of light coming into the camera lens. We might think of a photograph as a continuous function from image position (x, y) to color (r, g, b), but in reality we only know the image value at nm separated locations on the image plane. Similarly, in machine learning and statistics, often we only are given samples of a function at points where we collected data, and we must interpolate to have values elsewhere; in a medical setting we may monitor a patient's response to different dosages of a drug but must predict what will happen at a dosage we have not tried explicitly.

In these cases, before we can minimize a function, find its roots, or even compute values $f(\vec{x})$ at arbitrary locations $\vec{x}$, we need a model for interpolating $f(\vec{x})$ to all of $\mathbb{R}^n$ (or some subset thereof) given a collection of samples $f(\vec{x}_i)$. Techniques for this *interpolation* problem are inherently approximate, since we do not know the true values of f, so instead we seek for the interpolated function to be smooth and serve as a reasonable prediction of function values. Mathematically, the definition of "reasonable" will depend on the particular application. If we want to evaluate $f(\vec{x})$ directly, we may choose an interpolant and sample positions $\vec{x}_i$ so that the distance of the interpolated $f(\vec{x})$ from the true values can be bounded above given smoothness assumptions on f; future chapters will estimate derivatives, integrals, and other properties of f from samples and may choose an interpolant designed to make these approximations accurate or stable.

In this chapter, we will assume that the values $f(\vec{x}_i)$ are known with complete certainty; in this case, we can think of the problem as extending f to the remainder of the domain without perturbing the value at any of the input locations. To contrast, the *regression* problem considered in §4.1.1 and elsewhere may forgo matching $f(\vec{x}_i)$ exactly in favor of making f more smooth.

13.1 INTERPOLATION IN A SINGLE VARIABLE

Before considering the general case, we will design methods for interpolating functions of a single variable $f : \mathbb{R} \to \mathbb{R}$. As input, we will take a set of k pairs (x_i, y_i) with the assumption $f(x_i) = y_i$; our job is to predict $f(x)$ for $x \notin \{x_1, \ldots, x_k\}$. Desirable *interpolants* $f(x)$ should be smooth and should interpolate the data points faithfully without adding extra features like spurious local minima and maxima.

We will take inspiration from linear algebra by writing $f(x)$ in a *basis*. The set of all possible functions $f : \mathbb{R} \to \mathbb{R}$ is far too large to work with and includes many functions that are not practical in a computational setting. Thus, we simplify the search space by forcing f to be written as a linear combination of building block basis functions. This formulation is familiar from calculus: The Taylor expansion writes functions in the basis of polynomials, while Fourier series use sine and cosine.

The construction and analysis of interpolation bases is a classical topic that has been studied for centuries. We will focus on practical aspects of choosing and using interpolation bases, with a brief consideration of theoretical aspects in §13.3. Detailed aspects of error analysis can be found in [117] and other advanced texts.

13.1.1 Polynomial Interpolation

Perhaps the most straightforward class of interpolation formulas assumes that $f(x)$ is in $\mathbb{R}[x]$, the set of polynomials. Polynomials are smooth, and we already have explored linear methods for finding a degree $k-1$ polynomial through k sample points in Chapter 4.

Example 4.3 worked out the details of such an interpolation technique. As a reminder, suppose we wish to find $f(x) \equiv a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1}x^{k-1}$ through the points $(x_1, y_1), \ldots, (x_k, y_k)$; here our unknowns are the values $a_0, \ldots, a_{k-1}$. Plugging in the expression $y_i = f(x_i)$ for each i shows that the vector $\vec{a}$ satisfies the $k \times k$ *Vandermonde* system:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_k \end{pmatrix}.$$

By this construction, degree $k-1$ polynomial interpolation can be accomplished using a $k \times k$ linear solve for $\vec{a}$ using the linear algorithms in Chapter 3. This method, however, is far from optimal for many applications.

As mentioned above, one way to think about the space of polynomials is that it can be spanned by a basis of functions. Just like writing vectors in $\mathbb{R}^n$ as linear combinations of linearly independent vectors $\vec{v}_1, \ldots, \vec{v}_n \in \mathbb{R}^n$, in our derivation of the Vandermonde matrix, we wrote polynomials in the *monomial basis* $\{1, x, x^2, \ldots, x^{k-1}\}$ for polynomials of degree $k-1$. Although monomials may be an obvious basis for $\mathbb{R}[x]$, they have limited properties useful for simplifying the polynomial interpolation problem. One way to visualize this issue is to plot the sequence of functions $1, x, x^2, x^3, \ldots$ for $x \in [0, 1]$; in this interval, as shown in

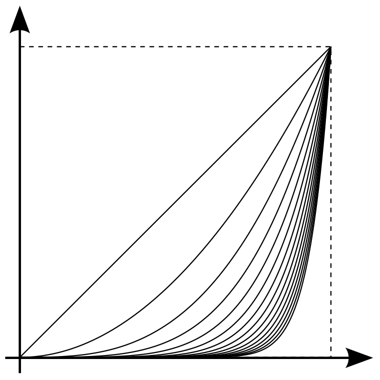


Figure 13.1 As k increases, the monomials x^k on $[0,1]$ begin to look more and more similar. This similarity creates poor conditioning for monomial basis problems like solving the Vandermonde system.

Figure 13.1, the functions x^k all start looking similar as k increases. As we know from our consideration of projection problems in Chapter 5, projection onto a set of similar-looking basis vectors can be unstable.

We may choose to write polynomials in a basis that is better suited to the problem at hand. Recall that we are given k pairs $(x_1, y_1), \ldots, (x_k, y_k)$. We can use these (fixed) points to define the *Lagrange interpolation* basis $\phi_1, \ldots, \phi_k$ by writing:

$$\phi_i(x) \equiv \frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)}.$$

Example 13.1 (Lagrange basis)**.** Suppose $x_1 = 0$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$. The Lagrange basis for this set of x_i's is:

$$\begin{aligned}
\phi_1(x) &= \frac{(x-2)(x-3)(x-4)}{-2\cdot -3 \cdot -4} = \frac{1}{24}(-x^3 + 9x^2 - 26x + 24)\\
\phi_2(x) &= \frac{x(x-3)(x-4)}{2\cdot(2-3)(2-4)} = \frac{1}{4}(x^3 - 7x^2 + 12x)\\
\phi_3(x) &= \frac{x(x-2)(x-4)}{3\cdot(3-2)\cdot(3-4)} = \frac{1}{3}(-x^3 + 6x^2 - 8x)\\
\phi_4(x) &= \frac{x(x-2)(x-3)}{4\cdot(4-2)\cdot(4-3)} = \frac{1}{8}(x^3 - 5x^2 + 6x).
\end{aligned}$$

This basis is shown in Figure 13.2.

As shown in this example, although we did not define it explicitly in the monomial basis $\{1, x, x^2, \ldots, x^{k-1}\}$, each ϕ_i is still a polynomial of degree $k-1$. Furthermore, the Lagrange basis has the following desirable property:

$$\phi_i(x_\ell) = \begin{cases} 1 & \text{when } \ell = i \\ 0 & \text{otherwise.} \end{cases}$$

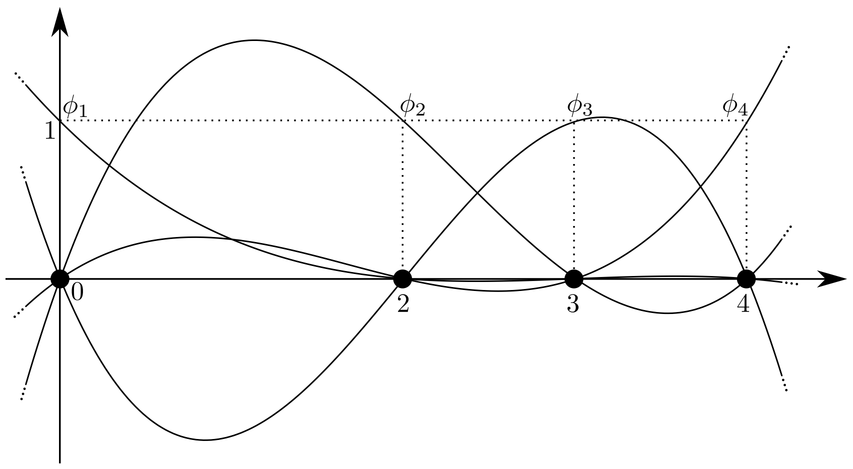


Figure 13.2 The Lagrange basis for $x_1 = 0, x_2 = 2, x_3 = 3, x_4 = 4$. Each ϕ_i satisfies $\phi_i(x_i) = 1$ and $\phi_i(x_j) = 0$ for all $i \neq j$.

Using this formula, finding the unique degree $k - 1$ polynomial fitting our (x_i, y_i) pairs is formulaic in the Lagrange basis:

$$f(x) \equiv \sum_i y_i \phi_i(x).$$

To check, if we substitute $x = x_j$ we find:

$$\begin{aligned} f(x_j) &= \sum_i y_i \phi_i(x_j) \\ &= y_j \text{ since } \phi_i(x_j) = 0 \text{ when } i \neq j. \end{aligned}$$

We have shown that in the Lagrange basis we can write a closed formula for $f(x)$ that does not require solving the Vandermonde system; in other words, we have replaced the Vandermonde matrix with the identity matrix. The drawback, however, is that each $\phi_i(x)$ takes $O(k)$ time to evaluate using the formula above, so computing $f(x)$ takes $O(k^2)$ time total; contrastingly, if we find the coefficients a_i from the Vandermonde system explicitly, the evaluation time for interpolation subsequently becomes $O(k)$.

Computation time aside, the Lagrange basis has an additional numerical drawback, in that the denominator is the product of a potentially large number of terms. If the x_i's are close together, then this product may include many terms close to zero; the end result is division by a small number when evaluating $\phi_i(x)$. As we have seen, this operation can create numerical instabilities that we wish to avoid.

A third basis for polynomials of degree $k - 1$ that attempts to compromise between the numerical quality of the monomials and the efficiency of the Lagrange basis is the *Newton* basis, defined as

$$\psi_i(x) = \prod_{j=1}^{i-1} (x - x_j).$$

This product has no terms when $i = 1$, so we define $\psi_1(x) \equiv 1$. Then, for all indices i, the function $\psi_i(x)$ is a degree $i - 1$ polynomial.

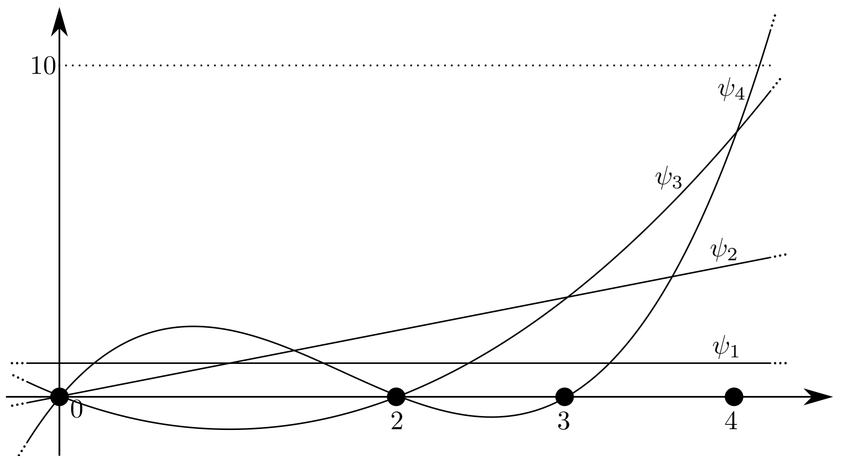


Figure 13.3 The Newton basis for $x_1 = 0, x_2 = 2, x_3 = 3, x_4 = 4$. Each ψ_i satisfies $\psi_i(x_j) = 0$ when $j < i$.

Example 13.2 (Newton basis)**.** Continuing from Example 13.1, again suppose $x_1 = 0$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$. The corresponding Newton basis is:

$$\begin{aligned}
\psi_1(x) &= 1\\
\psi_2(x) &= x\\
\psi_3(x) &= x(x-2) = x^2 - 2x\\
\psi_4(x) &= x(x-2)(x-3) = x^3 - 5x^2 + 6x.
\end{aligned}$$

This basis is illustrated in Figure 13.3.

By definition of ψ_i, $\psi_i(x_\ell) = 0$ for all $\ell < i$. If we wish to write $f(x) = \sum_i c_i\psi_i(x)$ and write out this observation more explicitly, we find:

$$\begin{aligned}
f(x_1) &= c_1\psi_1(x_1)\\
f(x_2) &= c_1\psi_1(x_2) + c_2\psi_2(x_2)\\
f(x_3) &= c_1\psi_1(x_3) + c_2\psi_2(x_3) + c_3\psi_3(x_3)\\
\vdots &\quad \vdots
\end{aligned}$$

These expressions provide the following lower-triangular system for $\vec{c}$:

$$\begin{pmatrix}
\psi_1(x_1) & 0 & 0 & \cdots & 0\\
\psi_1(x_2) & \psi_2(x_2) & 0 & \cdots & 0\\
\psi_1(x_3) & \psi_2(x_3) & \psi_3(x_3) & \cdots & 0\\
\vdots & \vdots & \vdots & \ldots & \vdots\\
\psi_1(x_k) & \psi_2(x_k) & \psi_3(x_k) & \cdots & \psi_k(x_k)
\end{pmatrix}
\begin{pmatrix} c_1\\ c_2\\ \vdots\\ c_k \end{pmatrix}
=
\begin{pmatrix} y_1\\ y_2\\ \vdots\\ y_k \end{pmatrix}.$$

This system can be solved in $O(k^2)$ time using forward-substitution, rather than the $O(k^3)$ time needed to solve the Vandermonde system using Gaussian elimination.[*] Evaluation time

[*]For completeness, we should mention that $O(k^2)$ Vandermonde solvers can be formulated; see [62] for discussion of these specialized techniques.

is similar to that of the Lagrange basis, but since there is no denominator, numerical issues are less likely to appear.

We now have three strategies of interpolating k data points using a degree $k-1$ polynomial by writing it in the monomial, Lagrange, and Newton bases. All three represent different compromises between numerical quality and speed, but the resulting interpolated function $f(x)$ is the *same* in each case. More explicitly, there is exactly one polynomial of degree $k-1$ going through a set of k points, so since all our interpolants are degree $k-1$ they must have the same output.

13.1.2 Alternative Bases

Although polynomial functions are particularly amenable to mathematical analysis, there is no fundamental reason why an interpolation basis cannot consist of different types of functions. For example, a crowning result of Fourier analysis implies that many functions are well-approximated by linear combinations of trigonometric functions $\cos(kx)$ and $\sin(kx)$ for $k \in \mathbb{N}$. A construction like the Vandermonde matrix still applies in this case, and the fast Fourier transform algorithm (which merits a larger discussion) solves the resulting linear system with remarkable efficiency.

A smaller extension of the development in §13.1.1 is to *rational* functions of the form:

$$f(x) \equiv \frac{p_0 + p_1 x + p_2 x^2 + \cdots + p_m x^m}{q_0 + q_1 x + q_2 x^2 + \cdots + q_n x^n}.$$

If we are given k pairs (x_i, y_i), then we will need $m + n + 1 = k$ for this function to be well-defined. One degree of freedom must be fixed to account for the fact that the same rational function can be expressed multiple ways by simultaneously scaling the numerator and the denominator.

Rational functions can have asymptotes and other features not achievable using only polynomials, so they can be desirable interpolants for functions that change quickly or have poles. Once m and n are fixed, the coefficients p_i and q_i still can be found using linear techniques by multiplying both sides by the denominator:

$$y_i(q_0 + q_1 x_i + q_2 x_i^2 + \cdots + q_n x_i^n) = p_0 + p_1 x_i + p_2 x_i^2 + \cdots + p_m x_i^m.$$

For interpolation, the unknowns in this expression are the p's and q's.

The flexibility of rational functions, however, can cause some issues. For instance, consider the following example:

Example 13.3 (Failure of rational interpolation, [117] §2.2)**.** Suppose we wish to find a rational function $f(x)$ interpolating the following data points: $(0,1)$, $(1,2)$, $(2,2)$. If we choose $m = n = 1$, then the linear system for finding the unknown coefficients is:

$$\begin{aligned} q_0 &= p_0 \\ 2(q_0 + q_1) &= p_0 + p_1 \\ 2(q_0 + 2q_1) &= p_0 + 2p_1. \end{aligned}$$

One nontrivial solution to this system is:

$$\begin{aligned} p_0 &= 0 & \qquad q_0 &= 0 \\ p_1 &= 2 & \qquad q_1 &= 1. \end{aligned}$$

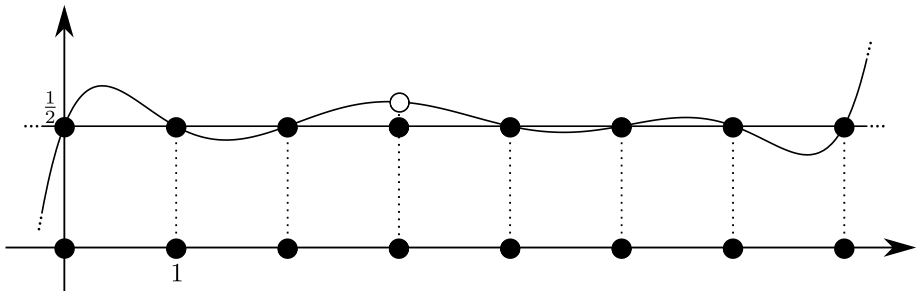


Figure 13.4 Interpolating eight samples of the function $f(x) \equiv 1/2$ using a seventh-degree polynomial yields a straight line, but perturbing a single data point at $x = 3$ creates an interpolant that oscillates far away from the infinitesimal vertical displacement.

This implies the following form for $f(x)$:

$$f(x) = \frac{2x}{x}.$$

This function has a degeneracy at $x = 0$, and canceling the x in the numerator and denominator does not yield $f(0) = 1$ as we might desire.

This example illustrates a larger phenomenon. The linear system for finding the p's and q's can run into issues when the resulting denominator $\sum_\ell p_\ell x^\ell$ has a root at any of the fixed x_i's. It can be shown that when this is the case, no rational function exists with the fixed choice of m and n interpolating the given values. A typical partial resolution in this case is presented in [117], which suggests incrementing m and n alternatively until a nontrivial solution exists. From a practical standpoint, however, the specialized nature of these methods indicates that alternative interpolation strategies may be preferable when the basic rational methods fail.

13.1.3 Piecewise Interpolation

So far, we have constructed interpolation bases out of elementary functions defined on all of $\mathbb{R}$. When the number k of data points becomes high, however, many degeneracies become apparent. For example, Figure 13.4 illustrates how polynomial interpolation is *nonlocal*, meaning that changing any single value y_i in the input data can change the behavior of f for all x, even those that are far away from x_i. This property is undesirable for most applications: We usually expect only the input data near a given x to affect the value of the interpolated function $f(x)$, especially when there is a large cloud of input points. While the Weierstrass Approximation Theorem from real analysis guarantees that any smooth function $f(x)$ on an interval $x \in [a, b]$ can be approximated arbitrarily well using polynomials, achieving a quality interpolation in practice requires choosing many carefully placed sample points.

As an alternative to global interpolation bases, when we design a set of basis functions $\phi_1, \ldots, \phi_k$, a desirable property we have not yet considered is *compact support*:

Definition 13.1 (Compact support)**.** A function $g(\vec{x})$ has *compact support* if there exists $C \in \mathbb{R}$ such that $g(\vec{x}) = 0$ for any $\vec{x}$ with $\|\vec{x}\|_2 > C$.

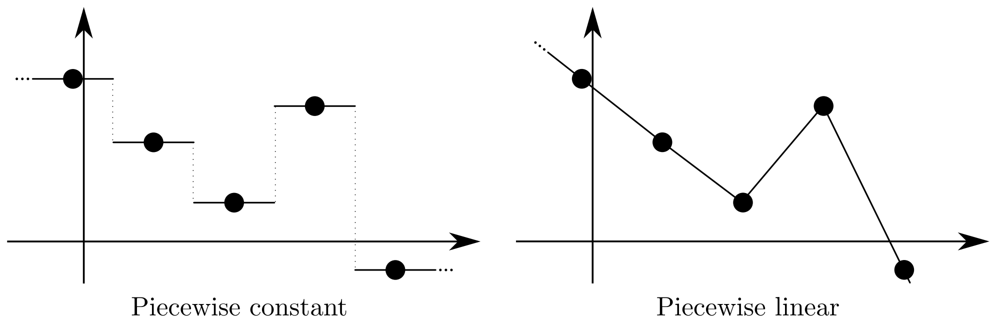


Figure 13.5 Two piecewise interpolation strategies.

That is, compactly supported functions only have a finite range of points in which they can take nonzero values.

Piecewise formulas provide one technique for constructing interpolatory bases with compact support. Most prominently, methods in computer graphics and many other fields make use of *piecewise polynomials*, which are defined by breaking $\mathbb{R}$ into a set of intervals and writing a different polynomial in each interval. To do so, we will order the data points so that $x_1 < x_2 < \cdots < x_k$. Then, two examples of piecewise interpolants are the following, illustrated in Figure 13.5:

- Piecewise constant interpolation: For a given $x \in \mathbb{R}$, find the data point x_i minimizing $|x - x_i|$ and define $f(x) = y_i$.
- Piecewise linear interpolation: If $x < x_1$ take $f(x) = y_1$, and if $x > x_k$ take $f(x) = y_k$. Otherwise, find the interval with $x \in [x_i, x_{i+1}]$ and define

$$f(x) = y_{i+1} \cdot \frac{x - x_i}{x_{i+1} - x_i} + y_i \cdot \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right).$$

Notice our pattern so far: Piecewise constant polynomials are discontinuous, while piecewise linear functions are continuous. Piecewise quadratics can be C^1, piecewise cubics can be C^2, and so on. This increased continuity and differentiability occurs even though each y_i has local support; this theory is worked out in detail in constructing "splines," or curves interpolating between points given function values and tangents.

Increased continuity, however, has its drawbacks. With each additional degree of differentiability, we put a stronger smoothness assumption on f. This assumption can be unrealistic: Many physical phenomena truly are noisy or discontinuous, and increased smoothness can negatively affect interpolatory results. One domain in which this effect is particularly clear is when interpolation is used in conjunction with physical simulation algorithms. Simulating turbulent fluid flows with excessively smooth functions inadvertently can remove discontinuous phenomena like shock waves.

These issues aside, piecewise polynomials still can be written as linear combinations of basis functions. For instance, the following functions serve as a basis for the piecewise constant functions:

$$\phi_i(x) = \begin{cases} 1 & \text{when } \frac{x_{i-1}+x_i}{2} \leq x < \frac{x_i+x_{i+1}}{2} \\ 0 & \text{otherwise.} \end{cases}$$

This basis puts the constant 1 near x_i and 0 elsewhere; the piecewise constant interpolation of a set of points (x_i, y_i) is written as $f(x) = \sum_i y_i \phi_i(x)$. Similarly, the so-called "hat" basis

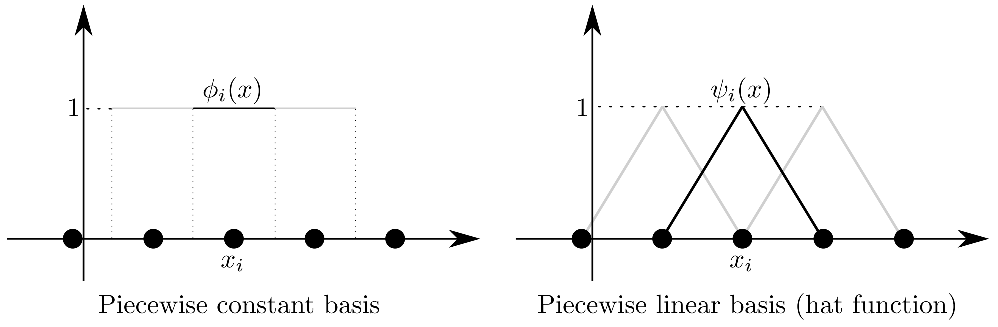


Figure 13.6 Basis functions corresponding to the piecewise interpolation strategies in Figure 13.5.

spans the set of piecewise linear functions with sharp edges at the data points x_i:

$$\psi_i(x) = \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & \text{when } x_{i-1} < x \leq x_i \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & \text{when } x_i < x \leq x_{i+1} \\ 0 & \text{otherwise.} \end{cases}$$

Once again, by construction, the piecewise linear interpolation of the given data points is $f(x) = \sum_i y_i \psi_i(x)$. Examples of both bases are shown in Figure 13.6.

13.2 MULTIVARIABLE INTERPOLATION

It is possible to extend the strategies above to the case of interpolating a function given data points $(\vec{x}_i, y_i)$ where $\vec{x}_i \in \mathbb{R}^n$ now can be multidimensional. Interpolation algorithms in this general case are challenging to formulate, however, because it is less obvious how to partition $\mathbb{R}^n$ into a small number of regions around the source points $\vec{x}_i$.

13.2.1 Nearest-Neighbor Interpolation

Given the complication of interpolation on $\mathbb{R}^n$, a common pattern is to interpolate using many *low-order* functions rather than fewer smooth functions, that is, to favor simplistic and efficient interpolants over ones that output C^∞ functions. For example, if all we are given is a set of pairs $(\vec{x}_i, y_i)$, then one piecewise constant strategy for interpolation is to use *nearest-neighbor interpolation*. In this case, $f(\vec{x})$ takes the value y_i corresponding to $\vec{x}_i$ minimizing $\|\vec{x} - \vec{x}_i\|_2$. Simple implementations iterate over all i to find the closest $\vec{x}_i$ to $\vec{x}$, and data structures like k-d trees can find nearest neighbors more quickly.

Just as piecewise constant interpolants on $\mathbb{R}$ take constant values on intervals about the data points x_i, nearest-neighbor interpolation yields a function that is piecewise-constant on *Voronoi cells*:

Definition 13.2 (Voronoi cell)**.** Given a set of points $S = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k\} \subseteq \mathbb{R}^n$, the *Voronoi cell* corresponding to a specific $\vec{x}_i \in S$ is the set $V_i \equiv \{\vec{x} : \|\vec{x} - \vec{x}_i\|_2 < \|\vec{x} - \vec{x}_j\|_2 \text{ for all } j \neq i\}$. That is, V_i is the set of points closer to $\vec{x}_i$ than to any other $\vec{x}_j$ in S.

Figure 13.7 shows an example of the Voronoi cells about a set of points in $\mathbb{R}^2$. These cells have many favorable properties; for example, they are convex polygons and are localized

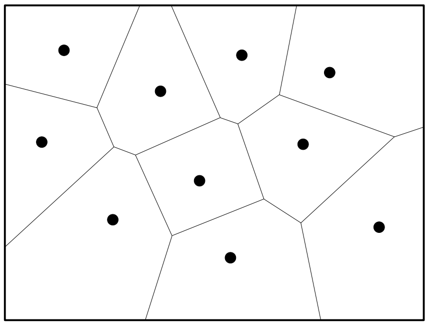


Figure 13.7 Voronoi cells associated with ten points in a rectangle.

about each $\vec{x}_i$. The adjacency of Voronoi cells is a well-studied problem in computational geometry leading to the construction of the celebrated Delaunay triangulation [33].

In many cases, however, it is desirable for the interpolant $f(\vec{x})$ to be continuous or differentiable. There are many options for continuous interpolation in $\mathbb{R}^n$, each with its own advantages and disadvantages. If we wish to extend the nearest-neighbor formula, we could compute multiple nearest neighbors $\vec{x}_1, \ldots, \vec{x}_k$ of $\vec{x}$ and interpolate $f(\vec{x})$ by averaging the corresponding $y_1, \ldots, y_k$ with distance-based weights; Exercise 13.4 explores one such weighting. Certain "k-nearest neighbor" data structures also can accelerate queries searching for multiple points in a dataset closest to a given $\vec{x}$.

13.2.2 Barycentric Interpolation

Another continuous multi-dimensional interpolant appearing frequently in the computer graphics literature is *barycentric* interpolation. Suppose we have exactly $n+1$ sample points $(\vec{x}_1, y_1), \ldots, (\vec{x}_{n+1}, y_{n+1})$, where $\vec{x}_i \in \mathbb{R}^n$, and we wish to interpolate the y_i's to all of $\mathbb{R}^n$; on the plane $\mathbb{R}^2$, we would be given three values associated with the vertices of a triangle. In the absence of degeneracies (e.g., three of the $\vec{x}_i$'s coinciding on the same line), any $\vec{x} \in \mathbb{R}^n$ can be written uniquely as a linear combination $\vec{x} = \sum_{i=1}^{n+1} a_i \vec{x}_i$ where $\sum_i a_i = 1$. This formula expresses $\vec{x}$ as a weighted average of the $\vec{x}_i$'s with weights a_i. For fixed $\vec{x}_1, \ldots, \vec{x}_{n+1}$, the weights a_i can be thought of as components of a function $\vec{a}(\vec{x})$ taking points $\vec{x}$ to their corresponding coefficients. Barycentric interpolation then defines $f(\vec{x}) \equiv \sum_i a_i(\vec{x}) y_i$.

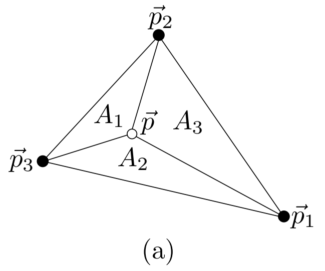
On the plane, barycentric interpolation has a straightforward geometric interpretation involving triangle areas, illustrated in Figure 13.8(a). Regardless of dimension, however, the barycentric interpolant $f(\vec{x})$ is *affine*, meaning it can be written $f(\vec{x}) = c + \vec{d} \cdot x$ for some $c \in \mathbb{R}$ and $\vec{d} \in \mathbb{R}^n$. Counting degrees of freedom, the $n+1$ sample points are accounted for via n unknowns in $\vec{d}$ and one unknown in c.

The system of equations to find $\vec{a}(\vec{x})$ corresponding to some $\vec{x} \in \mathbb{R}^n$ is:

$$\sum_i a_i \vec{x}_i = \vec{x} \qquad\qquad \sum_i a_i = 1$$

This system usually is invertible when there are $n+1$ points $\vec{x}_i$. In the presence of additional $\vec{x}_i$'s, however, it becomes *underdetermined*. This implies that there are multiple ways of writing $\vec{x}$ as a weighted average of the $\vec{x}_i$'s, making room for additional design decisions during barycentric interpolation, encoded in the particular choice of $\vec{a}(\vec{x})$.

One resolution of this non-uniqueness is to add more linear or nonlinear constraints on the weights $\vec{a}$. These yield different *generalized barycentric coordinates*. Typical constraints



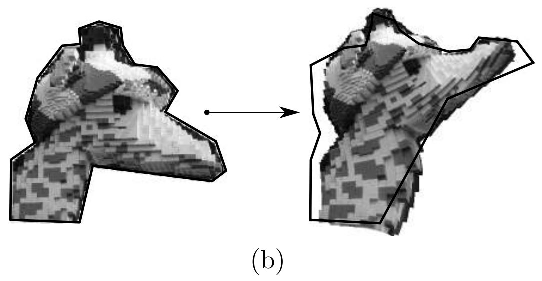


Figure 13.8 (a) The barycentric coordinates of $\vec{p} \in \mathbb{R}^2$ relative to the points $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$, respectively, are $({}^{A_1}/_{A}, {}^{A_2}/_{A}, {}^{A_3}/_{A})$, where $A \equiv A_1 + A_2 + A_3$ and A_i is the area of triangle i; (b) the barycentric deformation method [129] uses a generalized version of barycentric coordinates to deform planar shapes according to motions of a polygon with more than three vertices.

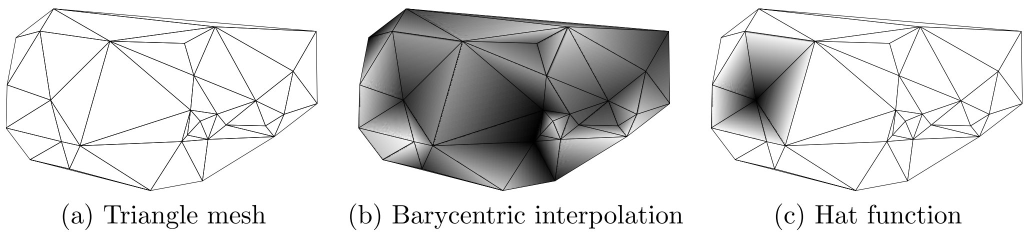


Figure 13.9 (a) A collection of points on $\mathbb{R}^2$ can be triangulated into a triangle mesh; (b) using this mesh, a per-point function can be interpolated to the interior using per-triangle barycentric interpolation; (c) a single "hat" basis function takes value one on a single vertex and is interpolated using barycentric coordinates to the remainder of the domain.

on $\vec{a}$ ask that it is smooth as a function of $\vec{x}$ on $\mathbb{R}^n$ and nonnegative on the interior of the polygon or polyhedron bordered by the $\vec{x}_i$'s. Figure 13.8(b) shows an example of image deformation using a recent generalized barycentric coordinates algorithm; the particular method shown makes use of complex-valued coordinates to take advantage of geometric properties of the complex plane [129].

Another way to carry out barycentric interpolation with more than $n + 1$ data points employs *piecewise* affine functions for interpolation; we will restrict our discussion to $\vec{x}_i \in \mathbb{R}^2$ for simplicity, although extensions to higher dimensions are possible. Suppose we are given not only a set of points $\vec{x}_i \in \mathbb{R}^2$ but also a triangulation linking those points together, as in Figure 13.9(a). If the triangulation is not known *a priori*, it can be computed using well-known geometric techniques [33]. Then, we can interpolate values from the vertices of each triangle to its interior using barycentric interpolation.

Example 13.4 (Shading). A typical representation of three-dimensional shapes in computer graphics is a set of triangles linked into a mesh. In the *per-vertex* shading model, one color is computed for each vertex on the mesh using lighting of the scene, material properties, and so on. Then, to render the shape on-screen, those per-vertex colors are interpolated using barycentric interpolation to the interiors of the triangles. Similar strategies are used for texturing and other common tasks. Figure 13.9(b) shows an example of this technique.

As an aside, one pertinent issue specific to computer graphics is the interplay between perspective transformations and interpolation. Barycentric interpolation of color along a triangulated 3D surface and then projection of that color onto the image plane is not the same as projecting triangles to the image plane and subsequently interpolating color along the projected two-dimensional triangles. Algorithms in this domain must use *perspective-corrected* interpolation strategies to account for this discrepancy during the rendering process.

Interpolation using a triangulation parallels the use of a piecewise-linear hat basis for one-dimensional functions, introduced in §13.1.3. Now, we can think of $f(\vec{x})$ as a linear combination $\sum_i y_i \phi_i(\vec{x})$, where each $\phi_i(\vec{x})$ is the piecewise affine function obtained by putting a 1 on $\vec{x}_i$ and 0 everywhere else, as in Figure 13.9(c).

Given a set of points in $\mathbb{R}^2$, the problem of triangulation is far from trivial, and analogous constructions in higher dimensions can scale poorly. When $n > 3$, methods that do not require explicitly partitioning the domain usually are preferable.

13.2.3 Grid-Based Interpolation

Rather than using triangles, an alternative decomposition of the domain of f occurs when the points $\vec{x}_i$ occur on a regular grid. The following examples illustrate situations when this is the case:

Example 13.5 (Image processing)**.** A typical digital photograph is represented as an $m \times n$ grid of red, green, and blue color intensities. We can think of these values as living on the lattice $\mathbb{Z} \times \mathbb{Z} \subset \mathbb{R} \times \mathbb{R}$. Suppose we wish to rotate the image by an angle that is not a multiple of 90°. Then, we must look up color values at potentially non-integer positions, requiring the interpolation of the image to $\mathbb{R} \times \mathbb{R}$.

Example 13.6 (Medical imaging)**.** The output of a magnetic resonance imaging (MRI) device is an $m \times n \times p$ grid of values representing the density of tissue at different points; a theoretical model for this data is as a function $f : \mathbb{R}^3 \to \mathbb{R}$. We can extract the outer surface of a particular organ by finding the level set $\{\vec{x} : f(\vec{x}) = c\}$ for some c. Finding this level set requires us to extend f to the entire voxel grid to find exactly where it crosses c.

Grid-based interpolation applies the one-dimensional formulae from §13.1.3 one dimension at a time. For example, *bilinear* interpolation in $\mathbb{R}^2$ applies linear interpolation in x_1 and then x_2 (or vice versa):

Example 13.7 (Bilinear interpolation)**.** Suppose f takes on the following values:

$$f(0,0) = 1 \qquad f(0,1) = -3 \qquad f(1,0) = 5 \qquad f(1,1) = -11$$

and that in between f is obtained by bilinear interpolation. To find $f(\frac{1}{4}, \frac{1}{2})$, we first interpolate in x_1 to find:

$$\begin{aligned} f\left(\frac{1}{4}, 0\right) &= \frac{3}{4} f(0,0) + \frac{1}{4} f(1,0) = 2 \\ f\left(\frac{1}{4}, 1\right) &= \frac{3}{4} f(0,1) + \frac{1}{4} f(1,1) = -5. \end{aligned}$$

Next, we interpolate in x_2:

$$f\left(\frac{1}{4},\frac{1}{2}\right)=\frac{1}{2}f\left(\frac{1}{4},0\right)+\frac{1}{2}f\left(\frac{1}{4},1\right)=-\frac{3}{2}.$$

We receive the same output interpolating first in x_2 and second in x_1.

Higher-order methods like bicubic and Lanczos interpolation use more polynomial terms but are slower to evaluate. For example, bicubic interpolation requires values from more grid points than just the four closest to $\vec{x}$ needed for bilinear interpolation. This additional expense can slow down image processing tools for which every lookup in memory incurs significant computation time.

13.3 THEORY OF INTERPOLATION

Our treatment of interpolation has been fairly heuristic. While relying on our intuition for what a "reasonable" interpolation for a set of function values for the most part is acceptable, subtle issues can arise with different interpolation methods that should be acknowledged.

13.3.1 Linear Algebra of Functions

We began our discussion by posing interpolation strategies using different bases for the set of functions $f:\mathbb{R}\to\mathbb{R}$. This analogy to vector spaces extends to a complete linear-algebraic theory of functions, and in many ways the field of *functional analysis* essentially extends the geometry of $\mathbb{R}^n$ to sets of functions. Here, we will discuss functions of one variable, although many aspects of the extension to more general functions are easy to carry out.

Just as we can define notions of span and linear combination for functions, for fixed $a,b\in\mathbb{R}$ we can define an *inner product* of functions $f(x)$ and $g(x)$ as follows:

$$\langle f,g\rangle\equiv\int_a^b f(x)g(x)\,dx.$$

We then can define the norm of a function $f(x)$ to be $\|f\|_2\equiv\sqrt{\langle f,f\rangle}$. These constructions parallel the corresponding constructions on $\mathbb{R}^n$; both the dot product $\vec{x}\cdot\vec{y}$ and the inner product $\langle f,g\rangle$ are obtained by multiplying the "elements" of the two multiplicands and summing—or integrating.

Example 13.8 (Functional inner product)**.** Take $p_n(x)=x^n$ to be the n-th monomial. Then, for $a=0$ and $b=1$,

$$\langle p_n,p_m\rangle=\int_0^1 x^n\cdot x^m\,dx=\int_0^1 x^{n+m}\,dx=\frac{1}{n+m+1}.$$

This shows:

$$\begin{aligned}\left\langle\frac{p_n}{\|p_n\|},\frac{p_m}{\|p_m\|}\right\rangle&=\frac{\langle p_n,p_m\rangle}{\|p_n\|\|p_m\|}\\&=\frac{\sqrt{(2n+1)(2m+1)}}{n+m+1}.\end{aligned}$$

This value is approximately 1 when $n\approx m$ but $n\neq m$, substantiating our earlier claim illustrated in Figure 13.1 that the monomials "overlap" considerably on $[0,1]$.

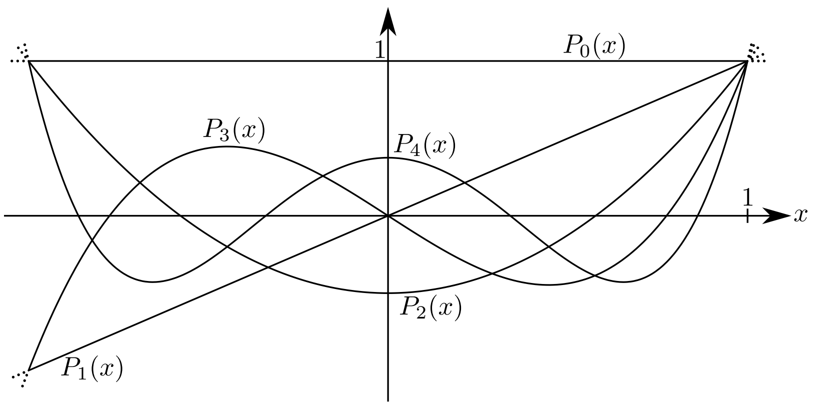


Figure 13.10 The first five Legendre polynomials, notated $P_0(x), \dots, P_4(x)$.

Given this inner product, we can apply the Gram-Schmidt algorithm to find an orthogonal basis for the set of polynomials, as we did in §5.4 to orthogonalize a set of vectors. If we take $a = -1$ and $b = 1$, applying Gram-Schmidt to the monomial basis yields the Legendre polynomials, plotted in Figure 13.10:

$$\begin{aligned}
P_0(x) &= 1\\
P_1(x) &= x\\
P_2(x) &= \frac{1}{2}(3x^2 - 1)\\
P_3(x) &= \frac{1}{2}(5x^3 - 3x)\\
P_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3)\\
&\vdots \quad \vdots
\end{aligned}$$

These polynomials have many useful properties thanks to their orthogonality. For example, suppose we wish to approximate $f(x)$ with a sum $\sum_i a_i P_i(x)$. If we wish to minimize $\|f - \sum_i a_i P_i\|_2$ in the functional norm, this is a *least-squares* problem! By orthogonality of the Legendre basis for $\mathbb{R}[x]$, our formula from Chapter 5 for projection onto an orthogonal basis shows:

$$a_i = \frac{\langle f, P_i \rangle}{\langle P_i, P_i \rangle}.$$

Thus, approximating f using polynomials can be accomplished by integrating f against the members of the Legendre basis. In the next chapter, we will learn how this integral can be carried out numerically.

Given a positive function $w(x)$, we can define a more general inner product $\langle \cdot, \cdot \rangle_w$ as

$$\langle f, g \rangle_w \equiv \int_a^b w(x) f(x) g(x)\, dx.$$

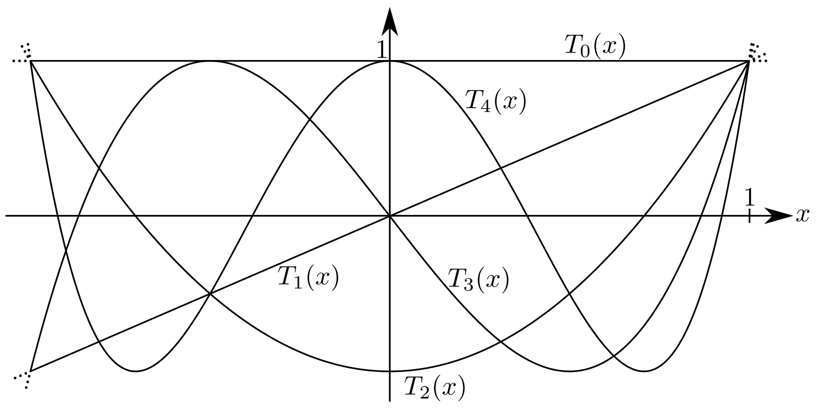


Figure 13.11 The first five Chebyshev polynomials, notated $T_0(x), \ldots, T_4(x)$.

If we take $w(x) = \frac{1}{\sqrt{1-x^2}}$ with $a = -1$ and $b = 1$, then Gram-Schmidt on the monomials yields the *Chebyshev* polynomials, shown in Figure 13.11:

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
\vdots \quad & \quad \vdots
\end{aligned}$$

A surprising identity holds for these polynomials:

$$T_k(x) = \cos(k \arccos(x)).$$

This formula can be checked by explicitly verifying it for T_0 and T_1, and then inductively applying the observation:

$$\begin{aligned}
T_{k+1}(x) &= \cos((k+1)\arccos(x)) \\
&= 2x\cos(k\arccos(x)) - \cos((k-1)\arccos(x)) \text{ by the identity} \\
&\qquad\qquad \cos((k+1)\theta) = 2\cos(k\theta)\cos(\theta) - \cos((k-1)\theta) \\
&= 2xT_k(x) - T_{k-1}(x).
\end{aligned}$$

This three-term recurrence formula also gives a way to generate explicit expressions for the Chebyshev polynomials in the monomial basis.

Thanks to this trigonometric characterization of the Chebyshev polynomials, the minima and maxima of T_k oscillate between $+1$ and -1. Furthermore, these extrema are located at $x = \cos(i\pi/k)$ (the *Chebyshev points*) for i from 0 to k. This even distribution of extrema avoids oscillatory phenomena like that shown in Figure 13.4 when using a finite number of polynomial terms to approximate a function. More technical treatments of polynomial interpolation recommend placing samples x_i for interpolation near Chebyshev points to obtain smooth output.

13.3.2 Approximation via Piecewise Polynomials

Suppose we wish to approximate a function $f(x)$ with a polynomial of degree n on an interval $[a,b]$. Define Δx to be the spacing $b-a$. One measure of the error of an approximation is as a function of Δx. If we approximate f with piecewise polynomials, this type of analysis tells us how far apart we should space the sample points to achieve a desired level of approximation.

Suppose we approximate $f(x)$ with a constant $c = f(\frac{a+b}{2})$, as in piecewise constant interpolation. If we assume $|f'(x)| < M$ for all $x \in [a,b]$, we have:

$$\begin{aligned}\max_{x\in[a,b]} |f(x)-c| &\leq \Delta x \max_{x\in[a,b]} M \text{ by the mean value theorem}\\ &\leq M\Delta x.\end{aligned}$$

Thus, we expect $O(\Delta x)$ error when using piecewise constant interpolation.

Suppose instead we approximate f using piecewise linear interpolation, that is, by taking

$$\tilde{f}(x) = \frac{b-x}{b-a}f(a) + \frac{x-a}{b-a}f(b).$$

We can use the Taylor expansion about x to write expressions for $f(a)$ and $f(b)$:

$$\begin{aligned}f(a) &= f(x) + (a-x)f'(x) + \frac{1}{2}(a-x)^2 f''(x) + O(\Delta x^3)\\ f(b) &= f(x) + (b-x)f'(x) + \frac{1}{2}(b-x)^2 f''(x) + O(\Delta x^3).\end{aligned}$$

Substituting these expansions into the formula for $\tilde{f}(x)$ shows

$$\begin{aligned}\tilde{f}(x) &= f(x) + \frac{1}{2\Delta x}((x-a)(b-x)^2 + (b-x)(x-a)^2)f''(x) + O(\Delta x^3)\\ &= f(x) + \frac{1}{2}(x-a)(x-b)f''(x) + O(\Delta x^3) \text{ after simplification.}\end{aligned}$$

This expression shows that linear interpolation holds up to $O(\Delta x^2)$, assuming f'' is bounded. Furthermore, for all $x \in [a,b]$ we have the bound $|x-a||x-b| \leq {}^{\Delta x^2}/_4$, implying an error bound proportional to ${}^{\Delta x^2}/_8$ for the second term.

Generalizing this argument shows that approximation with a degree-n polynomial generates $O(\Delta x^{n+1})$ error. In particular, if $f(x)$ is sampled at $x_0, x_1, \ldots, x_n$ to generate a degree-n polynomial p_n, then assuming $x_0 < x_1 < \cdots < x_n$, the error of such an approximation can be bounded as

$$|f(x) - p_n(x)| \leq \frac{1}{(n+1)!}\left[\max_{x\in[x_0,x_n]} \prod_k |x - x_k|\right] \cdot \left[\max_{x\in[x_0,x_n]} |f^{(n+1)}(x)|\right],$$

for any $x \in [x_0, x_n]$.

13.4 EXERCISES

13.1 Write the degree-three polynomial interpolating between the data points $(-2,15)$, $(0,-1)$, $(1,0)$, and $(3,-2)$.
Hint: Your answer does not have to be written in the monomial basis.

13.2 Show that the interpolation from Example 13.7 yields the same result regardless of whether x_1 or x_2 is interpolated first.

13.3 ("Runge function") Consider the function

$$f(x) \equiv \frac{1}{1+25x^2}.$$

Suppose we approximate $f(x)$ using a degree-k polynomial $p_k(x)$ through $k+1$ points $x_0, \ldots, x_k$ with $x_i = {}^{2i}\!/_k - 1$.

(a) Plot $p_k(x)$ for a few samples of k. Does increasing k improve the quality of the approximation?

(b) Specialize the bound at the end of §13.3.2 to show

$$\max_{x\in[-1,1]} |f(x) - p_k(x)| \leq \frac{1}{(k+1)!}\left[\max_{x\in[-1,1]} \prod_i |x - x_i|\right] \cdot \left[\max_{x\in[-1,1]} |f^{(k+1)}(x)|\right].$$

Does this bound get tighter as k increases?

(c) Suggest a way to fix this problem assuming we cannot move the x_i's.

(d) Suggest an alternative way to fix this problem by moving the x_i's.

13.4 ("Inverse distance weighting") Suppose we are given a set of distinct points $\vec{x}_1, \ldots, \vec{x}_k \in \mathbb{R}^n$ with labels $y_1, \ldots, y_k \in \mathbb{R}$. Then, one interpolation strategy defines an interpolant $f(\vec{x})$ as follows [108]:

$$f(\vec{x}) \equiv \begin{cases} y_i & \text{if } \vec{x} = \vec{x}_i \text{ for some } i \\ \frac{\sum_i w_i(\vec{x}) y_i}{\sum_i w_i(\vec{x})} & \text{otherwise,} \end{cases}$$

where $w_i(\vec{x}) \equiv \|\vec{x} - \vec{x}_i\|_2^{-p}$ for some fixed $p \geq 1$.

(a) Argue that as $p \to \infty$, the interpolant $f(\vec{x})$ becomes piecewise constant on the Voronoi cells of the $\vec{x}_i$'s.

(b) Define the function

$$\phi(\vec{x}, y) \equiv \left(\sum_i \frac{(y - y_i)^2}{\|\vec{x} - \vec{x}_i\|_2^p}\right)^{1/p}.$$

Show that for fixed $\vec{x} \in \mathbb{R}^n \backslash \{\vec{x}_1, \ldots, \vec{x}_k\}$, the value $f(\vec{x})$ is the minimum of $\phi(\vec{x}, y)$ over all y.

(c) Evaluating the sum in this formula can be expensive when k is large. Propose a modification to the w_i's that avoids this issue; there are many possible techniques here.

13.5 ("Barycentric Lagrange interpolation," [12]) Suppose we are given k pairs $(x_1, y_1), \ldots, (x_k, y_k)$.

(a) Define $\ell(x) \equiv \prod_{j=1}^k (x - x_j)$. Show that the Lagrange basis satisfies

$$\phi_i(x) = \frac{w_i \ell(x)}{x - x_i},$$

for some weight w_i depending on $x_1, \ldots, x_n$. The value w_i is known as the *barycentric weight* of x_i.

(b) Suppose $f(x)$ is the degree $k-1$ polynomial through the given (x_i, y_i) pairs. Assuming you have precomputed the w_i's, use the result of the previous part to give a formula for Lagrange interpolation that takes $O(k)$ time to evaluate.

(c) Use the result of 13.5b to write a formula for the constant function $g(x) \equiv 1$.

(d) Combine the results of the previous two parts to provide a third formula for $f(x)$ that does not involve $\ell(x)$.
Hint: $^{f(x)}/_{1} = f(x)$.

13.6 ("Cubic Hermite interpolation") In computer graphics, a common approach to drawing curves is to use cubic interpolation. Typically, artists design curves by specifying their endpoints as well as the tangents to the curves at the endpoints.

(a) Suppose $P(t)$ is the cubic polynomial:

$$P(t) = at^3 + bt^2 + ct + d.$$

Write a set of linear conditions on a, b, c, and d such that $P(t)$ satisfies the following conditions for fixed values of h_0, h_1, h_2, and h_3:

$$\begin{array}{ll} P(0) = h_0 & P'(0) = h_2 \\ P(1) = h_1 & P'(1) = h_3. \end{array}$$

(b) Write the *cubic Hermite* basis for cubic polynomials $\{\phi_0(t), \phi_1(t), \phi_2(t), \phi_3(t)\}$ such that $P(t)$ satisfying the conditions from 13.6a can be written

$$P(t) = h_0\phi_0(t) + h_1\phi_1(t) + h_2\phi_2(t) + h_3\phi_3(t).$$

13.7 ("Cubic blossom") We continue to explore interpolation techniques suggested in the previous problem.

(a) Given $P(t) = at^3 + bt^2 + ct + d$, define a *cubic blossom function* $F(t_1, t_2, t_3)$ in terms of $\{a, b, c, d\}$ satisfying the following properties [102]:

Symmetric: $F(t_1, t_2, t_3) = F(t_i, t_j, t_k)$
for any permutation (i, j, k) of $\{1, 2, 3\}$

Affine: $F(\alpha u + (1-\alpha)v, t_2, t_3) = \alpha F(u, t_2, t_3) + (1-\alpha)F(v, t_2, t_3)$

Diagonal: $f(t) = F(t, t, t)$

(b) Now, define

$$\begin{array}{ll} p = F(0,0,0) & q = F(0,0,1) \\ r = F(0,1,1) & s = F(1,1,1). \end{array}$$

Write expressions for $f(0)$, $f(1)$, $f'(0)$, and $f'(1)$ in terms of p, q, r, and s.

(c) Write a basis $\{B_0(t), B_1(t), B_2(t), B_3(t)\}$ for cubic polynomials such that given a cubic blossom $F(t_1, t_2, t_3)$ of $f(t)$ we can write

$$f(t) = F(0,0,0)B_0(t) + F(0,0,1)B_1(t) + F(0,1,1)B_2(t) + F(1,1,1)B_3(t).$$

The functions $B_i(t)$ are known as the cubic Bernstein basis.

(d) Suppose $F_1(t_1, t_2, t_3)$ and $F_2(t_1, t_2, t_3)$ are the cubic blossoms of functions $f_1(t)$ and $f_2(t)$, respectively, and define $\vec{F}(t_1, t_2, t_3) \equiv (F_1(t_1, t_2, t_3), F_2(t_1, t_2, t_3))$. Consider the four points shown in Figure 13.12. By bisecting line segments and drawing new ones, show how to construct $\vec{F}(^1/_2, ^1/_2, ^1/_2)$.

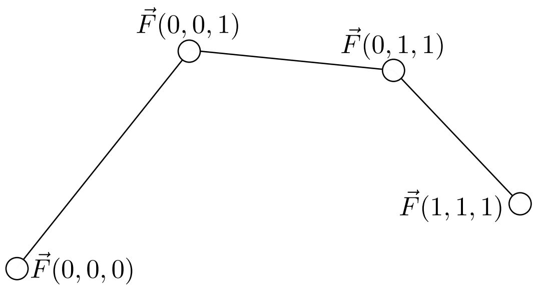


Figure 13.12 Diagram for Exercise 13.7d.

DH 13.8 Consider the polynomial $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1}$. Alternatively, we can write $p(x)$ in the Newton basis relative to $x_1, \ldots, x_n$ as

$$p(x) = c_1 + c_2 (x - x_1) + c_3 (x - x_1)(x - x_2) + \cdots + c_n \prod_{i=1}^{n-1} (x - x_i),$$

where $x_1, \ldots, x_n$ are fixed constants.

(a) Argue why we can write any $(n-1)$-st degree $p(x)$ in this form.

(b) Find explicit expressions for c_1, c_2, and c_3 in terms of x_1, x_2, and evaluations of $p(\cdot)$. Based on these expressions (and computing more terms if needed), propose a pattern for finding c_k.

(c) Use function evaluation to define the *zeroth divided difference* of p as $p[x_1] = p(x_1)$. Furthermore, define the *first divided difference* of p as

$$p[x_1, x_2] = \frac{p[x_1] - p[x_2]}{x_1 - x_2}.$$

Finally, define the *second divided difference* as

$$p[x_1, x_2, x_3] = \frac{p[x_1, x_2] - p[x_2, x_3]}{x_1 - x_3}.$$

Based on this pattern and the pattern you observed in the previous part, define $p[x_i, x_{i+1}, \ldots, x_j]$ and use it to provide a formula for the coefficients c_k.

(d) Suppose we add another point (x_{n+1}, y_{n+1}) and wish to recompute the Newton interpolant. How many Newton coefficients need to be recomputed? Why?

13.9 ("Horner's rule") Consider the polynomial $p(x) \equiv a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k$. For fixed $x_0 \in \mathbb{R}$, define $c_0, \ldots, c_k \in \mathbb{R}$ recursively as follows:

$$\begin{aligned} c_k &\equiv a_k \\ c_i &\equiv a_i + c_{i+1} x_0 \ \forall i < k. \end{aligned}$$

Show $c_0 = p(x_0)$, and compare the number of multiplication and addition operations needed to compute $p(x_0)$ using this method versus the formula in terms of the a_i's.

DH 13.10 Consider the L_2 distance between polynomials f, g on $[-1, 1]$, given by

$$||f-g||_2 \equiv \left[\int_{-1}^{1}(f(x)-g(x))^2\,dx\right]^{1/2},$$

which arises from the inner product $\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\,dx$. Let $\mathcal{P}_n$ be the vector space of polynomials of degree no more than n, endowed with this inner product. As we have discussed, polynomials $\{p_i\}_{i=1}^{m}$ are *orthogonal* with respect to this inner product if for all $i \neq j$, $\langle p_i, p_j\rangle = 0$; we can systematically obtain a set of orthonormal polynomials using the Gram-Schmidt process.

(a) Derive constant multiples of the first three Legendre polynomials via Gram-Schmidt orthogonalization on the monomials 1, x, and x^2.

(b) Suppose we wish to approximate a function f with a degree-n polynomial g. To do so, we can find the $g \in \mathcal{P}_n$ that is the best least-squares fit for f in the norm above. Write an optimization problem for finding g.

(c) Suppose we construct the Gram matrix G with entries $g_{ij} \equiv \langle p_i, p_j\rangle$ for a basis of polynomials $p_1, \ldots, p_n \in \mathcal{P}_n$. How is G involved in solving Exercise 13.10b? What is the structure of G when $p_1, \ldots, p_n$ are the first n Legendre polynomials?

DH 13.11 For a given n, the *Chebyshev points* are given by $x_k = \cos\left(\frac{k\pi}{n}\right)$, where $k \in \{0, \ldots, n\}$.

(a) Show that the Chebyshev points are the projections onto the x axis of n *evenly spaced* points on the upper half of the unit circle.

(b) Suppose that we *define* the Chebyshev polynomials using the expression $T_k(x) \equiv \cos(k \arccos(x))$. Starting from this expression, compute the first four Chebyshev polynomials in the monomial basis.

(c) Show that the Chebyshev polynomials you computed in the previous part are orthogonal with respect to the inner product $\langle f, g\rangle \equiv \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}}\,dx$.

(d) Show that the extrema of T_n are located at Chebyshev points x_k.

13.12 We can use interpolation strategies to formulate methods for root-finding in one or more variables.

(a) Show how to recover the parameters a, b, c of the *linear fractional transformation*

$$f(x) \equiv \frac{x+a}{bx+c}$$

going through the points (x_0, y_0), (x_1, y_1), and (x_2, y_2), either in closed form or by posing a 3×3 linear system of equations.

(b) Find x_4 such that $f(x_4) = 0$.

(c) Suppose we are given a function $f(x)$ and wish to find a root x^* with $f(x^*) = 0$. Suggest an algorithm for root-finding using the construction in Exercise 13.12b.