CHAPTER 4

INTERPOLATION AND APPROXIMATION

One of the oldest problems in mathematics—and, at the same time, one of the most applied—is the problem of constructing an approximation to a given function f from among simpler functions, typically (but not always) polynomials. A slight variation of this problem is that of constructing a smooth function from a discrete set of data points.

In this chapter we will study both of these problems and develop several methods for solving them. We start with a more general treatment of an idea we first saw in Chapter 2.

4.1 LAGRANGE INTERPOLATION

The basic interpolation problem can be posed in one of two ways:

1. Given a set of *nodes* $\{x_i, 0 \le i \le n\}$ and corresponding data values $\{y_i, 0 \le i \le n\}$, find the polynomial $p_n(x)$ of degree less than or equal to n, such that

$$p_n(x_i) = y_i, \quad 0 \le i \le n.$$

2. Given a set of *nodes* $\{x_i, 0 \le i \le n\}$ and a continuous function $f(x)$, find the polynomial $p_n(x)$ of degree less than or equal to n, such that

$$p_n(x_i) = f(x_i), \quad 0 \le i \le n.$$

Note that in the first case we are trying to fit a polynomial to the data, and in the second we are trying to approximate a given function with the interpolating polynomial.

While the two cases are in fact different, we can always consider the first one to be a special case of the second (by taking $y_i = f(x_i)$, for each i), so we will present most of the material here in terms of the second version of the problem.

It is actually very easy to prove that both versions of the problem have a unique solution. Moreover, the proof shows us how to construct the polynomial.

Theorem 4.1 (Polynomial Interpolation Existence and Uniqueness) *Let the nodes $x_i \in I$, $0 \le i \le n$ be given. So long as each of the x_i is distinct, i.e., $x_i = x_j$ if and only if $i = j$, then there exists a unique polynomial p_n, of degree less than or equal to n, which satisfies either of*

(4.1) $$p_n(x_i) = y_i, \quad 0 \le i \le n$$

for a given set of data values $\{y_i\}$, or

(4.2) $$p_n(x_i) = f(x_i), \quad 0 \le i \le n$$

for a given function $f \in C(I)$.

Proof: We begin by defining the family of functions[1]

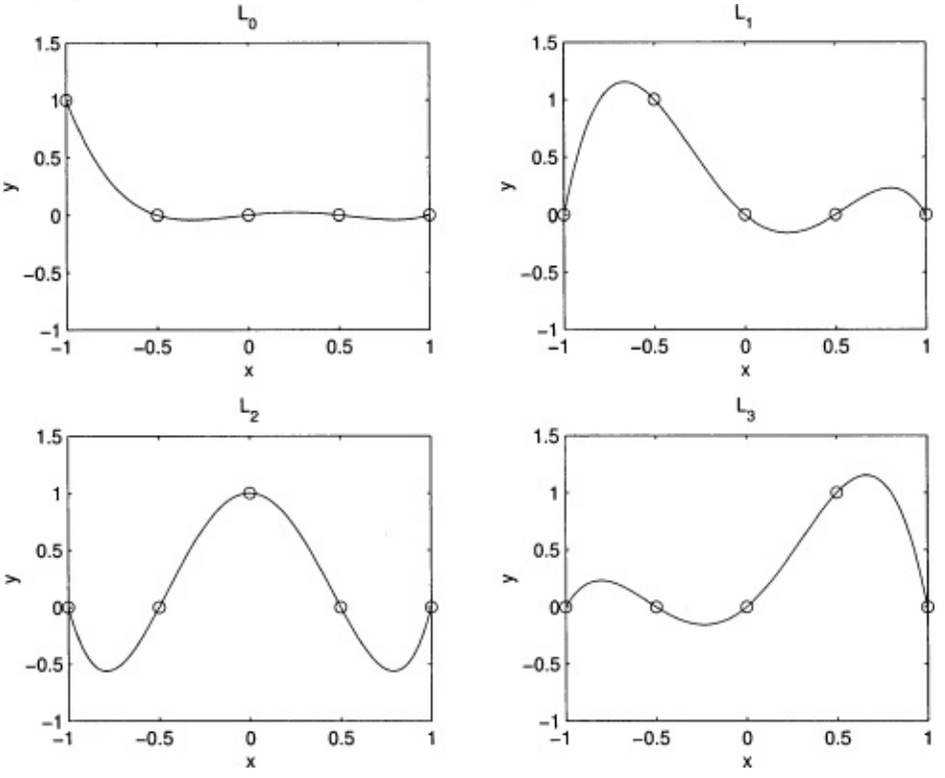
$$L_i^{(n)}(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k}$$

and note that they are polynomials of degree n and have the interesting property that

$$L_i^{(n)}(x_j) = \delta_{ij} = \begin{cases} 1, & i = j; \\ 0, & i \neq j. \end{cases} \tag{4.3}$$

(The symbol δ_{ij} implicitly defined in the above is called the *Kronecker*[2] *delta*.) Figure 4.1 shows examples of these polynomials for five equally spaced nodes on [−1, 1]. Based on (4.3), then, if we define the polynomial by

Figure 4.1 Lagrange polynomials for five equally spaced nodes on [−1, 1].



$$p_n(x) = \sum_{k=0}^{n} y_k L_k^{(n)}(x),$$

it follows that

$$p_n(x_i) = \sum_{k=0}^{n} y_k L_k^{(n)}(x_i) = y_i.$$

Thus, the interpolatory conditions are satisfied, and it remains only to prove the uniqueness of the polynomial. To this end, assume that there exists a second interpolatory polynomial; call it q, and define

$$r(x) = p_n(x) - q(x).$$

Since both p_n and q are polynomials of degree less than or equal to n, so is their difference. However, we must note that

$$r(x_i) = p_n(x_i) - q(x_i) = y_i - y_i = 0$$

for each of the $n + 1$ nodes. Thus, we have a polynomial of degree less than or equal to n that has $n + 1$ roots. The only such polynomial is the zero polynomial; that is,

$$r(x) \equiv 0 \Rightarrow p_n(x) \equiv q(x),$$

and thus p_n is unique. •

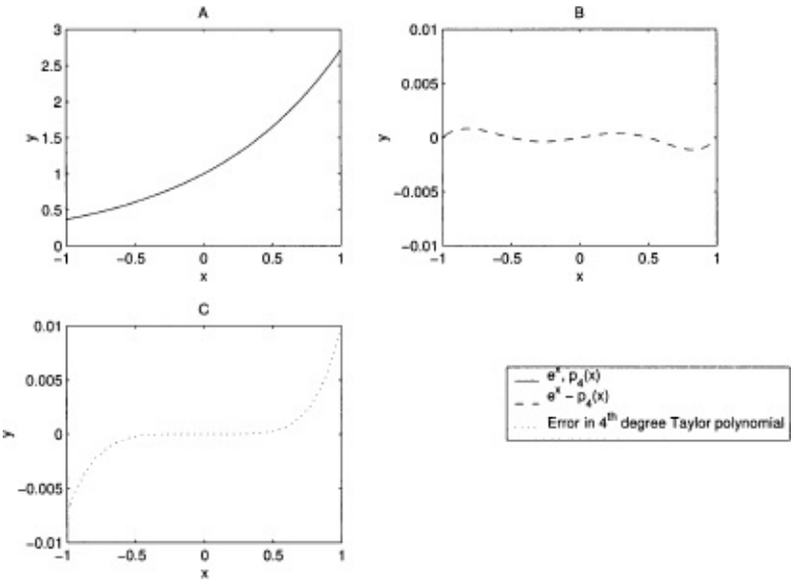
EXAMPLE 4.1

To illustrate this construction let $f(x) = e^x$ and consider the problem of constructing an approximation to this function on the interval $[-1, 1]$, using the nodes $\{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$. We have

$$\begin{aligned} p_4(x) &= \frac{(x+\frac{1}{2})(x-0)(x-\frac{1}{2})(x-1)}{(-1+\frac{1}{2})(-1-0)(-1-\frac{1}{2})(-1-1)}e^{-1} \\ &+ \frac{(x+1)(x-0)(x-\frac{1}{2})(x-1)}{(-\frac{1}{2}+1)(-\frac{1}{2}-0)(-\frac{1}{2}-\frac{1}{2})(-\frac{1}{2}-1)}e^{-\frac{1}{2}} \\ &+ \frac{(x+1)(x+\frac{1}{2})(x-\frac{1}{2})(x-1)}{(0+1)(0+\frac{1}{2})(0-\frac{1}{2})(0-1)}e^{0} \\ &+ \frac{(x+1)(x+\frac{1}{2})(x-0)(x-1)}{(\frac{1}{2}+1)(\frac{1}{2}+\frac{1}{2})(\frac{1}{2}-0)(\frac{1}{2}-1)}e^{\frac{1}{2}} \\ &+ \frac{(x+1)(x+\frac{1}{2})(x-0)(x-\frac{1}{2})}{(1+1)(1+\frac{1}{2})(1-0)(1-\frac{1}{2})}e^{1}, \end{aligned}$$

which simplifies to $p_4(x) = 1.0 + 0.997853749x + 0.499644939x^2 + 0.177347443x^3 + 0.043435696x^4$. Figure 4.2A is a the plot of f and $p_4(x)$. The approximation is sufficiently accurate that it is impossible to discern that there are two curves present, so it is probably better to look at Figure 4.2B, which plots the error $e^x - p_4(x)$. Compare this to the error obtained by doing a fourth-degree Taylor polynomial, centered at the origin, which is given in Figure 4.2C. Note that while the Taylor polynomial is *very* accurate near the middle of the interval, and much less accurate near the ends, the interpolating polynomial has less variation in its error, and a much smaller worst-case error over the same interval.

Figure 4.2 Interpolation to $f(x) = e^x$.

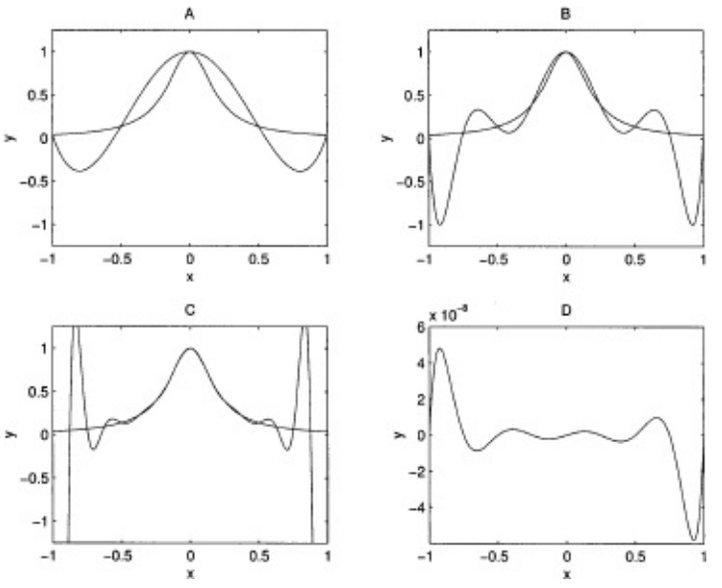


The construction presented in this section is called *Lagrange*[3] *interpolation*. Other approaches to the interpolation problem are studied in other sections; this one is the most basic and fundamental.

How good is interpolation at approximating a function? We will study this a little more carefully in §§4.3 and 4.12, but we can illustrate some of the issues by a few examples right now. We can already note that the fourth-degree approximation to the exponential function was very accurate, even when compared to the fourth-degree Taylor polynomial. What about some other functions?

Consider now the function $f(x) = (1 + 25x^2)^{-1}$. If we use a fourth-degree interpolating polynomial to approximate this function, the results are as shown in Figure 4.3A. These are not nearly as good as for the exponential function. If we increase the number of points (and therefore the degree of the interpolating polynomial), things improve on part of the interval, but not all of it. Figure 4.3B shows the plots for $n = 8$, and Figure 4.3C shows the $n = 16$ results. In contrast, Figure 4.3D plots the *error* for the 8-degree polynomial interpolation to the exponential function.[4])

Figure 4.3 Polynomial interpolation to $f(x) = (1 + 25x^2)^{-1}$ with $n = 4$(A), $n = 8$ (B), $n = 16$ (C), and the error in interpolation for $n = 8$ to $f(x) = e^x$(D).



Clearly, there are circumstances ($f(x) = e^x$) in which polynomial interpolation as approximation will work very well, and circumstances ($f(x) = (1 + 25x^2)^{-1}$) in which it will not. More study is therefore required.

The attentive reader will have noted that we have not discussed an algorithm for computing Lagrange interpolating polynomials. The reason for this is quite simple, actually: The Lagrange form of the interpolating polynomial is not well-suited for actual computations, and there is an alternate construction that is far superior to it. So we will defer all discussion of algorithms until Section 4.2.

Exercises:

1. Find the polynomial of degree 2 that interpolates at the data points $x_0 = 0$, $y_0 = 1$, $x_1 = 1$, $y_1 = 2$, and $x_2 = 4$, $y_2 = 2$. You should get $p_2(t) = -\frac{1}{4}t^2 + \frac{5}{4}t + 1$..

2. Find the polynomial of degree 2 that interpolates to $y = x^3$ at the nodes $x_0 = 0$, $x_1 = 1$, and $x_2 = 2$. Plot $y = x^3$ and the interpolating polynomial over the interval [0, 2].

3. Construct the quadratic polynomial that interpolates to $y = 1/x$ at the nodes $x_0 = 1/2$, $x_1 = 3/4$, and $x_2 = 1$. Plot both the interpolate and the function over the interval $[\frac{1}{4}, \frac{5}{4}]$.

4. Construct the quadratic polynomial that interpolates to $y = \sqrt{x}$ at the nodes $x_0 = 1/4$, $x_1 = 9/16$, and $x_2 = 1$. Plot both the interpolate and the function over the interval [0, 2].

5. For each function listed below, construct the Lagrange interpolating polynomial for the set of nodes specified. Plot the function, the interpolating polynomial, and the error, on the interval defined by the nodes. (You may want to plot the error separately, because of scale issues.)

(a) $f(x) = \ln x$, $x_i = 1, \frac{3}{2}, 2$;

(b) $f(x) = \sqrt{x}$, $x_i = 0, 1, 4$;

(c) $f(x) = \log_2 x$, $x_i = 1, 2, 4$;

(d) $f(x) = \sin \pi x$, $x_i = -1, 0, 1$.

6. Find the polynomial of degree 3 that interpolates $y = x^3$ at the nodes $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$. (Simplify your interpolating polynomial as much as possible.) *Hint:* This is easy if you think about the implications of the uniqueness of the interpolating polynomial.

7. Construct the Lagrange interpolating polynomial to the function $f(x) = x^2 + 2x$, using the nodes $x_0 = 0$, $x_1 = 1$, $x_2 = -2$. Repeat, using the nodes $x_0 = 2$, $x_1 = 1$, and $x_2 = -1$. (For both sets of nodes, simplify your interpolating polynomial as much as possible.) Comment on your results, especially in light of the uniqueness part of Theorem 4.1, and then write down the interpolating polynomial for interpolating to $f(x) = x^3 + 2x^2 + 3x + 1$ at the nodes $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $x_4 = 4$, and $x_5 = 5$. *Hint:* You should be able to do this last part without doing any computations.

8. Let f be a polynomial of degree $\leq n$, and let p_n be a polynomial interpolant to f, at the $n + 1$ distinct nodes $x_0, x_1, \ldots, x_n$. Prove that $p_n(x) = f(x)$ for all x, i.e., that interpolating to a polynomial will reproduce the polynomial, if you use enough nodes. *Hint:* Consider the uniqueness part of Theorem 4.1.

9. Let p be a polynomial of degree $\leq n$. Use the uniqueness of the interpolating polynomial to show that we can write

$$p(x) = \sum_{i=0}^{n} L_i^{(n)}(x) p(x_i)$$

for any distinct set of nodes $x_0, x_1, \ldots, x_n$. *Hint:* See the previous exercise.

10. Show that

$$\sum_{i=0}^{n} L_i^{(n)}(x) = 1$$

for any set of distinct nodes x_k, $0 \leq k \leq n$. *Hint*: This does not require *any* computation with the sum but, rather, a perceptive choice of polynomial p in the previous exercise.

4.2 NEWTON INTERPOLATION AND DIVIDED DIFFERENCES

The Lagrange form of the interpolating polynomial gives us a very tidy construction, but it does not lend itself well to actual computation. The reason is partly because whenever we decide to add a point to the set of nodes, we have to completely recompute all of the $L_i^{(n)}$ functions; we cannot (easily) write p_{n+1} in terms of p_n using the Lagrange construction.

An alternate form of the polynomial, known as the Newton form, avoids this problem, and allows us to easily write p_{n+1} in terms of p_n. The basic result is contained in the following

theorem.

Theorem 4.2 (Newton Interpolation Construction) *Let p_n be the polynomial that interpolates f at the nodes x_i, $i = 0, 1, 2, 3, \ldots, n$. Let p_{n+1} be the polynomial that interpolates f at the nodes x_i, $i = 0, 1, 2, 3, \ldots, n, n+1$. Then p_{n+1} is given by*

$$p_{n+1}(x) = p_n(x) + a_{n+1}w_n(x), \tag{4.4}$$

where

$$\begin{aligned} w_n(x) &= \prod_{i=0}^{n}(x - x_i), \\ a_{n+1} &= \frac{f(x_{n+1}) - p_n(x_{n+1})}{w_n(x_{n+1})}, \end{aligned} \tag{4.5}$$

and

$$p_0(x) \equiv a_0 = f(x_0).$$

Proof: Since we know that the interpolation polynomial is unique, all we have to do is show that p_{n+1}, as given in (4.4), satisfies the interpolation conditions. For $k \le n$, we have $w_n(x_k) = 0$, so

$$p_{n+1}(x_k) = p_n(x_k) + a_{n+1}w_n(x_k) = p_n(x_k) = f(x_k);$$

hence, p_{n+1} interpolates all but the last point. To check x_{n+1}, we directly compute:

$$\begin{aligned} p_{n+1}(x_{n+1}) &= p_n(x_{n+1}) + a_{n+1}w_n(x_{n+1}) \\ &= p_n(x_{n+1}) + f(x_{n+1}) - p_n(x_{n+1}) \\ &= f(x_{n+1}). \end{aligned}$$

Thus, p_{n+1} interpolates at all the nodes; moreover, it clearly is a polynomial of degree less than or equal to $n + 1$, so we are done. •

Corollary 4.1 *For $\{a_k\}$ and w_n as defined in Theorem 4.2, we have*

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1}) \\ &= \sum_{k=0}^{n} a_k w_{k-1}(x), \end{aligned}$$

where we have used $w_{-1}(x) \equiv 1$.

Proof: This is a straight-forward inductive argument; see Problem 5. •

We can use this Newton form of the interpolating polynomial to construct and evaluate interpolating polynomials in an efficient fashion that is reminiscent of Horner's rule for nested multiplication (§2.1). We have

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\cdots(x - x_{n-1}),$$

so that we can write

$$p_n(x) = a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + \cdots + (x - x_{n-1})a_n)\cdots)).$$

■ EXAMPLE 4.2

To consider an example, let us construct the quadratic polynomial that interpolates the sine

function on the interval $[0, \pi]$, using equally spaced nodes. We have $f(x) = \sin x$, $x_i = 0, \frac{1}{2}\pi, \pi$ and $y_i = 0, 1, 0$. Then

$$\begin{aligned} a_0 &= 0 \quad \Rightarrow \quad p_0(x) = 0 \\ a_1 = \frac{\sin\frac{1}{2}\pi - p_0(\frac{1}{2}\pi)}{(\frac{1}{2}\pi - 0)} &= \frac{2}{\pi} \quad \Rightarrow \quad p_1(x) = \frac{2}{\pi}x; \\ a_2 = \frac{\sin\pi - p_1(\pi)}{(\pi - 0)(\pi - \frac{1}{2}\pi)} &= -\frac{4}{\pi^2} \quad \Rightarrow \quad p_2(x) = \frac{2}{\pi}x - \frac{4}{\pi^2}x\left(x - \frac{1}{2}\pi\right). \end{aligned}$$

The reader should check that each interpolating polynomial does interpolate at the appropriate points.

The coefficients a_k are called *divided differences*, and we can construct an algorithm for computing them, based on the formula (4.5). It is also possible to slightly modify this algorithm to allow the user to *update* an existing set of divided difference coefficients by adding new points to the set of nodes. See Problem 6.

Algorithm 4.1 *Newton Interpolation (Construction) via Divided Difference Coefficients*

```
input n, x, y
a(0) = y(0)
for k=1 to n do
    w = 1
    p = 0
    for j=0 to k-1 do
        p = p + a(j)*w
        w = w*(x(k) - x(j))
    endfor
    a(k) = (y(k) - p)/w
endfor
```

Algorithm 4.2 *Newton Interpolation (Evaluation) via Divided Difference Coefficients*

```
    input n, xx, x, a
!
!    n = degree of polynomial
!    xx = point at which the polynomial is
!         to be evaluated
!    x = array of nodes
!    a = array of divided difference coefficients
!
    px = a(n)
!    px = polynomial value at xx
!
    for k = n-1, 0, -1
        xd = xx - x(k)
        px = a(k) + px*xd
    endfor
```

Programming Hint: We know, from the exercises in §4.1, that polynomial interpolation of sufficiently high degree to a polynomial will reproduce the original polynomial. Thus, one way to check that an interpolation code is working is to see if it can reproduce a polynomial of degree n when asked to interpolate it at $n + 1$ nodes.

An alternate way of arriving at the divided difference coefficients—and, historically, the original way it was done—is by means of a *divided difference table*. Since this is an easy means by which hand calculations can be performed, it is worth looking at here as well.

We begin by defining some terminology and notation. Given a set of nodes and corresponding function values, the function values will be referred to as the *zero-th divided differences* and, in this context, we will write them as $f_0(x_k)$. *The first divided differences* are then formed from the zero-th ones according to

$$f_1(x_k) = \frac{f_0(x_{k+1}) - f_0(x_k)}{x_{k+1} - x_k},$$

and then the *second divided differences* are given by

$$f_2(x_k) = \frac{f_1(x_{k+1}) - f_1(x_k)}{x_{k+2} - x_k},$$

and so on. The general formula is

$$f_j(x_k) = \frac{f_{j-1}(x_{k+1}) - f_{j-1}(x_k)}{x_{k+j} - x_k}.$$

Now using this terminology and notation,[5] we can build a set of tables that can readily be used to construct the interpolating polynomials.

EXAMPLE 4.3

Take $f(x) = \log_2 x$ as the function to be interpolated, using the nodes $x_0 = 1$, $x_1 = 2$, $x_2 = 4$.

We begin by arranging the nodes and corresponding functional data in two columns, as in Table 4.1. Note that we write the functional data using the divided difference notation.

Table 4.1 Initial table for divided differences.

k	x_k	$f_0(x_k)$
0	1	0
1	2	1
2	4	2

The third column of the table is formed by the first divided differences; thus,

$$1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{1 - 0}{2 - 1} = f_1(x_0)$$

and

$$\frac{1}{2} = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{2 - 1}{4 - 2} = f_1(x_1),$$

so we get Table 4.2.

Table 4.2 Table for divided differences after computation of first differences.

k	x_k	$f_0(x_k)$	$f_1(x_k)$
0	1	0	
			1
1	2	1	
			1/2
2	4	2	

The fourth (and, in this case, final) column is formed by the second divided difference:

$$-\frac{1}{6} = \frac{1/2 - 1}{4 - 1} = f_2(x_0);$$

see Table 4.3.

Table 4.3 Table for divided differences after computation of second differences.

k	x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$
0	1	0		
			1	
1	2	1		$-\frac{1}{6}$
			$\frac{1}{2}$	
2	4	2		

(If we had more data, subsequent columns would be formed in a similar fashion.) How do we use this result to construct the interpolating polynomial? The top diagonal row of table values gives us the divided difference coefficients based on the nodes as numbered in ascending order. Thus

$$p_2(x) = 0 + (1)(x - 1) - \frac{1}{6}(x - 1)(x - 2) = -\frac{1}{6}(x - 1)(x - 8).$$

(These are the same values that the pseudocode produces.) The bottom diagonal row gives the divided difference coefficients based on numbering the nodes in a reversed order. Thus we also have

$$p_2(x) = 2 + \frac{1}{2}(x - 4) - \frac{1}{6}(x - 4)(x - 2) = -\frac{1}{6}(x - 1)(x - 8).$$

Finally, to add a node to the data set and construct the new polynomial is easy, based on these tables. Let's add the new node $x_3 = \frac{1}{2}$ to our current example. We have, initially, the arrangement in Table 4.4. (Note that the nodes do not have to be arranged in ascending order—or, in fact, in any particular order.)

Table 4.4 Table for divided differences with new node added.

k	x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$
0	1	0		
			1	
1	2	1		$-\frac{1}{6}$
			$\frac{1}{2}$	
2	4	2		
3	$\frac{1}{2}$	−1		

Computing forward we get the (new) final result in Table 4.5, from which we conclude that the interpolating polynomial is

Table 4.5 Table for divided differences after computation of third differences.

k	x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$	$f_3(x_k)$
0	1	0			
			1		
1	2	1		$-\frac{1}{6}$	
			$\frac{1}{2}$		$\frac{1}{7}$
2	4	2		$-\frac{5}{21}$	
			$\frac{6}{7}$		
3	$\frac{1}{2}$	-1			

$$p_3(x) = 0 + (1)(x-1) - \frac{1}{6}(x-1)(x-2) + \frac{1}{7}(x-1)(x-2)(x-4).$$

The reader should confirm that this does indeed interpolate correctly at the given points, and therefore is the correct polynomial.

Programming Hint: An interpolating polynomial, if it is correctly constructed, has to reproduce the values at the nodes exactly, so this is an easy test to use in evaluating and debugging your code. If the values at the nodes are computed correctly, then the polynomial has to be correct; if the values at the nodes are not computed correctly, then there is something wrong, either with the construction of the polynomial or with the code that is evaluating the polynomial.

Exercises:

1. Construct the polynomial of degree 3 that interpolates to the data $x_0 = 1$, $y_0 = 1$, $x_1 = 2$, $y_1 = 1/2$, $x_2 = 4$, $y_2 = 1/4$, and $x_3 = 3$, $y_3 = 1/3$. You should get $p(t) = (50 - 35t + 10t^2 - t^3)/24$.

2. For each function listed below, use divided difference tables to construct the Newton interpolating polynomial for the set of nodes specified. Plot the function, the interpolating polynomial, and also the error, on the interval defined by the nodes.

(a) $f(x) = \sqrt{x}$, $x_i = 0, 1, 4$;

(b) $f(x) = \ln x$, $x_i = 1, \frac{3}{2}, 2$;

(c) $f(x) = \sin \pi x$, $x_i = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$;

(d) $f(x) = \log_2 x$, $x_i = 1, 2, 4$;

(e) $f(x) = \sin \pi x$, $x_i = -1, 0, 1$.

3. Let $f(x) = e^x$. Define $p_n(x)$ to be the Newton interpolating polynomial for $f(x)$, using $n + 1$ equally spaced nodes on the interval $[-1, 1]$. Thus we are taking higher and higher degree polynomial approximations to the exponential function. Write a program that computes $p_n(x)$ for $n = 2, 4, 8, 16, 32$, and which samples the error $f(x) - p_n(x)$ at 501 equally spaced points on $[-1, 1]$. Record the maximum error as found by the sampling, as a function of n, that is, define E_n as

$$E_n = \max_{0 \le k \le 500} |f(t_k) - p_n(t_k)|,$$

where $t_k = -1 + 2k/500$, and plot E_n versus n.

4. In §3.7 we used linear interpolation to construct an initial guess for Newton's method as a means of approximating $\sqrt{a}$. Construct the quadratic polynomial that interpolates the square root function at the nodes $x_0 = \frac{1}{4}$, $x_1 = \frac{4}{9}$, $x_2 = 1$. Plot the error between p_2 and $\sqrt{x}$ over the interval $[\frac{1}{4}, 1]$ and try to estimate the worst error. What impact will this have on the use of Newton's method for finding $\sqrt{a}$?

5. Prove Corollary 4.1.

6. Write and test a computer code that takes a given set of nodes, function values, and corresponding divided difference coefficients, and computes new divided difference coefficients for new nodes and function values. Test it by recursively computing interpolating polynomials that approximate $f(x) = e^x$ on the interval [0, 1].

7. In 1973 the horse Secretariat became the first (and, so far, only) winner of the Kentucky Derby to finish the race in less than 2 minutes, running the 1 $\frac{1}{4}$-mile distance in 1 minute, 59.4 seconds, a record that still stands as this edition goes to press in 2013. Remarkably, he ran each quarter mile *faster* than the previous one, as Table 4.6 shows. Here t is the elapsed time (in seconds) since the race began and x

Table 4.6 Data for Problem 7.

x	0.0	0.25	0.50	0.75	1.00	1.25
t	0.0	25.0	49.4	73.0	96.4	119.4

is the distance (in miles) that Secretariat has traveled.

(a) Find the cubic polynomial that interpolates this data at $x = 0, 1/2, 3/4, 5/4$.

(b) Use this polynomial to estimate Secretariat's speed at the finish of the race, by finding $p'_3(5/4)$.

(c) Find the quintic polynomial that interpolates the entire data set.

(d) Use the quintic polynomial to estimate Secretariat's speed at the finish line.

8. The data in Table 4.7 gives the actual thermal conductivity data for the element mercury. Use Newton interpolation and the data for 300 K, 500 K, and 700 K to construct a quadratic interpolate for this data. How well does it predict the values at 400 K and 600 K?

Table 4.7 Data for Problem 8.

Temperature (° K), u	300	400	500	600	700
Conductivity (W/cm ° K), k	0.084	0.098	0.109	0.12	0.127

9. The gamma function, denoted by $\Gamma(x)$, is an important special function in probability, combinatorics, and other areas of applied mathematics. Because it can be shown that $\Gamma(n + 1) = n!$, the gamma function is considered a generalization of the factorial function to non-integer arguments. Table 4.8 gives the values of $\Gamma(x)$ on the interval [1, 2]. Use these to construct the fifth-degree polynomial based on the nodes $x = 1, 1.2, 1.4, 1.6, 1.8, 2.0$, and then use this polynomial to estimate the values at $x = 1.1, 1.3, 1.5, 1.7, 1.9$. Plot your polynomial and compare it to the intrinsic gamma function on your computing system or

calculator.

Table 4.8 Table of $\Gamma(x)$ values.

x	$\Gamma(x)$
1.00	1.0000000000
1.10	0.9513507699
1.20	0.9181687424
1.30	0.8974706963
1.40	0.8872638175
1.50	0.8862269255
1.60	0.8935153493
1.70	0.9086387329
1.80	0.9313837710
1.90	0.9617658319
2.00	1.0000000000

10. The error function, which we saw briefly in Chapters 1 and 2, is another important special function in applied mathematics, with applications to probability theory and the solution of heat conduction problems. The formal definition of the error function is

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Table 4.9 gives values of erf(x) in increments of 0.1, over the interval [0, 1].

Table 4.9 Table of erf(x) values for Problem 10.

x	erf(x)
0.0	0.00000000000000
0.1	0.11246291601828
0.2	0.22270258921048
0.3	0.32862675945913
0.4	0.42839235504667
0.5	0.52049987781305
0.6	0.60385609084793
0.7	0.67780119383742
0.8	0.74210096470766
0.9	0.79690821242283
1.0	0.84270079294971

(a) Construct the quadratic interpolating polynomial to the error function using the data at the nodes $x_0 = 0$, $x_1 = 0.5$, and $x_2 = 1.0$. Plot the polynomial and the data from Table 4.9 and comment on the observed accuracy.

(b) Repeat part (a), but this time construct the cubic interpolating polynomial using the nodes $x_0 = 0.0$, $x_2 = 0.3$, $x_2 = 0.7$, and $x_3 = 1.0$.

11. As steam is heated up, the pressure it generates is increased. Over the temperature

range [220, 300] (degrees Fahrenheit) the pressure, in pounds per square inch, is as given in Table 4.10.[6]

Table 4.10 Temperature–pressure values for steam; Problem 11

T	220	230	240	250	260	270	280	290	300
P	17.188	20.78	24.97	29.82	35.42	41.85	49.18	57.53	66.98

(a) Construct the quadratic interpolating polynomial to this data at the nodes $T_0 = 220$, $T_1 = 260$, and $T_2 = 300$. Plot the polynomial and the data in the table and comment on the accuracy you obtain.

(b) Repeat Part (a), but this time construct the quartic interpolating polynomial using the nodes $T_0 = 220$, $T_1 = 240$, $T_2 = 260$, $T_3 = 280$, and $T_4 = 300$.

(c) Which of the two polynomials would you think it is best to use to get values for $P(T)$ that are not in Table 4.10?

12. Similar data for gaseous ammonia is given in Table 4.11.

Table 4.11 Temperature—pressure values for gaseous ammonia: Problem 12

T	0	5	10	15	20	25	30	35	40
P	30.42	34.27	38.51	43.14	48.21	53.73	59.74	66.26	73.32

(a) Construct the quadratic interpolating polynomial to this data at the nodes $T_0 = 0$, $T_1 = 20$, and $T_2 = 40$. Plot the polynomial and the data in the table and comment on the accuracy you observe.

(b) Repeat part (a), but this time construct the quartic interpolating polynomial using the nodes $T_0 = 0$, $T_1 = 10$, $T_2 = 20$, $T_3 = 30$, and $T_4 = 40$.

(c) Which of the two polynomials would you think is best to use to get values for $P(T)$ that are not in Table 4.11?

13. In Problems 8 of §3.1 and 8 of §3.8, we looked at the motion of a liquid–solid interface under a simplified model of the physics involved, in which the interface moved according to

$$x = 2\beta\sqrt{t}$$

for $\beta = \alpha/\sqrt{k}$. Here k is a material property and α is the root of $f(z) = \theta e^{-z^2} - z\,\mathrm{erf}(z)$, where θ also depends on material properties. Figure 4.4 shows a plot of α versus $\log_{10}\theta$, based on finding the root of $f(z)$. Some of the data used to create this curve is given in Table 4.12.

Figure 4.4 Figure for Problem 13.

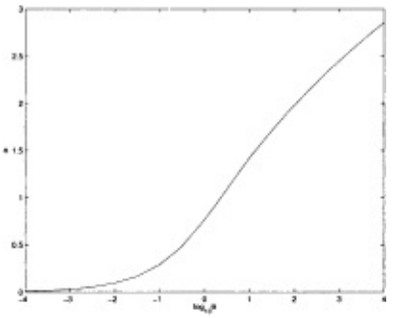


Table 4.12 Data for Problem 13.

$\log_{10}\theta$	α
−6.0000	0.944138E−03
−5.0000	0.298500E−02
−4.0000	0.941277E−02
−3.0000	0.297451E−01
−2.0000	0.938511E−01
−1.0000	0.289450E+00
0.0000	0.767736E+00
1.0000	0.141492E+01
2.0000	0.198151E+01
3.0000	0.245183E+01
4.0000	0.285669E+01
5.0000	0.321632E+01
6.0000	0.354269E+01

(a) Use the data at the nodes {−6, −4, −2,0, 2, 4, 6} to construct an interpolating polynomial for this table. Plot the polynomial and compare to the actual graph in Figure 4.4. Use the polynomial to compute values at $\log_{10}\theta = -5, -3, \ldots. 3, 5$. How do these compare to the actual values from Table 4.12?

(b) Compute the higher-degree Newton polynomial based on the entire table of data. Plot this polynomial and compare it to the one generated using only part of the data.

14. Write a computer program to construct the Newton interpolating polynomial to $f(x) = \sqrt{x}$ using equally spaced nodes on the interval [0, 1]. Plot the error $f(x) - p_n(x)$ for $n = 4, 8, 16$ and comment on what you get.

4.3 INTERPOLATION ERROR

How good an approximation is the interpolating polynomial? We saw in §2.4 that an error estimate was fairly easily derived for linear interpolation, and here we will generalize that result to arbitrary polynomial interpolation. We expect, based on the examples $f(x) = e^x$ and $f(x) = (1 + 25x^2)^{-1}$, that using more points could make the approximation more accurate and yet could also lead to problems.

Theorem 4.3 (Polynomial Interpolation Error Theorem) *Let $f \in C^{n+1}([a, b])$ and let the nodes $x_k \in [a, b]$ for $0 \le k \le n$. Then, for each $x \in [a, b]$, there is a $\xi_x \in [a, b]$ such that*

$$f(x) - p_n(x) = \frac{w_n(x)}{(n+1)!} f^{(n+1)}(\xi_x), \tag{4.6}$$

where

$$w_n(x) = \prod_{k=0}^{n} (x - x_k).$$

Proof: The proof is an involved argument using repeated applications of Rolle's Theorem, and is deferred to Appendix A.1. •

This result is too general to be easily interpreted at this point in our study, so we will deal directly with some specific cases. Before doing so, we introduce some notation.

In measuring the error in certain approximations, it will be useful from now on to have some convenient means of measuring the *size* of a function. This is done via the concept of a *norm*. Briefly, a norm is any mapping from functions into the non-negative real numbers (usually denoted by a double-bar notation: $\|f\|$), which satisfies certain basic axioms.

Definition 4.1 (Function norm) *A function norm is any computation, denoted by the symbol $\|f\|$, that satisfies the following conditions:*

1. $\|f\| > 0$ *for any function f that is not identically zero;*
2. $\|af\| = |a|\|f\|$ *for any constant a.*
3. $\|f + g\| \le \|f\| + \|g\|$ *for any two functions f and g.*

Note that the use of the double vertical bar notation ($\|f\|$) is similar to the use of the single vertical bar notation for the absolute value; this is deliberate, since the notion of a norm plays much the same role for functions as the absolute value does for ordinary numbers: it helps us to measure size, or magnitude.

Some examples of common norms include:

- The *infinity norm* or pointwise norm; if f is continuous on the closed interval $[a, b]$, then we define
$$\|f\|_{\infty,[a,b]} = \max_{x \in [a,b]} |f(x)|.$$
- The *2-norm*; if f is such that its square can be integrated over the interval $[a, b]$, then we define
$$\|f\|_{2,[a,b]} = \left(\int_a^b [f(x)]^2 dx \right)^{1/2}.$$

Note the use of subscripts on the norm notation to distinguish between different norms; the subscript defining the interval will often be dropped or omitted when there is no danger of confusion. Note also that different norms will give different numerical values for the same function, and will convey different kinds of information.

■ EXAMPLE 4.4

If $f(x) = e^{-x}$ on the interval $[-1, 1]$, then

$$\|f\|_\infty = e = 2.71828...$$

and

$$\|f\|_2 = \left(\frac{e^2 - e^{-2}}{2}\right)^{1/2} = 1.90443...$$

See Problems 14 and 15 for some further exploration of what different norms mean.

We already know the interpolation error result for $n = 1$: in other words, linear interpolation. In §2.4, we saw that we could prove

$$|f(x) - p_1(x)| \leq \frac{1}{8}(x_1 - x_0)^2 \max_{x_0 \leq x \leq x_1} |f''(x)|, \tag{4.7}$$

which holds for all $x \in [x_0, x_1]$. We can write this in norm notation as

$$\|f - p\|_{\infty,I} \leq \frac{1}{8}(x_1 - x_0)^2 \|f''\|_{\infty,I}, \tag{4.8}$$

where $I = [x_0, x_1]$ is the interval defined by the nodes used in the interpolation. What happens for larger values of n?

If $n = 2$ (quadratic interpolation), we have, from (4.6),

$$f(x) - p_2(x) = \frac{1}{6}(x - x_0)(x - x_1)(x - x_2) f'''(\xi_x). \tag{4.9}$$

To get an upper bound like we did for linear interpolation, we assume that the nodes satisfy

$$x_1 - x_0 = x_2 - x_1 = h$$

for some $h > 0$. We thus can write

$$(x - x_0)(x - x_1)(x - x_2) = ((x - x_1) + h)(x - x_1)((x - x_1) - h) = t(t^2 - h^2)$$

for $t = x - x_1$. We now have

$$|f(x) - p_2(x)| \leq \frac{1}{6}\left(\max_{-h \leq t \leq h} |t(t^2 - h^2)|\right) \max_{x_0 \leq t \leq x_2} |f'''(t)|, \tag{4.10}$$

which holds for all $x \in [x_0, x_2]$. Following the same kind of argument used in §2.4, we can show that

$$\max_{-h \leq t \leq h} |t(t^2 - h^2)| = \frac{2}{3\sqrt{3}} h^3$$

so that the upper bound on the error becomes

$$|f(x) - p_2(x)| \leq \frac{1}{9\sqrt{3}} h^3 \max_{x_0 \leq t \leq x_2} |f'''(t)|. \tag{4.11}$$

This can be rendered into norm notation as

$$\|f - p_2\|_{\infty,I} \leq \frac{1}{9\sqrt{3}} h^3 \|f'''\|_{\infty,I} \tag{4.12}$$

where I is again the interval defined by the smallest and largest nodes, and h is the spacing between nodes.

■ EXAMPLE 4.5

Let $f(x)$ be the error function, i.e.,

$$f(x) = \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Suppose that we want to construct interpolating polynomial approximations to this function on the interval [0, 1] that are accurate to within 10^{-6}. We have

$$f''(x) = \frac{2}{\sqrt{\pi}} \left(e^{-x^2}\right)' = \frac{2}{\sqrt{\pi}} \left(-2xe^{-x^2}\right),$$

so that a crude upper bound on the second derivative is

$$\|f''\|_{\infty,[0,1]} \leq 2 \times 1 \times 2e^{-x^2}/\sqrt{\pi} \leq 2 \times 2e^0/\sqrt{\pi} \leq 7.1.$$

Therefore the error in linear interpolation is bounded by $(h^2/8)(7.1) = h^2(0.8875)$; thus we need to have $h \leq 9.5 \times 10^{-4}$ to get the error less than 10^{-6}. This tells us that the table of error function values must have $N = 1/h > 1,062$ points for linear interpolation between points to achieve the desired accuracy.

On the other hand, we have

$$f'''(x) = \frac{2}{\sqrt{\pi}} \left(-2e^{-x^2} + 4x^2e^{-x^2}\right) = \frac{4}{\sqrt{\pi}} \left(2x^2 - 1\right) e^{-x^2},$$

for which we have (again, crudely)

$$\|f'''\|_{\infty,[0,1]} \leq 4 \times (1) \times e^{-x^2}/\sqrt{\pi} \leq 4 \times e^0/\sqrt{\pi} \leq 5.$$

Hence, the error in quadratic interpolation is bounded by $(h^3/9\sqrt{3})(5) \leq 0.4h^3$. To get the error less than the desired 10^{-6}, we then want to have

$$0.4h^3 \leq 10^{-6} \Rightarrow h \leq (0.4)^{-1/3} \times 10^{-2} \leq 1.4 \times 10^{-2}.$$

So we now have to have $N = 1/h > 72$ points in the table to get the desired accuracy.

A similar analysis to that used to get (4.12) shows that the error in cubic interpolation satisfies

$$\|f - p_3\| \leq \frac{1}{24} h^4 \|f^{(4)}\|_{\infty,I}. \tag{4.13}$$

See Problem 4.

We could of course keep going on with specific cases, but since we will eventually learn, in §4.12.1, that interpolation with high-degree polynomials is not always a good idea, we will stop at this point.

Exercises:

1. What is the error in quadratic interpolation to $f(x) = \sqrt{x}$, using equally spaced nodes on the interval $[\frac{1}{4}, 1]$?

2. Repeat the above for $f(x) = x^{-1}$ on $[\frac{1}{2}, 1]$.

3. Repeat the previous two problems, using cubic interpolation.

4. Show that the error in third-degree polynomial interpolation satisfies

$$\|f - p_3\|_\infty \le \frac{1}{24} h^4 \|f^{(4)}\|_\infty,$$

if the nodes x_0, x_1, x_2, x_3 are equally spaced, with $x_i - x_{i-1} = h$. *Hint:* Use the change of variable $t = x - x - \frac{1}{2}h$.

5. Show that the error in polynomial interpolation using six equally spaced points (quintic interpolation) satisfies

$$\|f - p_5\|_\infty \le Ch^6 \|f^{(6)}\|_\infty,$$

where $C \approx 0.0235$. *Hint:* See previous problem.

6. Generalize the derivation of the error bound (4.12) for quadratic interpolation to the case where the nodes are *not* equally spaced. Take $x_1 - x_0 = h$ and $x_2 - x_1 = \theta h$ for some $\theta > 0$.

7. Apply your result for the previous problem to the error in quadratic interpolation to $f(x) = \sqrt{x}$ using the nodes $x_0 = \frac{1}{4}$, $x_1 = \frac{9}{16}$, and $x_2 = 1$.

8. If we want to use a table of exponential values to interpolate the exponential function on the interval $[-1, 1]$, how many nodes are needed to guarantee 10^{-6} accuracy with linear interpolation? Quadratic interpolation?

9. If we want to use a table of values to interpolate the error function on the interval $[0, 5]$, how many points are needed to get 10^{-6} accuracy using linear interpolation? Quadratic interpolation? Would it make sense to use one grid spacing on, say, $[0, 1]$, and another one on $[1, 5]$? Explain.

10. If we want to use a table of values to interpolate the sine function on the interval $[0, \pi]$, how many points are needed for 10^{-6} accuracy with linear interpolation? Quadratic interpolation? Cubic interpolation?

11. Let's return to the computation of the natural logarithm. Consider a computer that stores numbers in the form $z = f \cdot 2^\beta$, where $\frac{1}{2} \le f \le 1$. We want to consider using this, in conjunction with interpolation ideas, to compute the natural logarithm function.

(a) Using piecewise linear interpolation over a grid of equally spaced points, how many table entries would be required to accurately approximate ln z to within 10^{-14}?

(b) Repeat part (a), using piecewise quadratic interpolation.

(c) Repeat part (a) again, using piecewise cubic interpolation.

Explain, in a brief essay, the importance of restricting the domain of z to the interval $[\frac{1}{2}, 1]$.

12. Assume that for any real c,

$$\lim_{n\to\infty} \frac{c^n}{n!} = 0.$$

Use this to prove that, if p_n is the polynomial interpolate to $f(x) = \sin x$ on the interval $[a, b]$, using *any* distribution of distinct nodes x_k, $0 \le k \le n$, then $\|f - p_n\|_\infty \to 0$ as $n \to$

∞. *Hint:* Can we justify $|w_n(x)| \le c^{n+1}$ for some c?

13. Can you repeat the above for $f(x) = e^x$? Why/why not?

14. Define the norms

$$\|f\|_\infty = \max_{x\in[0,1]} |f(x)|$$

and

$$\|f\|_2 = \left(\int_0^1 [f(x)]^2 dx\right)^{1/2}.$$

Compute $\|f\|_\infty$ and $\|f\|_2$ for the following list of functions:

(a) e^{-ax}, $a > 0$;

(b) $\sin n\pi x$, n integer;

(c) $\sqrt{x}e^{-ax^2}$, $a > 0$;, $a > 0$;

(d) $1/\sqrt{1+x}$.

In parts (b) and (c), a is a constant parameter.

15. Define

$$f_n(x) = \begin{cases} 1 - nx, & 0 \le x \le \frac{1}{n}; \\ 0, & \frac{1}{n} \le x \le 1. \end{cases}$$

Show that

$$\lim_{n\to\infty} \|f_n(x)\|_\infty = 1,$$

but

$$\lim_{n\to\infty} \|f_n(x)\|_2 = 0.$$

Hint: Draw a graph of $f_n(x)$.

16. Show that

$$\|f\|_{1,[a,b]} = \int_a^b |f(x)|dx$$

defines a function norm.

◁ ● ● ● ▷

4.4 APPLICATION: MULLER'S METHOD AND INVERSE QUADRATIC INTERPOLATION

We can use the idea of interpolation to develop more sophisticated root-finding methods.

The secant method can be viewed as finding the root of a linear interpolating polynomial. Our error results tell us that a quadratic (second-degree) polynomial should be a better approximation to the function; hence, its root ought to be a better approximation to the root of the original function. Can we take this outline of an idea and use it to construct a practical method for finding roots?

The answer is yes, and the method is known as *Muller's method*. Given three points, x_0,

x_1, and x_2, we find the quadratic polynomial $p(x)$ such that $p(x_i) = f(x_i)$, $i = 0, 1, 2$; and then define x_3 as the root of p that is closest to x_2. The actual implementation of Muller's method does present a few problems that need to be resolved, however.

The best way to write the interpolating polynomial is in the Newton form, as[7]

(4.14) $$p(x) = f(x_2) + A(x - x_2) + B(x - x_2)(x - x_1),$$

where A and B are the divided differences (see Problem 7)

(4.15) $$A = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

and

(4.16) $$B = \frac{1}{x_2 - x_0}\left(\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}\right).$$

Note that these coefficients will have to be re-computed for each iteration.

Finding the root of (4.14) via the quadratic formula requires that some care be taken. Since we want to find the root nearest x_2, we rewrite p as a quadratic in $x - x_2$:

$$\begin{aligned} p(x) &= f(x_2) + A(x - x_2) + B(x - x_2)(x - x_1) \\ &= f(x_2) + A(x - x_2) + B(x - x_2)(x - x_2) + B(x - x_2)(x_2 - x_1) \\ &= f(x_2) + (A + B(x_2 - x_1))(x - x_2) + B(x - x_2)(x - x_2) \\ &= f(x_2) + C(x - x_2) + B(x - x_2)^2, \end{aligned}$$

where $C = A + B(x_2 - x_1)$. The quadratic formula then says that the root of p is defined by

$$x - x_2 = \frac{1}{2B}\left(-C \pm \sqrt{C^2 - 4f(x_2)B}\right),$$

where we choose the sign to minimize the right side of the equation. To avoid the pitfalls of subtractive cancellation, we rationalize the numerator to get

$$x - x_2 = -\frac{2f(x_2)}{C \pm \sqrt{C^2 - 4f(x_2)B}},$$

where the sign is chosen to maximize the denominator. Thus, the next point in the iteration is given by

$$x_3 = x_2 - \frac{2f(x_2)}{C + \operatorname{sgn}(C)\sqrt{C^2 - 4f(x_2)B}},$$

where sgn is the "sign" function, defined by

$$\operatorname{sgn}(x) = \begin{cases} -1, & x < 0; \\ 1, & x > 0; \\ 0, & x = 0. \end{cases}$$

Thus, the general case is given by

$$x_{n+1} = x_n - \frac{2f(x_n)}{C_n + \operatorname{sgn}(C_n)\sqrt{C_n^2 - 4f(x_n)B_n}},$$

where

$$B_n = \frac{1}{x_n - x_{n-2}}\left(\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} - \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}\right)$$

and

$$C_n = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} + B_n(x_n - x_{n-1}).$$

Table 4.13 shows the result of applying Muller's method to the example we used thoughout Chapter 3, $f(x) = 2 - e^x$, so that the exact root is $\alpha = \ln 2$. Compare the number of iterations used here to that of the secant method (Table 3.5); note that we can code Muller's method to use only a single new function evaluation in each step.

Table 4.13 Muller's method, $f(x) = 2 - e^x$.

n	x_n	$f(x_n)$	$\alpha - x_n$	$\log_{10}(\alpha - x_n)$
0	0.000000000000	0.100000e+01	0.693147e+00	–0.1592
1	0.500000000000	0.351279e+00	0.193147e+00	–0.7141
2	1.000000000000	–0.718282e+00	0.306853e+00	–0.5131
3	0.687259367753	0.117410e–01	0.588781e–02	–2.2300
4	0.693087847691	0.118662e–03	0.593329e–04	–4.2267
5	0.693147161269	0.385812e–07	0.192906e–07	–7.7147
6	0.693147180560	–0.222045e–14	0.122125e–14	–14.9132
7	0.693147180560	0.000000e+00	0.111022e–15	–15.9546

An alternative to Muller's method, which avoids the difficulties associated with the quadratic formula, is *inverse quadratic interpolation*. In this case, we find the quadratic polynomial $q(y)$ such that

$$q(f(x_i)) = x_i, \quad i = 0, 1, 2.$$

Direct computation with the Newton form shows that

$$q(y) = x_2 + a(y - y_2) + b(y - y_2)(y - y_1),$$

where $y_k = f(x_k)$,

$$a = \frac{x_2 - x_1}{y_2 - y_1}, \tag{4.17}$$

and

$$b = \frac{1}{y_2 - y_0}\left(\frac{x_2 - x_1}{y_2 - y_1} - \frac{x_1 - x_0}{y_1 - y_0}\right). \tag{4.18}$$

Thus, q is quadratic in y and still passes through the points $(x_i, f(x_i))$; we have essentially reversed the notion of ordinate and abscissa in our coordinate system. The approximate root is the value of $q(y) = x$ corresponding to $y = 0$; thus, it is obtained by the simple expedient of evaluating q at 0:

$$x_3 = q(0) = x_2 - ay_2 + by_2y_1.$$

In the general case, we have

$$x_{n+1} = x_n - a_n f(x_n) + b_n f(x_n) f(x_{n-1}),$$

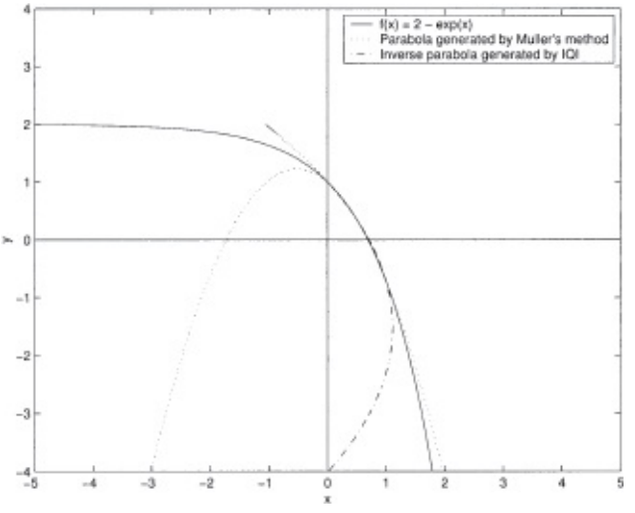
where a_n and b_n are computed from appropriately generalized versions of (4.17) and (4.18). Table 4.14 shows the result of applying inverse quadratic interpolation to the example $f(x) = 2 - e^x$.

Table 4.14 Inverse quadratic interpolation, $f(x) = 2 - e^x$.

n	x_n	$f(x_n)$	$\alpha - x_n$	$\log_{10}(\alpha - x_n)$
0	0.000000000000	0.100000e+01	0.693147e+00	–0.1592
1	0.500000000000	0.351279e+00	0.193147e+00	–0.7141
2	1.000000000000	–0.718282e+00	0.306853e+00	–0.5131
3	0.708748678748	–0.314477e–01	0.156015e–01	–1.8068
4	0.692849949759	0.594373e–03	0.297231e–03	–3.5269
5	0.693146743432	0.874255e–06	0.437127e–06	–6.3594
6	0.693147180561	–0.134603e–11	0.673128e–12	–12.1719
7	0.693147180560	0.000000e+00	0.111022e–15	–15.9546

Figure 4.5 shows a graph of $f(x) = 2 - e^x$, along with the graphs of the parabola generated by Muller's method and the inverse parabola generated by inverse quadratic interpolation, using $x_0 = 0$, $x_1 = 1$, and $x_2 = 1/2$ as the initial values in each case.

Figure 4.5 Muller's method and inverse quadratic interpolation.



The convergence theory for both Muller's method and inverse quadratic interpolation is beyond the intended scope of this text, so we will content ourselves with reporting that both methods are locally convergent and superlinear, with $p \approx 1.84$. Note that this order of convergence is not much greater than that for the secant method.

One great utility of Muller's method is that it is able to find complex roots of real-valued functions, because of the square root in the computation. (The same is true of Halley's method (3.43).)

Inverse quadratic interpolation is used as part of the Brent–Dekker root-finding algorithm, which is a commonly implemented automatic root-finding program.

Exercises:

1. Do three steps of Muller's method for $f(x) = 2 - e^x$, using the initial points 0, 1/2, and 1. Make sure that you reproduce the values in Table 4.13.

2. Repeat the above for inverse quadratic interpolation.

3. Let $f(x) = x^6 - x - 1$; show that this has a root on the interval [0, 2] and do three steps of Muller's method using the ends of the interval plus the midpoint as the initial values.

4. Repeat the above using inverse quadratic interpolation.

5. Let $f(x) = e^x$ and consider the nodes $x_0 = -1$, $x_1 = 0$, and $x_2 = 1$. Let p_2 be the quadratic polynomial that interpolates f at these nodes, and let q_2 be the quadratic polynomial (in y) that *inverse*-interpolates f at these nodes. Construct $p_2(x)$ and $q_2(y)$ and plot both, together with $f(x)$.

6. Repeat the above using $f(x) = \sin \pi x$ and the nodes $x_0 = 0$, $x_1 = \frac{1}{4}$, and $x_2 = \frac{1}{2}$.

7. Show that the formulas (4.15) and (4.16) for the divided-difference coefficients in Muller's method (4.14) are correct.

8. Show that the formulas (4.17) and (4.18) for the coefficients in inverse quadratic interpolation are correct.

9. Apply Muller's method and inverse quadratic interpolation to the functions in Problem 3 of §3.1, and compare your results to those obtained with Newton's method and/or the secant method.

10. Refer back to the discussion of the error estimate for the secant method in §3.11.3. Adapt this argument to derive an error estimate for Muller's method.

◁ ● ● ● ▷

4.5 APPLICATION: MORE APPROXIMATIONS TO THE DERIVATIVE

We can use polynomial interpolation to derive more formulas for approximating the derivative. Essentially, since interpolation error theory shows that polynomials can approximate more complicated functions reasonably well, we infer that the derivative of a polynomial interpolate might approximate the derivative of the complicated function reasonably well.

The interpolation error theorem gives us that

$$f(x) - p_n(x) = \frac{1}{(n+1)!} w_n(x) f^{(n+1)}(\xi_x),$$

and the fact that this is an equality is important, for we can now differentiate both sides to get

$$f'(x) - p_n'(x) = \frac{1}{(n+1)!} \frac{d}{dx} \left[w_n(x) f^{(n+1)}(\xi_x) \right].$$

We have to be careful evaluating the derivative of the bracketed term; recall that ξ_x depends on x, so that

$$\frac{d}{dx} \left[w_n(x) f^{(n+1)}(\xi_x) \right] = w_n'(x) f^{(n+1)}(\xi_x) + w_n(x) \frac{d}{dx} \left[f^{(n+1)}(\xi_x) \right]. \tag{4.19}$$

The problem in using this formula is that we do not know how ξ_x depends on x; thus, we cannot estimate or predict the size of the derivative in the second term. However, if we evaluate the approximation at one of the nodes, x_i, then we will have $w_n(x_i) = 0$ and the

second term on the right side of (4.19) will vanish, leaving us with

$$f'(x_i) - p_n'(x_i) = \frac{1}{(n+1)!} w_n'(x_i) f^{(n+1)}(\xi_i), \tag{4.20}$$

where we use $\xi_i = \xi_{x_i}$ for simplicity of notation.[8]

We can illustrate this by using a quadratic polynomial to approximate f, and thus derive three formulas for approximating the first derivative. Let the nodes be denoted by x_0, x_1, and x_2, and assume that they are equally spaced, so we have $x_1 - x_0 = h$ and $x_2 - x_1 = h$, and so on. Then the interpolating polynomial in Lagrange form is

$$p_2(x) = f(x_0)L_0^{(2)}(x) + f(x_1)L_1^{(2)}(x) + f(x_2)L_2^{(2)}(x).$$

The approximate derivative is then

$$f'(x_i) \approx p_2'(x_i) = f(x_0)(L_0^{(2)})'(x_i) + f(x_1)(L_1^{(2)})'(x_i) + f(x_2)(L_2^{(2)})'(x_i),$$

where $i = 0, 1, 2$. The error is given as in (4.20); thus,

$$f'(x_i) - p_2'(x_i) = \frac{1}{6} w_2'(x_i) f'''(\xi_i)$$

and we only need to evaluate $(L_j^{(2)})'(x_i)$ and $w'_2(x_i)$ for $i, j = 0, 1, 2$, to complete the construction of the approximations. We get

$$\begin{aligned}
&(L_0^{(2)})'(x_0) = -3/2h, \quad (L_1^{(2)})'(x_0) = 2/h, \quad (L_2^{(2)})'(x_0) = -1/2h,\\
&(L_0^{(2)})'(x_1) = -1/2h, \quad (L_1^{(2)})'(x_1) = 0, \quad (L_2^{(2)})'(x_1) = 1/2h,\\
&(L_0^{(2)})'(x_2) = 1/2h, \quad (L_1^{(2)})'(x_2) = -2/h, \quad (L_2^{(2)})'(x_2) = 3/2h,
\end{aligned}$$

and

$$w_2'(x_0) = 2h^2, \quad w_2'(x_1) = -h^2, \quad w_2'(x_2) = 2h^2,$$

so that the approximations are

$$f'(x_0) \approx \frac{1}{2h}\left(-f(x_2) + 4f(x_1) - 3f(x_0)\right), \tag{4.21}$$

$$f'(x_1) \approx \frac{1}{2h}\left(f(x_2) - f(x_0)\right), \tag{4.22}$$

$$f'(x_2) \approx \frac{1}{2h}\left(3f(x_2) - 4f(x_1) + f(x_0)\right), \tag{4.23}$$

with errors

$$f'(x_0) - \frac{1}{2h}\left(-f(x_2) + 4f(x_1) - 3f(x_0)\right) = \frac{1}{3}h^2 f'''(\xi_0), \tag{4.24}$$

$$f'(x_1) - \frac{1}{2h}\left(f(x_2) - f(x_0)\right) = -\frac{1}{6}h^2 f'''(\xi_1), \tag{4.25}$$

$$f'(x_2) - \frac{1}{2h}\left(3f(x_2) - 4f(x_1) + f(x_0)\right) = \frac{1}{3}h^2 f'''(\xi_2). \tag{4.26}$$

We recognize the approximation (4.22) as the same central difference formula derived in §2.2; the other two formulas are new. All three are second-order accurate, in that the error is $\mathcal{O}(h^2)$, but (4.21) and (4.23) require three function evaluations, whereas (4.22) requires only two. In some sense, this makes (4.22) a "better" approximation to use, but we will see situations where (4.21) and (4.23) are more useful in §§5.2 and 6.6.2.

Consider, for example, a function defined only on a grid of equally spaced points. Thus we have $y_k = f(x_k)$ for only the nodes x_k. Suppose further that we need an accurate approximation to $f'(x_0)$ (the derivative at the left endpoint) or $f'(x_n)$ (the derivative at the right endpoint). We can't use the centered difference approximation (2.5) at the endpoints, but we can use (4.24) and (4.26) to get

$$f'(x) = \left(\frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h}\right) + \mathcal{O}(h^2)$$

and

$$f'(x) = \left(\frac{3f(x) - 4f(x-h) + f(x-2h)}{2h}\right) + \mathcal{O}(h^2).$$

This will be useful when we need derivative approximations at the ends of intervals for constructing spline approximations (§4.8) or for improving the trapezoid rule (§5.2).

Exercises:

1. Apply the derivative approximations (4.21) and (4.23) to the approximation of $f'(x)$ for $f(x) = x^3 + x^2 + 1$ for $x = 1$ and $h = \frac{1}{8}$.

2. Apply the derivative approximations (4.21) and (4.23) to the same set of functions as in Problem 3 of §2.2, using a decreasing sequence of mesh values, $h^{-1} = 2, 4, 8, \ldots$. Do we achieve the expected rate of accuracy as h decreases?

3. Derive a version of (4.24) under the assumption that $x_1 - x_0 = h$, but $x_2 - x_1 = \eta = \theta h$ for some real, positive, θ. Be sure to include the error estimate as part of your work, and confirm that when $\theta = 1$ you get the same results as in the text.

4. Repeat the above for (4.25).

5. Repeat the above for (4.26).

6. Use the derivative approximations from this section to construct a table of values for the derivative of the gamma function, based on the data in Table 4.8.

7. Try to extend the ideas of this section to construct an approximation to $f''(x_k)$. Is it possible? What happens?

◁ ● ● ● ▷

4.6 HERMITE INTERPOLATION

So far, we have studied interpolation that involved only knowledge of function values. What happens if we include information about the derivative as well?

To be specific, we want to find a polynomial $p(x)$ that interpolates $f(x)$ in the sense that[9]

$$p(x_i) = f(x_i), \quad p'(x_i) = f'(x_i), \quad 1 \le i \le n.$$

This is known as the *Hermite*[10] *interpolation problem*. Can we do this? The answer is yes, as seen in the following theorem.

Theorem 4.4 (Hermite Interpolation Theorem) *Given the n nodes x_i, $1 \le i \le n$, and a*

differentiate function $f(x)$, if the nodes are distinct, then there exists a unique polynomial H_n of degree less than or equal to $2n-1$, such that

(4.27) $$H_n(x_i) = f(x_i), \quad H_n'(x_i) = f'(x_i), \quad 1 \le i \le n.$$

Proof: This is basically the same as the proof of Theorem 4.1. We define the two families of polynomials

$$h_k(x) = [1 - 2[L_k^{(n)}]'(x_k)(x - x_k)][L_k^{(n)}(x)]^2$$

and

$$\tilde{h}_k(x) = (x - x_k)[L_k^{(n)}(x)]^2;$$

now observe that

$$h_k(x_j) = \delta_{kj}, \qquad h_k'(x_j) = 0,$$

and

$$\tilde{h}_k(x_j) = 0, \qquad \tilde{h}_k'(x_j) = \delta_{kj}.$$

Therefore, the polynomial

$$H_n(x) = \sum_{k=1}^{n} \Big(f(x_k)h_k(x) + f'(x_k)\tilde{h}_k(x) \Big)$$

satisfies the interpolating conditions (4.27) and is clearly of degree less than or equal to $2n-1$. Uniqueness follows by the same argument as used before. •

Like the Lagrange form in ordinary interpolation, this form of the Hermite polynomial is not conducive to efficient computation. A variation of the Newton approach, including the use of divided difference tables, can be constructed; however, since the most common use of Hermite interpolation is the cubic case (which involves only two nodes, $x - a$ and $x = b$), we will forgo the more general development and show only the cubic construction.

We begin by setting up a divided difference table (Table 4.15), much as we did in §4.2. Note, however, that we enter the functional data twice and use the derivative data for part of the first differences column.

Table 4.15 Divided difference table for Hermite interpolation: initial setup.

k	x_k	$f_0(x_k)$	$f_1(x_k)$
1	a	$f(a)$	
			$f'(a)$
1	a	$f(a)$	
2	b	$f(b)$	
			$f'(b)$
2	b	$f(b)$	

We then complete the table just as we would an ordinary divided difference table, getting the results shown in Table 4.16.

Table 4.16 Divided difference table for Hermite interpolation: final form.

k	x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$	$f_3(x_k)$
1	a	$f(a)$			
			$f'(a)$		
1	a	$f(a)$		B	
			A		D
2	b	$f(b)$		C	
			$f'(b)$		
2	b	$f(b)$			

The elements denoted by letters are defined as follows:

$$A = \frac{f(b) - f(a)}{b - a}, \qquad B = \frac{A - f'(a)}{b - a},$$

$$C = \frac{f'(b) - A}{b - a}, \qquad D = \frac{C - B}{b - a}.$$

The polynomial is then given by

(4.28) $$H_2(x) = f(a) + f'(a)(x - a) + B(x - a)^2 + D(x - a)^2(x - b).$$

It can easily be checked that this does interpolate to f and f' at both a and b (see Problem 1).

■ EXAMPLE 4.6

As an illustration of Hermite interpolation, consider the problem of interpolating to the exponential function on the interval $[-1, 1]$ using data only at the endpoints. We have $x_1 = -1$, $x_2 = 1$, and the divided difference table is shown as Table 4.17, so that the polynomial is

Table 4.17 Divided difference table for cubic Hermite interpolation to $f(x) = e^x$.

k	x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$	$f_3(x_k)$
1	1	2.718281828			
			2.718281828		
1	1	2.718281828		0.771540317	
			1.175201194		0.1839397203
2	−1	0.3678794412		0.4036608764	
			0.3678794412		
2	−1	0.3678794412			

$$\begin{aligned} H_2(x) &= 2.718281828 + 2.718281828(x - 1) \\ &+ 0.771540317(x - 1)^2 + 0.1839397203(x - 1)^2(x + 1), \end{aligned}$$

or (after some simplification),

$$H_2(x) = 0.1839397206x^3 + 0.5876005967x^2 + 0.9912614728x + 0.9554800379.$$

Figure 4.6 shows a graph of both the exponential and H_2 (they are indistinguishable); Figure 4.7 shows the error $e^x - H_2(x)$.

Figure 4.6 Hermite interpolation to $f(x) = e^x$.

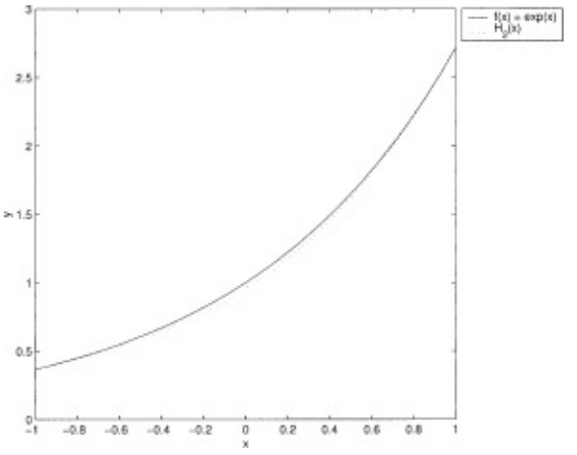
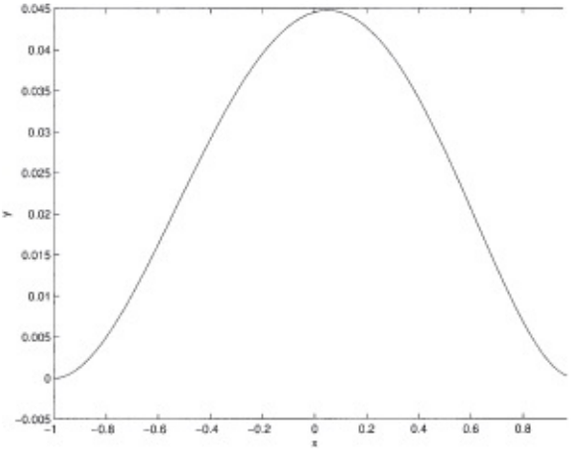


Figure 4.7 Error in Hermite interpolation to $f(x) = e^x$.



The accuracy of Hermite interpolation is based on a result very similar to Theorem 4.3.

Theorem 4.5 (Hermite Interpolation Error Theorem) *Let $f \in C^{2n}([a, b])$ and let the nodes $x_k \in [a, b]$ for all k, $1 \le k \le n$. Then, for each $x \in [a, b]$, there is a $\zeta_x \in [a, b]$ such that*

$$f(x) - H_n(x) = \frac{\psi_n(x)}{(2n)!} f^{(2n)}(\xi_x), \tag{4.29}$$

where

$$\psi_n(x) = \prod_{k=1}^{n} (x - x_k)^2.$$

Proof: Follows essentially the same argument as Theorem 4.3; see Appendix A.1 for details. •

Exercises:

1. Show that H_2, as defined in (4.28), is the cubic Hermite interpolate to f at the nodes $x = a$ and $x = b$.

2. Construct the cubic Hermite interpolate to $f(x) = \sin x$ using the nodes $a = 0$ and $b = \pi$. Plot the error between the interpolate and the function.

3. Construct the cubic Hermite interpolate to $f(x) = \sqrt{1 + x}$ using the nodes $a = 0$ and $b = 1$. Plot the error between the interpolate and the function.

4. Show that the error in cubic Hermite interpolation at the nodes $x = a$ and $x = b$ is given by

$$\|f - H_2\|_\infty \leq \frac{(b-a)^4}{384}\|f^{(4)}\|_\infty.$$

5. Construct the cubic Hermite interpolate to $f(x) = \sqrt{x}$ on the interval $[\frac{1}{4}, 1]$. What is the maximum error on this interval, as predicted by theory? What is the maximum error that actually occurs (as determined by observation; no need to do a complete calculus max/min problem)?

6. Construct the cubic Hermite interpolate to $f(x) = 1/x$ on the interval $[\frac{1}{2}, 1]$. What is the maximum error as predicted by theory? What is the actual (observed) maximum error?

7. Construct the cubic Hermite interpolate to $f(x) = x^{1/3}$ on the interval $[\frac{1}{8}, 1]$. What is the maximum error as predicted by theory? What is the actual (observed) maximum error?

8. Construct the cubic Hermite interpolate to $f(x) = \ln x$ on the interval $[\frac{1}{2}, 1]$. What is the maximum error as predicted by theory? What is the actual (observed) maximum error?

9. Extend the divided difference table for cubic Hermite interpolation to *quintic* Hermite interpolation, using the three nodes $x = a$, $x = b$, and $x = c$.

10. Construct the quintic Hermite interpolate to $f(x) = \ln x$ on the interval $[\frac{1}{2}, 1]$; use $x = 3/4$ as the third node.

11. What is the error in quintic Hermite interpolation?

12. Extend the ideas of §4.5 to allow us to compute second derivative approximations using Hermite interpolation.

4.7 PIECEWISE POLYNOMIAL INTERPOLATION

The results of Figure 4.3 suggest that interpolation might be unsatisfactory as an approximation tool. This is true if we insist on letting the *order* of the polynomial get larger and larger (see §4.12). However, if we keep the order of the polynomial fixed, and use different polynomials over different intervals, with the length of the intervals getting smaller and smaller, then interpolation can be a very accurate and powerful approximation tool. These ideas reach their full potential in §4.8 on splines; here we confine ourselves to the mere basics.

Consider the problem of constructing an approximation to a function $f(x)$, defined only by 101 equally spaced points on the interval $[-1, 1]$. If we use ordinary interpolation, we will have to construct a 100-degree polynomial, and we suspect that this might produce unsatisfactory results. However, we could use linear interpolation over each subinterval

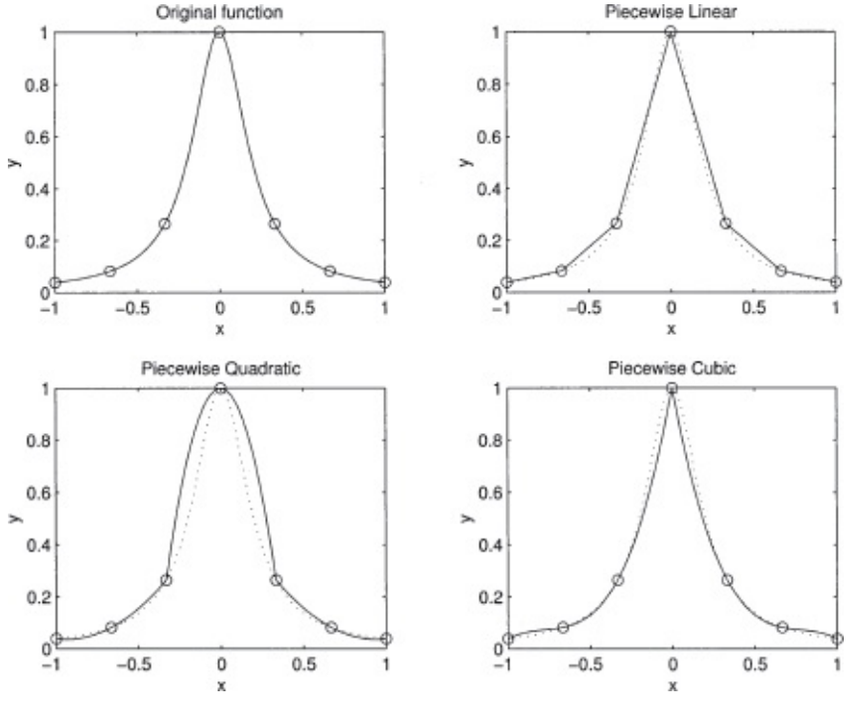
$[x_{k-1}, x_k]$ or quadratic interpolation over each subinterval $[x_{2k-2}, x_{2k}]$. That is, we define the approximating function as

$$q_d(x) = p_{d,k}(x), \quad x_{d(k-1)} \le x \le x_{dk},$$

where each $p_{d,k}$ is a polynomial of degree d. The accuracy of the approximation comes from the error estimates (4.7), (4.11), or (4.13), since the distance between nodes is small.

We illustrate this by approximating $f(x) = (1 + 25x^2)^{-1}$ over $[-1, 1]$ using the seven equally spaced points $x_k = -1 + x/3$, $k = 0,1, \ldots, 6$. The results are shown in Figure 4.8 for piecewise linear, quadratic, and cubic interpolation. For the piecewise quadratic case, we have (see Problem 3)

Figure 4.8 Piecewise interpolation to $f(x) = (1 + 25x^2)^{-1}$. The nodes are denoted by circles.



$$q_2(x) = \begin{cases} 0.0385 + 0.1323(x - x_0) + 0.6211(x - x_0)(x - x_1), & x_0 \le x \le x_2; \\ 0.2647 + 2.2059(x - x_2) - 6.6176(x - x_2)(x - x_3), & x_2 \le x \le x_4; \\ 0.2647 - 0.5464(x - x_4) + 0.6211(x - x_4)(x - x_5), & x_4 \le x \le x_6. \end{cases} \tag{4.30}$$

Note that all of these piecewise approximations are much more accurate than the 16-degree interpolating polynomial that was constructed in Figure 4.3C. This contrast is heightened even more when we take lots of points. Figures 4.9 and 4.10 show the results when we use 33 equally spaced points and piecewise quadratic approximation; compare this to Figure 4.3C.

Figure 4.9 Piecewise quadratic interpolation to $f(x) = (1 + 25x^2)^{-1}$ using 33 nodes.

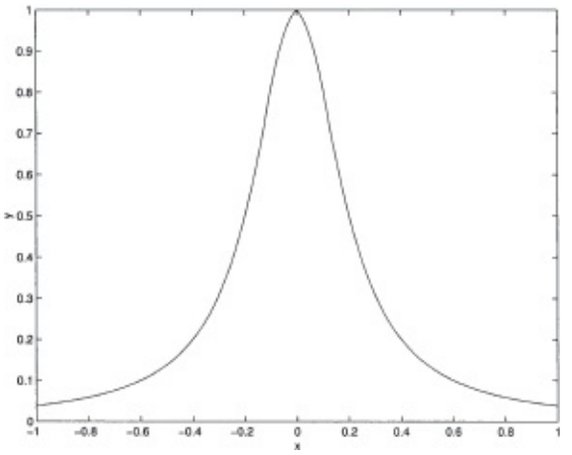
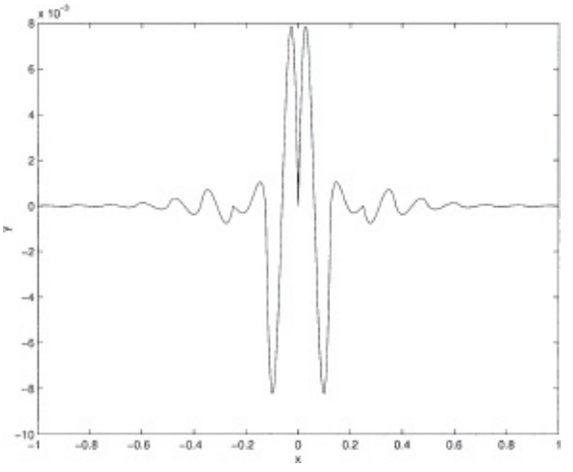


Figure 4.10 Error in 33-node piecewise quadratic interpolation to $f(x) = (1 + 25x^2)^{-1}$.



The use of different polynomials over different intervals complicates the entire process somewhat. We now have one set of points (the nodes, x_k, $0 \le k \le n$) that define the interpolation conditions, and a second set of points, called *knots*, that define the subintervals on which the separate polynomial pieces are defined. (In our presentation here we will assume that the knots are a subset of the nodes, but this doesn't have to be the case.) For a piecewise polynomial of degree d, the knots will be x_{dj}, $0 \le j \le m$, where m is the number of polynomial pieces in the approximation. Thus, for the quadratic example defined in (4.30), the knots are x_0, x_2, x_4, and x_6.

■ EXAMPLE 4.7

Consider the problem of constructing a piecewise quadratic approximation to $f(x) = \sin \pi x$, using the nodes $x_k = k/6$, $k = 0, 1, 2, \ldots, 6$. Thus there will be three separate polynomial approximations, and the knots will be x_0, x_2, x_4, and x_6.

We set up three divided difference tables (Table 4.18), one for each polynomial piece:

Table 4.18 Divided difference tables for piecewise quadratic approximation to $f(x) = \sin \pi x$.

x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$
0	0.0000000		
		3.0000000	
$\frac{1}{6}$	0.5000000		–2.4115427
		2.1961524	
$\frac{1}{3}$	0.8660254		

x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$
$\frac{1}{3}$	0.8660254		
		0.80384758	
$\frac{1}{2}$	1.0000000		–4.8230855
		–0.080384758	
$\frac{2}{3}$	0.8660254		

x_k	$f_0(x_k)$	$f_1(x_k)$	$f_2(x_k)$
$\frac{2}{3}$	0.8660254		
		–2.1961524	
$\frac{5}{6}$	0.5000000		–2.4115427
		–3.0000000	
1	0.0000000		

Thus we know that the three polynomials are given by

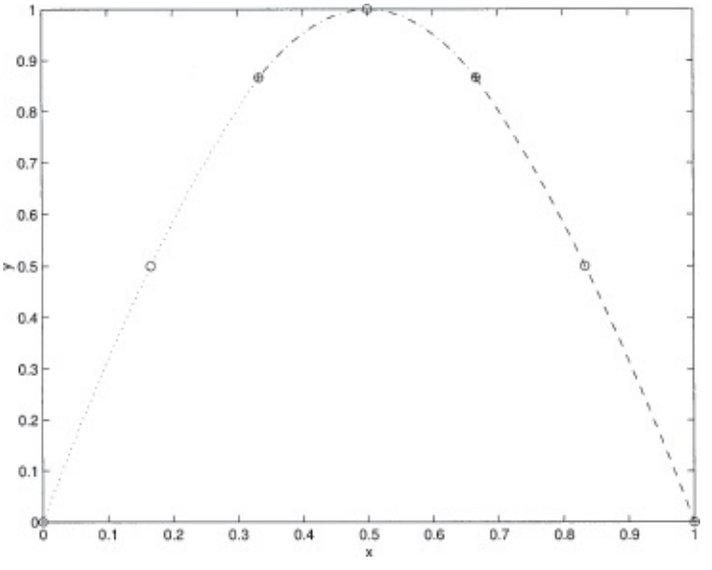
$$\begin{aligned} p_{2,1}(x) &= 3.0000000x - 2.4115427x\left(x - \frac{1}{6}\right), \\ p_{2,2}(x) &= 0.8660254 + 0.8038476\left(x - \frac{1}{3}\right) - 4.8230855\left(x - \frac{1}{3}\right)\left(x - \frac{1}{2}\right), \\ p_{2,3}(x) &= 0.8660254 - 2.1961524\left(x - \frac{2}{3}\right) - 2.4115427\left(x - \frac{2}{3}\right)\left(x - \frac{5}{6}\right), \end{aligned}$$

so that the piecewise polynomial is given by

$$q_2(x) = \begin{cases} 3.0000000x - 2.4115427x(x - \frac{1}{6}), & 0 \le x \le \frac{1}{3}; \\ 0.8660254 + 0.803847576(x - \frac{1}{3}) - 4.8230855(x - \frac{1}{3})(x - \frac{1}{2}), & \frac{1}{3} \le x \le \frac{2}{3}; \\ 0.8660254 - 2.1961524(x - \frac{2}{3}) - 2.4115427(x - \frac{2}{3})(x - \frac{5}{6}), & \frac{2}{3} \le x \le 1. \end{cases}$$

Figure 4.11 shows a plot of this function, using a different line style for each polynomial piece. The nodes and knots are also marked on the graph.

Figure 4.11 Piecewise polynomial approximation to $y = \sin \pi x$ over [0, 1]; a circle ("o") denotes a node, a plus sign ("+") denotes a knot.



It might be argued that approximating the sine function over [0, π] is not much of a challenge, that even simple polynomial interpolation ought to do a decent job of approximating such a function; this is fair criticism, and the student should do the necessary computations to show that Newton interpolation will match the sine function over [0, π] very well. But recall that we were able to match the troublesome function $f(x) = 1/(1 + 25x^2)$ very well with piecewise polynomial approximation. This bodes well for its ultimate utility.

Piecewise polynomial approximation is, in fact, extraordinarily useful as an approximation technique, because the use of different polynomials on different parts of the domain allows us to more closely mimic the range of function behavior that is possible. To accurately model more complicated functions, we need to make sure that the various pieces are joined in a way that maintains as much smoothness as possible. This leads us, very naturally, to s discussion of *splines*.

An algorithm for piecewise polynomial approximation will generally consist of two parts: One that constructs the approximation, usually in the form of an array of polynomial coefficients or divided difference coefficients (which is what we did in Example 4.7 above), and a second part that uses the output of the first part to evaluate the approximation. Pseudo-code for each of these is given in Algorithm 4.3.

Algorithm 4.3 *Pseudo-code for Constructing Piecewise Polynomial Approximation*

```
input n, (x(i), y(i), i=0, n), d
!
    kount = 0
    for k=0 to n by d do
        kount = kount + 1
        for i=0 to d do
            xc(i) = x(k+i)
            yc(i)    = y(k+i)
        endfor
!
!    compute divided difference coefficients
!    for the xc and yc arrays
!
        ac(0) = yc(0)
        for i=1 to d do
            w = 1
            p = 0
            for j=0 to i-1 do
                p = p + ac(j)*w
                w = w*(xc(i) - xc(j))
            endfor
            ac(i) = (yc(i) - p)/w
        endfor
        for i=0 to d do
            a(kount, i) = ac(i)
        endfor
!
!    This code evaluates the pew poly at the point xx
!    First, find the subinterval containing the evaluation
!    point xx
!
    input n, d, kount, (x(i), i=0, n), (a(k, i), k=1, kount, i=0, d), xx
    j = search(x, xx)
!
!    Next, extract the correct nodes and DD coefficients
!
    for k=0 to d
        ac(k) = a(j, k)
        xc(k) = x(j+k)
    endfor
    yy = ac(j, d)
    for k = d-1 to 0 by -1
        xd = xx - xc(k)
        yy = ac(k) + yy*xd
endfor
```

Programming Hints: Note the line `j = search(x, xx)` in the evaluation part of

Algorithm 4.3. Evaluating a piecewise polynomial requires, first of all, that we find out which "piece" of the polynomial to use for each given x value for which we want to compute $q_d(x)$, and this pseudo-code assumes that this is done in the separate routine called `search`. If the mesh spacing is uniform, the computation

$$j = \left\lfloor \frac{x - x_0}{h} \right\rfloor$$

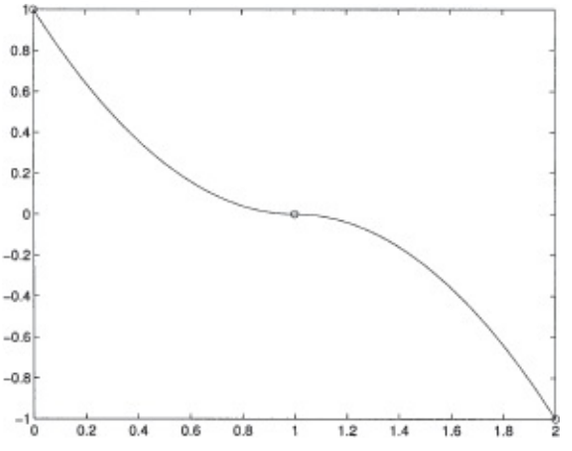
guarantees that $x_j \le x \le x_{j+1}$. Here $\lfloor \cdot \rfloor$ is the *floor function*, defined as the greatest integer less than or equal to the argument. From this it is an easy task to figure out which polynomial piece to use to compute q_d. If the mesh spacing is not uniform, then a more general search procedure has to be used to find the indices such that $x_{(d-1)>k} \le x \le x_{dk}$.

MATLAB has a number of routines for constructing and evaluating piecewise polynomial approximations within a specialized data structure:

- pchip—piecewise cubic Hermite polynomial;
- spline—spline construction (see §4.8);
- ppval—evaluates a piecewise polynomial;
- mkpp—Construct a piecewise polynomial from the knots and the polynomial coefficients.

For example, defining the polynomial coefficient vector pc = [1 −2 1; −1 0 0] and then executing `pp = mkpp([0 1 2], pc)` creates the piecewise quadratic shown in Figure 4.12—the circles mark the knots. (The reader should be aware that MATLAB assumes some shifting in the definitions of the polynomial pieces—the right half of Fig. 4.12 is $y = -(x - 1)^2$, which ordinarily would be rendered in MATLAB as [−1 2 −1], but had to be rendered as [−1 0 0] to achieve the desired effect.)

Figure 4.12 Very simple piecewise polynomial created in MATLAB.



The underlying error theorem is easy to state and prove.

Theorem 4.6 (Piecewise Polynomial Interpolation Error Estimate) *Let f be sufficiently differentiable on the closed interval $[a, b]$, and let q_d be the piecewise polynomial interpolate of degree d to f on $[a, b]$, using $n + 1$ equally spaced nodes x_k, $0 \le k \le n$, $n = md$. Then*

$$\|f - q_d\|_\infty \le C_d h^{d+1} \|f^{(d+1)}\|_\infty,\ d = 1, 2, 3,$$

where.

$$C_1 = 1/8;\ C_2 = 1/9\sqrt{3};\ C_3 = 1/24.$$

Proof: Follows directly from the separate error estimates (4.8), (4.12), and (4.13). We have

$$\begin{aligned}
\|f - q_d\|_\infty &= \max_{a \le x \le b} |f(x) - q_d(x)|, \\
&= \max_{1 \le k \le n} \left(\max_{x_{k-1} \le x \le x_k} |f(x) - q_d(x)| \right), \\
&= \max_{1 \le k \le n} \left(\max_{x_{k-1} \le x \le x_k} |f(x) - p_{d,k}(x)| \right), \\
&= \max_{1 \le k \le n} \|f - p_{d,k}\|_{\infty,[x_{k-1},x_k]}, \\
&\le \max_{1 \le k \le n} C_d h^{d+1} \|f^{(d+1)}\|_{\infty,[x_{k-1},x_k]}, \\
&= C_d h^{d+1} \|f^{(d+1)}\|_{\infty,[a,b]}, \\
&= C_d h^{d+1} \|f^{(d+1)}\|_\infty,
\end{aligned}$$

and we are done. •

While piecewise polynomial approximation is heavily used in a variety of applications (we used it, essentially, to construct the composite trapezoid rule in Chapter 2), it is most important, perhaps, as a precursor to the development of spline approximations, our next main topic.

One interesting and useful form of piecewise polynomial approximation is the use of piecewise cubic Hermite interpolation. Here, we use cubic Hermite polynomials in an obvious way to define the polynomial approximation between each pair of nodes x_{k-1} and x_k. Because we are also interpolating to the derivative values, the resulting approximation is much smoother than ordinary piecewise interpolation and is less prone to having "kinks" in the graph. Some of the exercises ask the student to look at this type of approximation.

Exercises:

1. Use divided difference tables to construct the separate parts of the piecewise quadratic polynomial $q_2(x)$ that interpolates to $f(x) = \cos \frac{1}{2}\pi x$ at $x = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$. Plot the approximation and the error $\cos \frac{1}{2}\pi x - q_2(x)$.

2. Repeat the above using $f(x) = \sqrt{x}$ with the nodes $x = \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1$.

3. Confirm that (4.30) is the correct piecewise quadratic approximation to $f(x) = 1/(1 + 25x^2)$ using the nodes $x_0 = -1$, $x_1 = -2/3$, $x_2 = -1/3$, $x_3 = 0$, $x_4 = 1/3$, $x_5 = 2/3$, and $x_6 = 1$.

4. Using the data in Table 4.8, construct a piecewise cubic interpolating polynomial to the gamma function, using the nodes $x = 1.0, 1.2, 1.3, 1.5$ for one piece, and the nodes $x = 1.5, 1.7, 1.8, 2.0$ for the other piece. Use this approximation to estimate $\Gamma(x)$ for $x = 1.1, 1.4, 1.6$ and 1.9. How accurate is the approximation?

5. Using the results of Problem 6 from §4.6, together with the data of function values from Problem 9 of §4.2, construct a piecewise cubic Hermite interpolating polynomial for the gamma function, using the nodes $x = 1.0, 1.3, 1.7, 2.0$. Test the accuracy of the interpolation by using it to approximate $\Gamma(x)$ for $x = 1.1, 1.2, 1.4, 1.5, 1.6, 1.8, 1.9$.

6. Construct a piecewise cubic interpolating polynomial to $f(x) = \ln x$ on the interval $[\frac{1}{2}, 1]$, using the nodes

$$x_k = \frac{1}{2} + \frac{k}{18}, \quad 0 \le k \le 9.$$

Compute the value of the error $\ln x - p(x)$ at 500 equally spaced points on the interval $[\frac{1}{2}, 1]$, and plot the error. What is the maximum sampled error?

7. Repeat the above, using piecewise cubic Hermite interpolation over the same grid.

8. Construct piecewise polynomial approximations of degree $d = 1, 2, 3$ to the data in Table 4.12, using only the nodes $\log_{10} \theta_k = -6, -4, -2, 0, 2, 4, 6$. Plot the resulting curve and compare it to the ordinary interpolating polynomial found in Problem 13 of §4.2. Test how well this approximation matches the tabulated values at $\log_{10} \theta = -5, -3, -1, 1, 3, 5$.

9. Show that the error in piecewise cubic Hermite interpolation satisfies

$$\|f - H_2\|_\infty \le \frac{1}{384} h^4 \|f^{(4)}\|_\infty,$$

where we have assumed uniformly spaced nodes with $x_k - x_{k-1} = h$.

10. Given a grid of points

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b,$$

define the piecewise linear functions ${\varphi_k}^h$, $1 \le k \le n - 1$, according to

$$\phi_k^h(x) = \begin{cases} \frac{x - x_{k-1}}{x_k - x_{k-1}}, & x_{k-1} \le x \le x_k; \\ \frac{x_{k+1} - x}{x_{k+1} - x_k}, & x_k \le x \le x_{k+1}; \\ 0, & \text{otherwise.} \end{cases}$$

Define the function space

$$S_0^h = \{f \in C([a, b]), f(a) = f(b) = 0, f \text{is piecewise linear on the given grid}\}.$$

Show that the ${\phi_k}^h$ are a basis for ${S_0}^h$, i.e., that every element of the space ${S_0}^h$ can be written as a linear combination of the ${\phi_k}^h$ functions.

11. Implement a routine for approximating the natural logarithm using piecewise polynomial interpolation, i.e., a table look-up scheme. Assume that the table of (known) logarithm values is uniformly distributed on the interval $[\frac{1}{2}, 1]$. Choose enough points in the table to guarantee 10^{-10} accuracy for any x. Use:

(a) Piecewise linear interpolation;

(b) Piecewise cubic Hermite interpolation.

Test your routine against the intrinsic logarithm function on your computer by evaluating the error in your approximation at 5,000 equally spaced points on the interval $[\frac{1}{10}, 10]$. Use the way that the computer stores floating-point numbers to reduce

the logarithm computation to the interval $[\frac{1}{2}, 1]$, as long as ln 2 is known to high accuracy.

◁●●●▷

4.8 AN INTRODUCTION TO SPLINES

Piecewise polynomial approximation (§4.7) allows us to construct highly accurate approximations, but it does not always produce approximations that are pleasing to the eye, and this is actually a problem in some applications. This occurs because the approximating function is not smooth at the junctions between separate pieces of the piecewise polynomial approximation. Although we have that q_d is continuous, it is *not* continuously differentiable on the entire interval of approximation. The graph of the interpolate can have "kinks" in it, and some of these are apparent in Figure 4.8.

Splines are an attempt to address this problem. The basic idea is to construct a piecewise polynomial approximation that not only interpolates given data or function values, but which also is "smooth," meaning continuously differentiable to some degree.

A complete theory of splines is beyond the scope of this book. Interested readers are referred to the book by Carl de Boor, *A Practical Guide to Splines* [8]. Another good reference is Paddy Prenter's *Splines and Variational Methods* [14]. We will outline two different spline constructions here and state some of the more basic theoretical results.[11]

4.8.1 Definition of the Problem

We first have to properly define the problem before we can solve it. Suppose that we are given a set of *nodes* $\{x_i, 0 \le i \le n\}$ at which we wish to interpolate a given function $f(x)$ with a spline of degree d. The problem is then to find a piecewise polynomial function, q_d, which satisfies the following conditions:

1. *Interpolation:*

$$q_d(x_k) = f(x_k), \quad 0 \le k \le n.$$

Here d is called the *degree of approximation* of the spline.

2. *Smoothness:*

$$\lim_{x \to x_k^-} q_d^{(i)}(x) = \lim_{x \to x_k^+} q_d^{(i)}(x)$$

for $0 \le i < N$. We call N the *degree of smoothness* of the spline.

3. *Interval of definition:* q_d is a polynomial of degree $\le d$ on each subinterval $[x_{k-1}, x_k]$.

We need to investigate the relationship (if any) between the degree of approximation, d, and the degree of smoothness, N. The degree of the polynomials is related to the number of unknown coefficients, i.e., the degrees of freedom, in the problem; whereas N is related to the number of constraints. We expect that the degrees of freedom and the number of

constraints have to balance in order for the spline to be well-defined.

Since there are n subintervals, each being the domain of definition for a separate polynomial of degree d, we have a total of $K_f = n(d + 1)$ degrees of freedom. On the other hand, there are $n + 1$ interpolation conditions and $n - 1$ junction points (again called *knots*) between subintervals (note that in the spline construction, neither x_0 nor x_n are considered knots) with $N + 1$ continuity conditions being imposed at each of them. Thus, there are $K_c = n + 1 + (n - 1)(N + 1)$ constraints. If we consider the difference $K_f - K_c$, we get

$$K_f - K_c = n(d+1) - [n+1+(n-1)(N+1)] = nd - n - nN + N = n(d-1-N) + N.$$

We can make the first term vanish by setting $d - 1 - N = 0$. This establishes a relationship between the polynomial degree of the spline and the smoothness degree. For example, if we consider the common case of cubic splines, then $d = 3$ and $N = 2$. However, we will not have the number of constraints equal the number of degrees of freedom: $K_f - K_d = N$. Thus, we need to add N additional constraints. Partly for this reason, odd polynomial order splines are preferred, because if d is odd then $N = d - 1$ is even and the additional constraints can be imposed equally at the two endpoints (which is the typical choice) of the interval.

It is worth noting, briefly, that we can consider a continuous piecewise linear function to be a spline with degree of smoothness $N = 0$. See Problem 18.

4.8.2 Cubic B-Splines

The construction we use here is based on the *B-spline*. This idea uses a single exemplar function from which a basis of splines is formed, and the approximation is then defined in terms of this basis. The notion of B-splines is much more general than is presented here, where we restrict ourselves to the (widely used) case of cubic B-splines, with coincident nodes and knots. For simplicity's sake, we assume at first a uniform grid

(4.31) $$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$$

with $x_k - x_{k-1} = h$ for all $k \geq 1$. We will also need to define the extra grid points

(4.32) $$x_{-3} = a - 3h, \quad x_{-2} = a - 2h, \quad x_{-1} = a - h,$$

and

(4.33) $$x_{n+1} = b + h, \quad x_{n+2} = b + 2h, \quad x_{n+3} = b + 3h.$$

Then define the function

(4.34) $$B(x) = \begin{cases} 0, & x \leq -2; \\ (x+2)^3, & -2 \leq x \leq -1; \\ 1 + 3(x+1) + 3(x+1)^2 - 3(x+1)^3, & -1 \leq x \leq 0; \\ 1 + 3(1-x) + 3(1-x)^2 - 3(1-x)^3, & 0 \leq x \leq 1; \\ (2-x)^3, & 1 \leq x \leq 2; \\ 0, & 2 \leq x; \end{cases}$$

and note that B is a cubic spline with nodes/knots at the points $x = -2, -1, 0, 1, 2$.

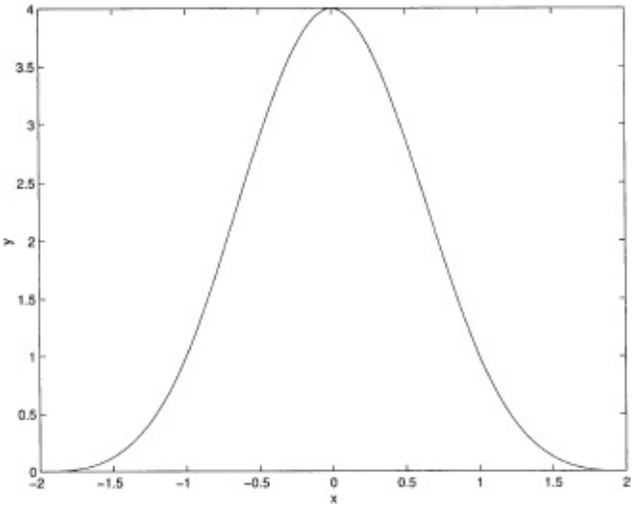
Wait a minute. How do we know that $B(x)$ is a cubic spline function? Well, a spline function is a piecewise polynomial with a certain amount of derivative continuity between the pieces, the amount of derivative continuity being related to the polynomial degree by $N = d - 1$, as we showed above. Thus, to check that $B(x)$ is a cubic spline, we simply note that it clearly is a piecewise cubic polynomial, and then we compute the one-sided derivatives at the knots:

$$B'_-(x_k) = \lim_{x \to x_k^-} B'(x), \quad B'_+(x_k) = \lim_{x \to x_k^+} B'(x),$$

and similarly for the second derivative (and the function value). If the one-sided values are equal to each other, then the first and second derivatives are continuous, hence B is a cubic spline. If only the first derivative was continuous, it would fail to be a spline because *cubic* splines require second derivative continuity.

Figure 4.13 shows a graph of B; note the bell shape of the curve. Note also that

Figure 4.13 Original B-spline.



$$B(0) = 4, \quad B(\pm 1) = 1, \quad B(\pm 2) = 0, \tag{4.35}$$

$$B'(0) = 0, \quad B'(\pm 1) = \mp 3, \quad B'(\pm 2) = 0, \tag{4.36}$$

$$B''(0) = -12, \quad B''(\pm 1) = 6, \quad B''(\pm 2) = 0. \tag{4.37}$$

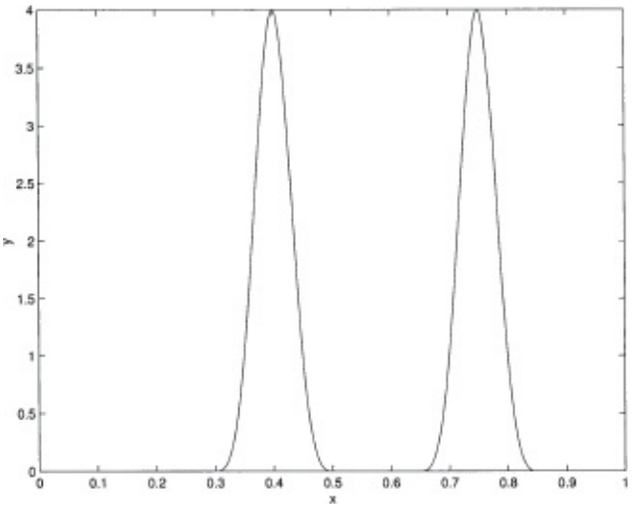
Finally, note that B is only "locally defined," meaning that it is non-zero on only a small interval.[12] This local definition property is important in the utility of B-splines as an approximation tool.

We can use B to construct a spline approximation to an arbitrary function f using the grid defined in (4.31), (4.32), and (4.33). Define the sequence of functions

$$B_i(x) = B((x - x_i)/h), \quad -1 \le i \le n + 1. \tag{4.38}$$

Each B_i is similar in shape to B, but is centered at x_i instead of at 0. Figure 4.14 shows plots of some of the B_i for a specific grid. Note that

Figure 4.14 B-splines on a grid.



(4.39) $$B_i(x_i) = B(0), \quad B_i(x_{i\pm1}) = B(\pm1), \quad B_i(x_{i\pm2}) = B(\pm2),$$

and similarly for the derivatives:

(4.40) $$B_i'(x_i) = 0, \quad B_i'(x_{i\pm1}) = \mp 3/h, \quad B_i'(x_{i\pm2}) = 0,$$

(4.41) $$B_i''(x_i) = -12/h^2, \quad B_i''(x_{i\pm1}) = 6/h^2, \quad B_i''(x_{i\pm2}) = 0.$$

To construct a spline interpolant to a given function *f*, we define the spline as a linear combination of the B_i functions, $-1 \le i \le n+1$:

$$q_3(x) = \sum_{i=-1}^{n+1} c_i B_i(x) \tag{4.42}$$

and seek the coefficients c_i, $-1 \le i \le n+1$, such that

$$f(x_k) = \sum_{i=-1}^{n+1} c_i B_i(x_k), \quad 0 \le k \le n. \tag{4.43}$$

It is at this point that the local definition of *B* becomes so useful. If we look at (4.43) for $k = 0$, we see that we have

$$f(x_0) = c_{-1}B_{-1}(x_0) + c_0 B_0(x_0) + c_1 B_1(x_0)$$

since $B_i(x_0) = 0$ for all $i \ge 2$. In general, then, we have that

$$f(x_k) = c_{k-1}B_{k-1}(x_k) + c_k B_k(x_k) + c_{k+1}B_{k+1}(x_k).$$

Thus, each equation in the system defined by (4.43) has only three non-zero terms—it is a *tridiagonal* system, just the kind we learned how to solve back in §2.6. Moreover, we can use (4.35) to show that (4.43) simplifies to

$$f(x_k) = c_{k-1} + 4c_k + c_{k+1}.$$

Written in matrix-vector form, the system is

$$\begin{bmatrix} 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} c_{-1} \\ c_0 \\ \vdots \\ c_{n+1} \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \tag{4.44}$$

Of course, this is still a system of $n+1$ equations in $n+3$ unknowns; we need to come up with two additional constraints in order to eliminate two of the unknowns. Two common choices are:

1. *The Natural Spline*. Here we impose $q''_3(x_0) = q''_3(x_n) = 0$. This leads to a very simple construction, but also leads to higher error near the endpoints.

2. *The Complete Spline*. Here we impose $q'_3(x_0) = f'(x_0)$ and $q'_3(x_n) = f'(x_n)$. This leads to better approximation properties, and does not actually require that we know the derivative at the endpoints, since we can use the function values to approximate it via one of the formulas from §4.5.

The Natural Spline We can directly compute, using (4.42) and (4.36), that we have

$$q_3''(x_0) = h^{-2}(6(c_{-1} + c_1) - 12c_0), \quad q_3''(x_n) = h^{-2}(6(c_{n+1} + c_{n-1}) - 12c_n).$$

We therefore achieve $q''_3(x_0) = q''_3(x_n) = 0$ simply by imposing

(4.45) $$c_{-1} + c_1 = 2c_0 \Rightarrow c_{-1} = 2c_0 - c_1$$

and

(4.46) $$c_{n+1} + c_{n-1} = 2c_n \Rightarrow c_{n+1} = 2c_n - c_{n-1}.$$

If we substitute these into the system (4.44), we find that the first and last equations simplify to

(4.47) $$6c_0 = f(x_0), \quad 6c_n = f(x_n),$$

so that we have the new system

(4.48)
$$\begin{array}{ccccccc} 4c_1 & + & c_2 & & & = & f(x_1) - \frac{1}{6}f(x_0), \\ c_1 & + & 4c_2 & + & c_3 & = & f(x_2), \\ & \ddots & & \ddots & & \vdots & \\ & & c_{n-2} & + & 4c_{n-1} & = & f(x_{n-1}) - \frac{1}{6}f(x_n). \end{array}$$

Not only is this now square, in addition to being tridiagonal, but it is also diagonally dominant. Therefore, as discussed in §2.6, the matrix is nonsingular and the solution algorithm of §2.6 can be applied to compute the coefficients.

The Complete Spline Here we impose derivative data at the endpoints. Direct calculation shows that

$$q_3'(x_0) = -3h^{-1}(c_{-1} - c_1), \quad q_3'(x_n) = -3h^{-1}(c_{n-1} - c_{n+1}),$$

so that the conditions $q'_3(x_0) = f'(x_0)$ and $q'_3(x_n) = f'(x_n)$ imply that

(4.49) $$c_{-1} = c_1 - \frac{1}{3}hf'(x_0), \quad c_{n+1} = c_{n-1} + \frac{1}{3}hf'(x_n).$$

When substituted into the system (4.44) we again lose the dependence on c_{-1} and c_{n+1} and get the square system

(4.50)
$$\begin{array}{ccccccc} 4c_0 & + & 2c_1 & \cdots & & = & f(x_0) + \frac{1}{3}hf'(x_0), \\ \ddots & & \ddots & & \ddots & \vdots & \\ & c_{k-1} & + & 4c_k & + & c_{k+1} = & f(x_k), \\ & & \ddots & & \ddots & \vdots & \\ & & & 2c_{n-1} & + & 4c_n = & f(x_n) - \frac{1}{3}hf'(x_n), \end{array}$$

which is (again) diagonally dominant, so the tridiagonal solution algorithm can be applied. Note that in the natural spline case, the system had $n - 1$ equations in the $n - 1$ unknowns c_i, $1 \le i \le n - 1$, whereas here we have a system of $n + 1$ equations in the $n + 1$ unknowns

c_i, $0 \le i \le n$.

Evaluation The natural or complete spline can be easily constructed, in the sense that the coefficients c_k are defined, by solving (4.48) or (4.50). How do we use these values to evaluate the spline q_3, defined by (4.42)?

For any $x \in [x_{k-1}, x_k]$, the fact that each B_i is only locally non-zero means that we have

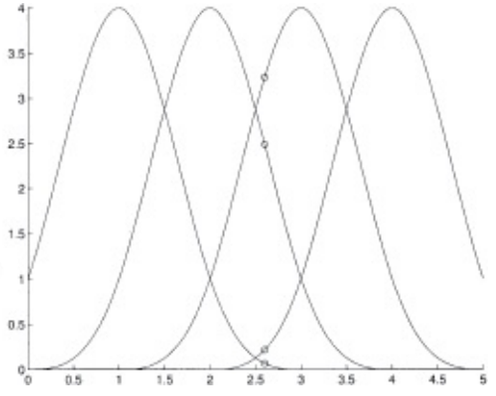
(4.51) $$q_3(x) = c_{k-2}B_{k-2}(x) + c_{k-1}B_{k-1}(x) + c_kB_k(x) + c_{k+1}B_{k+1}(x),$$

so evaluation of $q_3(x)$ requires only that we find the index k such that $x_{k-1} \le x \le x_k$. For a uniform grid this is easily accomplished:

$$k = \left\lfloor \frac{x - x_0}{h} \right\rfloor + 1.$$

For a nonuniform grid a more general search routine would have to be employed. Fig. 4.15 illustrates this by graphing B-splines centered at x = 1, 2, 3, 4 and marking the image of x = 2.6 on each of the four curves.

Figure 4.15 Demonstration of local definition of B-splines.



Programming Hint: Note that (4.51) implies that

$$q_3(x_k) = f(x_k) = c_{k-1}B_{k-1}(x_k) + c_kB_k(x_k) + c_{k+1}B_{k+1}(x_k) = c_{k-1} + 4c_k + c_{k+1}.$$

This can be a useful check on the correctness of the spline coefficients.

■ EXAMPLE 4.8

We start with a simple example that can be carried out by hand. Suppose that we want to compute a cubic spline approximation to $f(x) = \sin \pi x$ on the interval $[0, \frac{1}{2}]$, using the nodes $x_0 = 0$, $x_1 = \frac{1}{4}$, and $x_2 = \frac{1}{2}$. Thus, $n - 2$ and the spline will be defined by

$$q_3(x) = \sum_{k=-1}^{3} c_kB_k(x).$$

For the natural spline, (4.47) implies that

$$c_0 = \frac{1}{6}\sin 0 = 0$$

and

$$c_2 = \frac{1}{6}\sin\frac{\pi}{2} = \frac{1}{6},$$

and instead of a system of equations to solve, we have the single equation

$$c_0 + 4c_1 + c_2 = f(x_1) \Rightarrow c_1 = \frac{1}{4}\left(f(x_1) - \frac{1}{6}f(x_0) - \frac{1}{6}f(x_2)\right),$$

from which we get

$$c_1 = \frac{1}{4}\left(\frac{\sqrt{2}}{2} - \frac{1}{6}\right) = 0.1351100286.$$

Finally, then, (4.45) and (4.46) imply that

$$c_{-1} = \frac{1}{4}\left(\frac{1}{6} - \frac{\sqrt{2}}{2}\right) = -0.1351100286, \quad c_3 = \frac{1}{3} - \frac{\sqrt{2}}{8} + \frac{1}{24} = 0.1982233048,$$

so the spline approximation is now given by

$$q_3(x) = c_{-1}B_{-1}(x) + c_1B_1(x) + c_2B_2(x) + c_3B_3(x).$$

To compute values of the spline is fairly simple. Suppose that we want to find $q_3(1/6) \approx \sin \pi/6 = 1/2$. We first determine that $x = 1/6$ is between $x_0 = 0$ and $x_1 = \frac{1}{4}$; this means that we want to compute

$$q_3(1/6) = c_{-1}B_{-1}(1/6) + c_1B_1(1/6) + c_2B_2(1/6);$$

the B_3 term vanishes because B_3 is identically zero on the subinterval $[x_0, x_1]$. We then have that

$$\begin{aligned} q_3(1/6) &= c_{-1}B_{-1}(1/6) + c_1B_1(1/6) + c_2B_2(1/6) \\ &= = c_{-1}B(5/3) + c_1B(-1/3) + c_2B(-4/3) \\ &= (-0.1351100286)(0.0370370370) + (0.1351100286)(3.4444444444) \\ &\quad + (0.1666666667)(0.2962962967) \\ &= 0.5097576284, \end{aligned}$$

which is a decent approximation to the exact value. The entire spline is graphed in Figure 4.16. The error is given in Figure 4.17.

Figure 4.16 Natural spline approximation to part of the sine function.

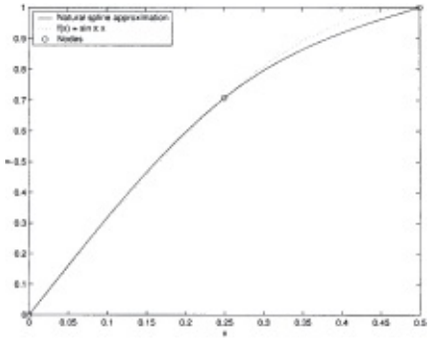
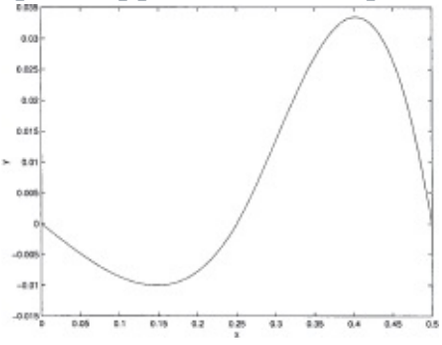


Figure 4.17 Error in natural spline approximation to part of the sine function.



Recall, however, that using a natural spline imposes the condition $q''_3(x_0) = q''_3(x_n) = 0$, which might not be the correct value to use (and, in the case of our example, isn't even a good approximation to the correct value at $x_2 - b = \frac{1}{2}$). If we construct the complete spline approximation, then we are led, via (4.49) and (4.50), to the 3×3 linear system

$$\begin{bmatrix} 4 & 2 & 0 \\ 1 & 4 & 1 \\ 0 & 2 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{12}\pi \\ \frac{1}{2}\sqrt{2} \\ 1 \end{bmatrix},$$

which we can easily solve to get

$$c_0 = 0.00017369124366, \quad c_1 = 0.13055231141225, \quad c_2 = 0.18472384429387.$$

Thus, the complete set of coefficients is

$$c_{-1} = -0.13124707638690, \quad c_0 = 0.00017369124366, \quad c_1 = 0.13055231141225,$$
$$c_2 = 0.18472384429387, \quad c_3 = 0.13055231141225,$$

and the spline is given by

$$q_3(x) = c_{-1}B_{-1}(x) + c_0B_0(x) + c_1B_1(x) + c_2B_2(x) + c_3B_3(x).$$

The spline itself is graphed in Figure 4.18 and the error is in Figure 4.19. Note that the approximation is much more accurate than was the case for the natural spline. This is because the natural spline imposed the inaccurate condition $q''_3(x) = 0$ at $x = \frac{1}{2}$, whereas the complete spline used the correct values of the first derivative at both endpoints. Just as a single point of comparison, this time we get

Figure 4.18 Complete spline approximation to part of the sine function.

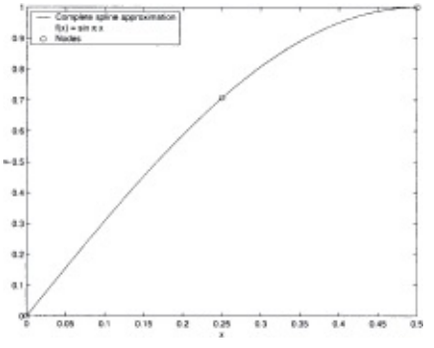
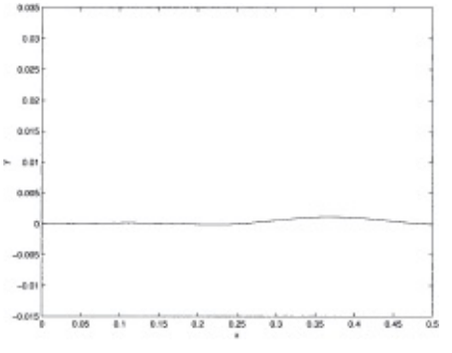


Figure 4.19 Error in complete spline approximation to part of the sine function (same vertical scale as in Figure 4.17).



$$q_3(1/6) = 0.4999381524,$$

which is a much better approximation to the correct value of $\frac{1}{2}$ than we got for the

natural spline.

■ EXAMPLE 4.9

Consider now the function $f(x) = (1 + 25x^2)^{-1}$, which we have had trouble approximating with ordinary interpolation. We can construct both natural and complete spline approximations to this function, using equally spaced points on the interval $[-1, 1]$, with $h^{-1} = 2, 4, 8, 16$. Figures 4.20 and 4.21 show the plots of the natural spline and associated error, while Figures 4.22 and 4.23 do the same for the complete spline and its error. Compare these plots to the simple piecewise polynomial fits shown in Figures 4.8 to 4.10.

Figure 4.20 Natural Spline Interpolation to $f(x) = (1 + 25x^2)^{-1}$. The nodes are marked by plus signs.

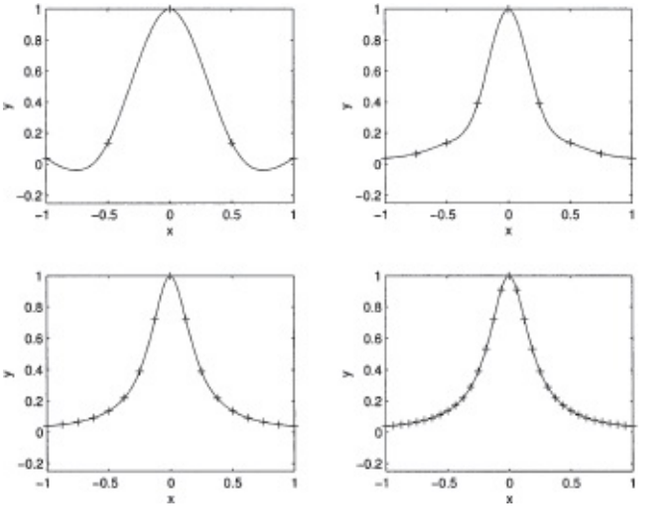


Figure 4.21 Error in Natural Spline Interpolation to $f(x) = (1 + 25x^2)^{-1}$.

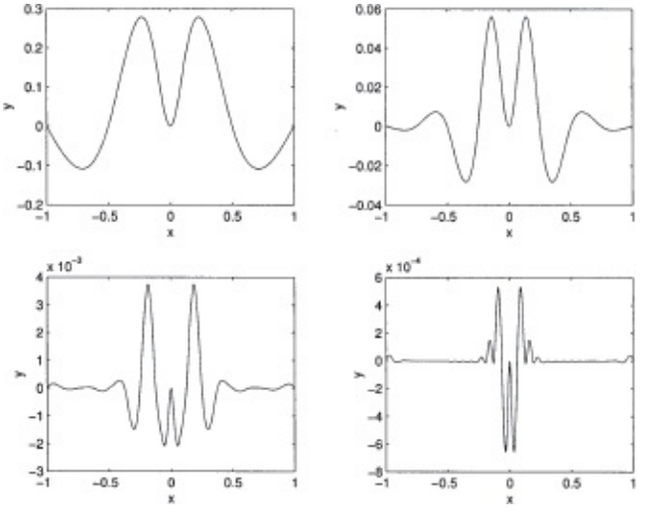


Figure 4.22 Complete Spline Interpolation to $f(x) = (1 + 25x^2)^{-1}$. The nodes are marked by plus signs.

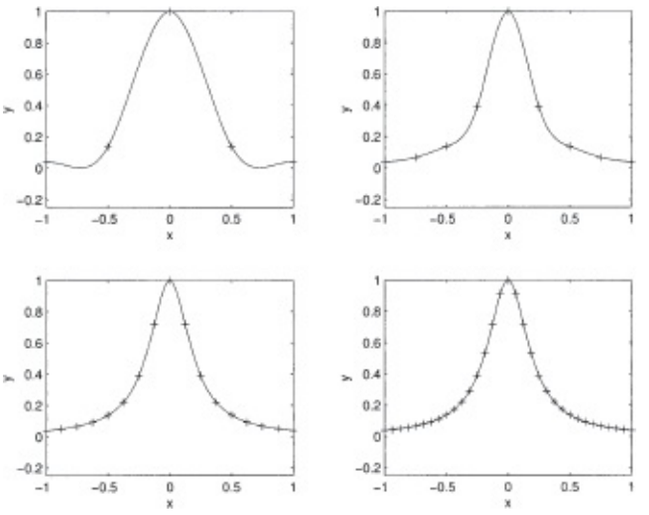
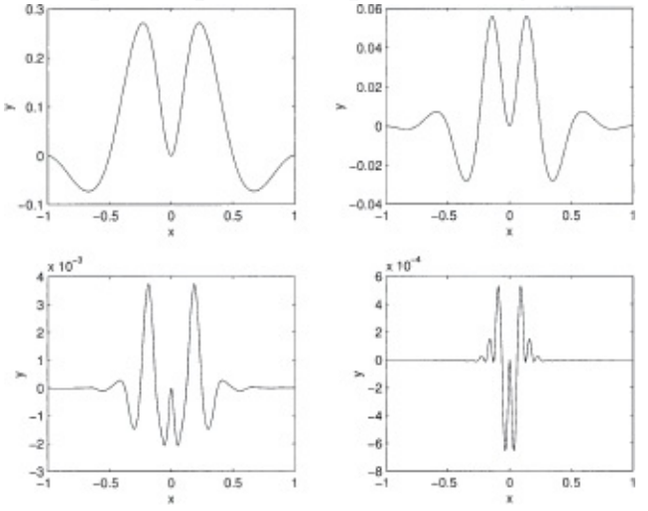


Figure 4.23 Error in Complete Spline Interpolation to $f(x) = (1 + 25x^2)^{-1}$.



Theoretical Results The error theory for spline approximations is somewhat more involved than that for ordinary piecewise polynomial interpolation, so we will here refer to the main result without proof and then comment on it. See Hall's article [10] for the details, or the article by de Boor [7].

Theorem 4.7 (Spline Approximation) *If* $f \in C^4([a, b])$ *and* q *is a cubic spline interpolant to* f *on a grid*

$$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$$

with $\max(x_i - x_{i-1}) \leq h$ *for all* i*, then*

$$\|f - q\|_\infty \leq \frac{5}{384} h^4 \|f^{(4)}\|_\infty.$$

Moreover, there exist constants C_k, $1 \leq k \leq 3$, *such that*

$$\|f^{(k)} - q^{(k)}\|_\infty \leq C_k h^{4-k} \|f^{(4)}\|_\infty.$$

Note that the spline interpolant is no more accurate, in terms of the exponent on the mesh size, than ordinary piecewise polynomial approximation (note that the constant in the estimate is smaller). The advantage of spline interpolation lies in the smoothness of the approximation.

Exercises:

1. Given the set of nodes $x_0 = 0$, $x_1 = 1/2$, $x_2 = 1$, $x_3 = 3/2$, and $x_4 = 2$, we construct the cubic spline function

$$\begin{aligned} q_3(x) &= 0.15B_{-1}(x) + 0.17B_0(x) + 0.18B_1(x) + 0.22B_2(x) + 0.30B_3(x) \\ &+ 0.31B_4(x) + 0.32B_5(x), \end{aligned}$$

where each B_k is computed from the exemplar B spline according to (4.38). Compute q_3 and its first derivative at each node.

2. Is the function

$$p(x) = \begin{cases} 0, & x \le 0; \\ x^2, & 0 \le x \le 1; \\ -2x^2 + 6x + 3, & 1 \le x \le 2; \\ (x-3)^2, & 2 \le x \le 3; \\ 0, & x \ge 3 \end{cases}$$

a spline function? Why or why not?

3. For what value of k is the following a spline function?

$$q(x) = \begin{cases} kx^2 + (3/2), & 0 \le x \le 1; \\ x^2 + x + (1/2), & 1 \le x \le 2. \end{cases}$$

4. For what value of k is the following a spline function?

$$q(x) = \begin{cases} x^3 - x^2 + kx + 1, & 0 \le x \le 1; \\ -x^3 + (k+2)x^2 - kx + 3 & 1, \le x \le 2. \end{cases}$$

5. For what value of k is the following a spline function?

$$q(x) = \begin{cases} x^3 + 3x^2 + 1, & -1 \le x \le 0; \\ -x^3 + kx^2 + 1, & 0 \le x \le 1. \end{cases}$$

6. Construct the natural cubic spline that interpolates to $f(x) = 1/x$ at the nodes 1/2, 5/8, 3/4, 7/8, 1. Do this as a hand calculation. Plot your spline function and f on the same set of axes, and also plot the error.

7. Repeat the above using the complete spline.

8. Construct a natural spline interpolate to the mercury thermal conductivity data (Table 4.7), using the 300 K, 500 K, and 700 K values. How well does this predict the values at 400 K and 600 K?

9. Confirm that the function $B(x)$, defined in (4.34), is a cubic spline.

10. Construct a natural cubic spline to the gamma function, using the data in Table 4.8, and the nodes $x = 1.0$, 1.2, 1.4, 1.6, 1.8, and 2.0. Use this approximation to estimate $\Gamma(x)$ at $x = 1.1$, 1.3, 1.5, 1.7, and 1.9.

11. Repeat the above using the complete spline approximation, and use the derivative approximations from §4.5 for the required derivative endpoint values.

12. Repeat Problem 6 of §4.7, but this time construct the complete cubic spline interpolate to $f(x) =$ in x, using the same set of nodes. Plot the approximation and the error. What is the maximum sampled error in this case?

13. Recall Problem 7 from §4.2, in which we constructed polynomial interpolates

to timing data from the 1973 Kentucky Derby, won by the horse Secretariat. For simplicity, we repeat the data in Table 4.19. Here t is the elapsed time (in seconds) since the race began and x is the distance (in miles) that Secretariat has traveled.

Table 4.19 Data for Problem 13.

x	0.0	0.25	0.50	0.75	1.00	1.25
t	0.0	25.0	49.4	73.0	96.4	119.4

(a) Construct a natural cubic spline that interpolates this data.

(b) What is Secretariat's speed at each quarter-mile increment of the race? (Use miles per hour as your units.)

(c) What is Secretariat's "initial speed," according to this model? Does this make sense?

Note: It is possible to do this exercise using a uniform grid. Construct the spline that interpolates t as a function of x, then use your knowledge of calculus to find $x'(t)$ from $t'(x)$.

14. Show that the complete cubic spline problem can be solved (approximately) without having an explicit expression for f' at the endpoints. *Hint:* Consider the material from §4.5.

15. Construct the natural cubic spline that interpolates the data in Table 4.12 at the nodes defined by $\log_{10}\theta = -6, -4, \ldots, 4, 6$. Test the accuracy of the approximation by computing $q_3(x)$ for $x = \log_{10}\theta = -5, -3, \ldots, 3, 5$ and comparing the results to the actual data in the table.

16. Construct the exemplar quadratic B-spline; that is, construct a piecewise quadratic function that is C^1 over the nodes/knots $x = -1, 0, 1, 2$, and which vanishes for x outside the interval $[-1, 2]$.

17. Construct the exemplar *quintic* B-spline.

18. For a linear spline function we have $d = 1$, which forces $N = 0$. Thus, a linear spline has no derivative continuity, only function continuity, and no additional conditions are required at the endpoints. Show that the exemplar B-spline of first degree is given by

$$B(x) = \begin{cases} 0, & x \le -1; \\ x+1, & -1 \le x \le 0; \\ 1-x, & 0 \le x \le 1; \\ 0, & 1 \le x. \end{cases}$$

.

Describe, in your own words, how to construct and evaluate a linear spline interpolant using this function as the basic B-spline.

19. Discuss, in your own words, the advantages or disadvantages of spline approximation compared to ordinary piecewise polynomial approximation.

20. Write an essay that compares and contrasts piecewise cubic Hermite

interpolation with cubic spline interpolation.

21. The data in Table 4.20 gives the actual thermal conductivity data for the element nickel. Construct a natural spline interpolate to this data, using only the data at 200 K, 400 K, …, 1400 K. How well does this spline predict the values at 300 K, 500 K, and so on?

Table 4.20 Data for Problem 21.

Temperature (K), u	200	300	400	500	600
Conductivity (watts/cm K), k	1.06	0.94	0.905	0.801	0.721
Temperature (K), u		700	800	900	1000
Conductivity (watts/cm K), k		0.655	0.653	0.674	0.696

22. Construct a natural spline interpolate to the thermal conductivity data in Table 4.21, below. Plot the spline and the nodal data.

Table 4.21 Data for Problem 22

Temperature K), u	200	300	400	500	600	700
Conductivity (watts/cm K), k	0.94	0.803	0.694	0.613	0.547	0.487

4.9 APPLICATION: SOLUTION OF BOUNDARY VALUE PROBLEMS

The range of application of spline approximations is incredibly wide, including such diverse things as scalable computer printer fonts, which are typically stored as spline approximations; and modern computer special effects in movies, which are often done by using spline approximations to complicated surfaces which can then be easily manipulated and moved about on the screen by the digital special effects artist. But some of the oldest and most important applications of spline functions are to the accurate solution of differential equations. In §2.7 we briefly touched on the approximate solution of boundary value problems using finite difference approximations. In this section, we will build on that experience by using splines to construct our approximation.

Consider the two-point boundary value problem

$$\begin{aligned} -u'' + a^2 u &= f(x), \quad 0 \le x \le 1, \\ u(0) &= g_0, \\ u(1) &= g_1, \end{aligned}$$

which is slightly more general than the one we looked at in §2.7. We construct the grid of points

$$0 = x_0 < x_1 < x_2 < \cdots x_{n-1} < x_n = 1,$$

where we assume (for simplicity, only) that the grid spacing is uniform; i.e., $x_k - x_{k-1} = h$ for all k, $1 \le k \le n$. We now look for our approximation in the form of a cubic spline defined on this grid. That is, we consider the function

$$u_h(x) = \sum_{k=-1}^{n+1} c_k B_k(x),$$

where the coefficients c_k are yet to be determined. The advantage of this approach over what we did in §2.7 is that here we get a continuous *smooth* function as our approximation, whereas in §2.7 all we got was a set of discrete points. We find the coefficients by requiring that u_h satisfy the differential equation at each of the nodes x_k, $0 \le k \le n$. (Note that the smoothness of the spline function is essential for this to even make sense. We have to have $u_h \in C^2$ to even talk about it satisfying the ODE.) Because we know the values of B_k and its derivatives at each of the nodes, we can easily reduce this to the system of equations

(4.52) $T'c' = b',$

where

$$T' = \begin{bmatrix} \alpha & \beta & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & \beta & \alpha & \vdots & 0 \\ \vdots & \ddots & \ddots & 0 & \ddots & \\ 0 & \cdots & 0 & \alpha & \beta & \alpha \end{bmatrix}$$

for

$$\alpha = -6h^{-2} + a^2, \quad \beta = 12h^{-2} + 4a^2,$$

and

$$c' = \begin{bmatrix} c_{-1} \\ c_0 \\ \vdots \\ c_{n+1} \end{bmatrix}, \quad b' = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

Since this is a system of only $n + 1$ equations in $n + 3$ unknowns, we can't construct a solution yet. We can eliminate the two extra unknowns by imposing the boundary conditions on the approximation:

(4.53) $u_h(0) = g_0 \Rightarrow c_{-1} + 4c_0 + c_1 = g_0 \Rightarrow c_{-1} = g_0 - 4c_0 - c_1$

and

(4.54) $u_h(1) = g_1 \Rightarrow c_{n-1} + 4c_n + c_{n+1} = g_1 \Rightarrow c_{n+1} = g_1 - 4c_n - c_{n-1}.$

If we substitute these into the first and last equations of the rectangular system, then we get

$$(-6h^{-2} + a^2)(g_0 - 4c_0 - c_1) + (12h^{-2} + 4a^2)c_0 + (-6h^{-2} + a^2)c_1 = f(x_0)$$

and

$$(-6h^{-2} + a^2)c_{n-1} + (12h^{-2} + 4a^2)c_n + (-6h^{-2} + a^2)(g_1 - 4c_n - c_{n-1}) = f(x_n).$$

The c_1 and c_{n-1} terms drop out here, so we can conclude that

(4.55) $$c_0 = \frac{h^2}{36}\left(f(x_0) + \left(\frac{6}{h^2} - a^2\right)g_0\right), \quad c_n = \frac{h^2}{36}\left(f(x_n) + \left(\frac{6}{h^2} - a^2\right)g_1\right).$$

With these two values known, we can look at the $(n - 1) \times (n - 1)$ system created by using (4.55) to define c_0 and c_n, and dropping the first and last equations in the

rectangular system. We are then left with the square system

$$Tc = b$$

where

$$T = \text{tridiag}(-6h^{-2} + a^2, 12h^{-2} + 4a^2, -6h^{-2} + a^2),$$

(which is diagonally dominant),

$$c = (c_1, c_2, \ldots, c_{n-1})^T,$$

and

$$b = \begin{bmatrix} f(x_1) + \frac{1}{6}(1 - \frac{1}{6}a^2h^2)f(x_0) + h^{-2}(1 - \frac{1}{6}h^2a^2)^2 g_0 \\ f(x_2) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) + \frac{1}{6}(1 - \frac{1}{6}a^2h^2)f(x_n) + h^{-2}(1 - \frac{1}{6}h^2a^2)^2 g_1 \end{bmatrix}.$$

Programming Note: We have used a grid of $n + 1$ points, with $h = 1/n$, but the system we solve is $n - 1 \times n - 1$.

The diagonal dominance of T means that we can use the solution algorithm from §2.6. Once the system is solved, we can get the rest of the coefficients from (4.53), (4.54), and (4.55).

■ EXAMPLE 4.10

Consider the example problem

$$\begin{aligned} -u'' + \pi^2 u &= 2\pi^2 \cos(\pi x), \quad 0 \le x \le 1, \\ u(0) &= 1, \\ u(1) &= -1, \end{aligned}$$

which has exact solution $u(x) = \cos \pi x$. The linear system for $h = \frac{1}{8}$ is

$$Tc = b$$

for

$$T = \text{tridiag}(-384 + \pi^2, 768 + 4\pi^2, -384 + \pi^2)$$

and

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 2\pi^2 \cos\left(\frac{\pi}{8}\right) + 64\left(1 - \frac{\pi^2}{384}\right)\left(1 + \frac{\pi^2}{192}\right) \\ 2\pi^2 \cos\left(\frac{2\pi}{8}\right) \\ 2\pi^2 \cos\left(\frac{3\pi}{8}\right) \\ 2\pi^2 \cos\left(\frac{4\pi}{8}\right) \\ 2\pi^2 \cos\left(\frac{5\pi}{8}\right) \\ 2\pi^2 \cos\left(\frac{6\pi}{8}\right) \\ 2\pi^2 \cos\left(\frac{7\pi}{8}\right) + 64\left(1 - \frac{\pi^2}{384}\right)\left(1 + \frac{\pi^2}{192}\right) \end{bmatrix}.$$

This has the solution

$$c = \begin{bmatrix} 0.15763927907189 \\ 0.12053536253339 \\ 0.06520274709146 \\ 0.00000000000000 \\ -0.06520274709146 \\ -0.12053536253339 \\ -0.15763927907189 \end{bmatrix}.$$

From this we get that

$$c_0 = 0.17095034913242, \quad c_{-1} = 0.15855932439844,$$

$$c_8 = -0.17095034913242, \quad c_9 = -0.15855932439844,$$

and the spline approximation is therefore given by

$$\begin{aligned} u_h(x) &= 0.15855932439844B_{-1}(x) + 0.17095034913242B_0(x) + 0.15763927907189B_1(x) \\ &+ 0.12053536253339B_2(x) + 0.06520274709146B_3(x) + 0B_4(x) \\ &- 0.06520274709146B_5(x) - 0.12053536253339B_6(x) - 0.15763927907189B_7(x) \\ &- 0.17095034913242B_8(x - 0.15855932439844B_9(x) \end{aligned}$$

and the error $\cos(\pi x) - u_h(x)$ is plotted in Figure 4.24. If we take a sequence of grids with $h^{-1} = 8, 16, 32, \ldots, 128$ and compute the approximate solutions, we find that the norm of the error, as estimated by sampling at 200 discrete points on the interval, is as indicated in Table 4.22; note that the error goes down like a factor of 4 as h is cut in half, indicating (but not proving) that this scheme is $\mathcal{O}(h^2)$ accurate. Note that this means that the spline solution of the boundary value problem is *less* accurate than a direct spline approximation of the exact solution (which, according to the estimate given in §4.8, would be $\mathcal{O}(h^4)$ accurate).

Figure 4.24 Error in spline approximation to boundary value problem for $h = 1/8$.

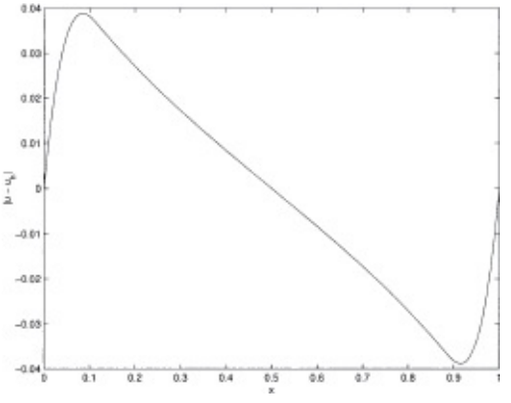


Table 4.22 Estimated error for spline approximation to BVP solution.

h^{-1}	Estimated error
8	0.03881321103858
16	0.01100251675792
32	0.00295824984969
64	0.00076798902568
128	0.00019487261990
256	0.00004945300189

It can be shown that this approximation is indeed $\mathcal{O}(h^2)$ accurate, but the analysis is beyond our intended scope.

What we have done in this section is an example of what is called *collocation*, a broadly used technique for solving differential equations (approximately) using expansions in terms of basis functions, and then solving for the coefficients in the expansion. Another method that uses a basis function expansion (and which uses a

more complicated means of constructing the system for the expansion coefficients) is the finite element method. The book by Prenter [14] has some discussion of collocation by splines. We will briefly discuss spectral collocation in Chapter 10, and the finite element method in §6.10.3 and Chapter 9.

Exercises:

1. Set up the linear system for solving the boundary value problem

$$-u'' + u = 1, \quad u(0) = 1, \quad u(1) = 0,$$

using $h = \frac{1}{4}$. You should get

$$\begin{bmatrix} 196 & -95 & 0 \\ -95 & 196 & -95 \\ 0 & -95 & 196 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 101/6 \\ 1 \\ 671/576 \end{bmatrix}.$$

2. Solve the system in the previous problem and find the coefficients for the spline expansion of the approximate solution. The exact solution is

$$u(x) = 1 - \frac{e}{e^2 - 1} e^x + \frac{e}{e^2 - 1} e^{-x};$$

plot your approximation and the error.

3. Use the method from this section to approximate the solution to each of the following boundary value problems using $h^{-1} = 8, 16, 32$. Estimate the maximum error in each case by sampling the difference between the exact and approximate solutions at 200 equally spaced points on the interval.

(a) $-u'' + u = (\pi^2 + 1) \sin \pi x, \quad u(0) = u(1) = 0; u(x) = \sin \pi x;$

(b) $-u'' + u = \pi(\pi \sin \pi x + 2 \cos \pi x) e^{-x}, \quad u(0) = u(1) = 0; u(x) = e^{-x} \sin \pi x;$

(c) $-u'' + u = 3 - \frac{1}{x} - (x^2 - x - 2) \log x, \quad u(0) = u(1) = 0; u(x) = x(1 - x) \log x.$

(d) $-u'' + u = 4e^{-x} - 4xe^{-x}, \quad u(0) = u(1) = 0; u(x) = x(1 - x)e^{-x};$

(e) $-u'' + \pi^2 u = 2\pi^2 \sin(\pi x), \quad u(0) = 1, \quad u(1) = 0, u(x) = \sin(\pi x)$

(f) $-u'' + u = \frac{x^2 + 2x - 1}{(1+x)^3}, \quad u(0) = 1, \quad u(1) = 1/2, u(x) = (1 + x)^{-1}.$

4. Try to extend the method from this section to the more general two-point boundary value problem defined by

$$\begin{aligned} -u'' + bu' + u &= f(x), \\ u(0) = u(1) &= 0. \end{aligned}$$

Is the resulting linear system diagonally dominant for all values of b?

5. Try to extend the method from this section to the more general two-point boundary value problem defined by

$$\begin{aligned} -a(x)u'' + u &= f(x), \\ u(0) = u(1) &= 0. \end{aligned}$$

Is the resulting linear system diagonally dominant for all choices of $a(x)$?

4.10 TENSION SPLINES

Splines are a wonderful tool for approximation, but they can still exhibit some poor behavior. Consider the data set plotted in Fig. 4.25. Obviously, this represents a function with a severe jump near $x = 0.5$, but there is no sign of oscillatory behavior. However, a B-spline representation of this data (Fig. 4.26) shows small wiggles on either side of a sharp front. This is fundamentally an artifact of the steep gradient in the data, but in other contexts a spline fit can display behavior that does not match the "sense" of the data. One way to avoid the problem is the notion of a *taut spline* or *tension spline*, an idea that appears to have been first published by Schweikert [17], but which also owes a lot to the work of A. K. Cline [4]; we relied heavily on a short paper of Marušić and Rogina [12] in our presentation here.

Figure 4.25 Data set for tensioned spline illustration.

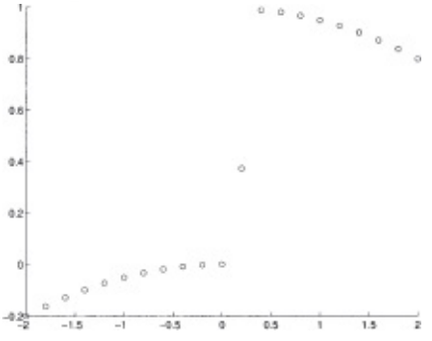
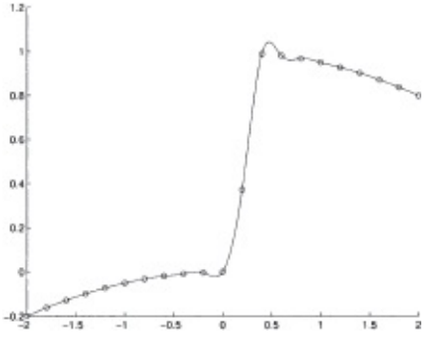


Figure 4.26 B-spline fit to data in Fig. 4.25.



Imagine that the curve in Fig 4.26 is a piece of string that is constrained to pass through small loops at the data points. If we were to pull the string taut, we would smooth out the spurious oscillations in the curve. This amounts to studying the mechanical properties of a cable hanging between two supports. More prosaically, we construct our spline from the new basis set $\{1, x, \cosh px, \sinh px\}$, where $p > 0$ is the tension parameter: $p = 0$ means no tension, and it can be shown that this corresponds to the pure spline approximation; $p \to \infty$ gives us a piecewise linear approximation. (We will not attempt to justify either of these statements other than by examples and exercises.)

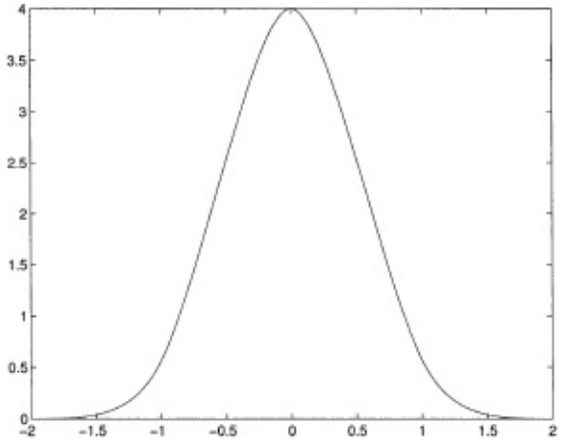
The reader may well be wondering how practical this scheme might be. After all, we have traded a set of polynomial basis functions for a set of transcendental basis functions. Not only is this going to make execution of any program more expensive, but it leaves open

the entire question of how to construct the approximation. The basic idea is the same as in §4.8: First, we construct a primary basis function, as in (4.34):

$$\tau(x) = \frac{2}{\alpha} \begin{cases} \sinh px \cosh 2p + \cosh px \sinh 2p - px - 2p, & x \in [-2,-1] \\ -(\sinh px - px)\beta - 2\cosh px \sinh p + \gamma & x \in [-1,0] \\ (\sinh px - px)\beta - 2\cosh px \sinh p + \gamma & x \in [0,1] \\ -\sinh px \cosh 2p + \cosh px \sinh 2p + px - 2p, & x \in [1,2] \end{cases} \tag{4.56}$$

where $\alpha = p \cosh p - \sinh p$, $\beta = (1 + 2 \cosh p)$, and $\gamma = 2p \cosh p$. Confirmation of this formula is deferred to the exercises. A plot of τ is given in Fig. 4.27, for $p = 4$; note that it does not look very different from Fig. 4.13.

Figure 4.27 Original taut B-spline, $p = 4$.



Construction of a tension spline follows precisely the same recipe as in §4.8: Given data $(x_i, y_i)_{i=1}^{i=n}$, we look for an approximation in the form

$$s(x) = \sum_{i=-1}^{n+1} c_i \tau_i(x),$$

where the c_i values are coefficients to be determined, and

$$\tau_i(x) = \tau\left(\frac{x - x_i}{h}\right).$$

Note that, as in §4.8, we have added additional grid points, which will again require the imposition of additional conditions. We will explicitly cover the *natural* spline case, leaving the complete spline case to the exercises.

For each original grid point we have the equation

$$\sum_{i=-1}^{n+1} c_i \tau_i(x) = y_j.$$

Because of the local nature of the τ_i functions, this becomes

$$c_{j-1}\tau_{j-1}(x_j) + c_j \tau_j(x_j) + c_{j+1}\tau_{j+1}(x_j) = y_j.$$

We quickly have that

$$\tau_j(x_j) = \tau\left(\frac{x_j - x_j}{h}\right) = \tau(0) = 4,$$

and

$$\tau_{j-1}(x_j) = \tau_{j+1}(x_j) = \tau(\pm 1) = \theta(p),$$

for

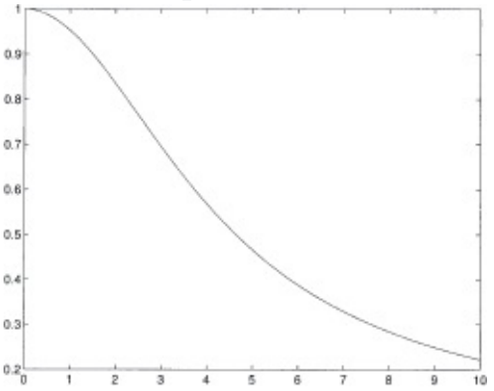
$$\theta(p) = \frac{2[(\sinh p - p)(1 + 2\cosh p) - 2\cosh p \sinh p + 2p\cosh p]}{p\cosh p - \sinh p}.$$

This certainly looks imposing, but it also looks like some simplification ought to be possible, and if we multiply out the numerator, we quickly get

$$\theta(p) = \frac{2[(\sinh p - p)(1 + 2\cosh p) - 2\cosh p \sinh p + 2p\cosh p]}{p\cosh p - \sinh p} = 2\left(\frac{\sinh p - p}{p\cosh p - \sinh p}\right).$$

This is plotted in Fig. 4.28, for p running from 0 to 10.

Figure 4.28 $\theta(p) = \tau(\pm 1)$ as a function of p.



The important thing for our purposes is that $0 < \theta(p) \leq 1$. The system of linear equations for the taut spline is, initially, as follows:

$$\begin{bmatrix} \theta(p) & 4 & \theta(p) & & & \\ & \theta(p) & 4 & \theta(p) & & \\ & & \ddots & \ddots & \ddots & \\ & & & \theta(p) & 4 & \theta(p) \end{bmatrix} \begin{bmatrix} c_{-1} \\ c_0 \\ \vdots \\ c_{n+1} \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}. \tag{4.57}$$

This, of course, is a system of $n + 1$ equations in $n + 3$ unknowns. Following the derivation for the natural spline in §4.8, we have the boundary conditions

$$\tau''(x_0) = 0 \Rightarrow c_{-1}\tau''_{-1}(x_{-1}) + c_0\tau''_0(x_0) + c_1\tau''_1(x_1) = 0,$$

and

$$\tau''(x_n) = 0 \Rightarrow c_{n-1}\tau''_{n-1}(x_n) + c_n\tau''_n(x_n) + c_{n+1}\tau''_{n+1}(x_n) = 0.$$

Continuing with $x = x_0$, we get that

$$c_{-1}\tau''(1) + c_0\tau''(0) + c_1\tau''(-1) = 0. \tag{4.58}$$

We still need to compute τ'' at various points. It is an involved, but not arduous computation, to show that

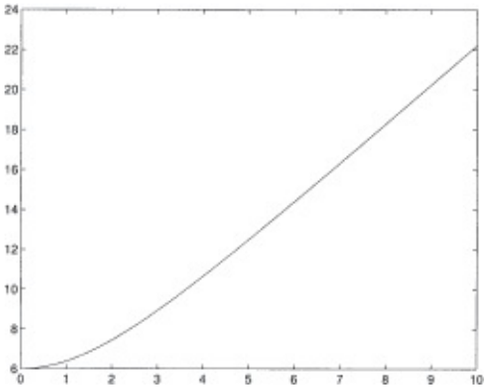
$$\tau''(0) = \delta_{2,0}(p) = \frac{-4p^2 \tanh p}{p - \tanh p},$$

and

$$\tau''(\pm 1) = \delta_{2,1}(p) = \frac{2p^2 \tanh p}{p - \tanh p} = -\frac{1}{2}\delta_{2,0}(p)$$

We have plotted $\delta_{2,1}(p)$ in Fig. 4.29. We therefore have, from (4.58),

Figure 4.29 $\delta_{2,1}(p) = \tau''(\pm 1)$ as a function of p.



$$c_{-1} = 2c_0 - c_1.$$

Similarly,

$$c_{n+1} = 2c_n - c_{n-1}.$$

Note that this looks a lot like what we got in §4.8 for the ordinary natural B-spline; we can eliminate c_{-1}, c_0, c_n, and c_{n+1}, so the system (4.57) becomes (after a bit of work)

where

$$K = \text{tridiag}(\theta(p), 4, \theta(p)),$$

(4.59) $$c = (c_1, c_2, \ldots, c_{n-1})^T,$$

and

$$F = (f(x_1) - \theta(p)c_0, f(x_2), \ldots, f(x_{n-2}), f(x_{n-1}) - \theta(p)c_n)^T,$$

with

$$c_{-1} = 2c_0 - c_1,$$

$$c_0 = \frac{f(x_0)}{4 - \frac{\theta(p)\delta_{2,0}(p)}{\delta_{2,1}(p)}} = \frac{f(x_0)}{4 + 2\theta(p)},$$

$$c_n = \frac{f(x_n)}{4 - \frac{\theta(p)\delta_{2,0}(p)}{\delta_{2,1}(p)}} = \frac{f(x_n)}{4 + 2\theta(p)},$$

and

$$c_{n+1} = 2c_n - c_{n-1}.$$

Note that the matrix is tridiagonal and diagonally dominant, so we know how to solve it.

■ EXAMPLE 4.11

As an obvious illustration of this, let's fit a series of tension splines to the data set in Fig. 4.25. Once the coefficients c_{-1}, c_0, …, c_n, c_{n+1} are computed, the spline is evaluated in the same way as for an ordinary B-spline. Fig. 4.30 shows the data and spline curve for $p = 0.01$; obviously, with such a small tension value we do not expect much difference, and Fig. 4.31, which plots the difference between the pure polynomial spline and the tension spline, confirms this (although the lower "overshoot" does appear to be significantly affected). If we take $p = 1$, we get Figs. 4.32–4.33 which shows that the lower "overshoot" is indeed beginning to damp out, but the upper one is much the same. Finally, for $p = 6$, we get Figs. 4.34–4.35; this is perhaps the smallest value of p for which both overshoots are gone.

Figure 4.30 Tension spline fit, $p = 0.01$.

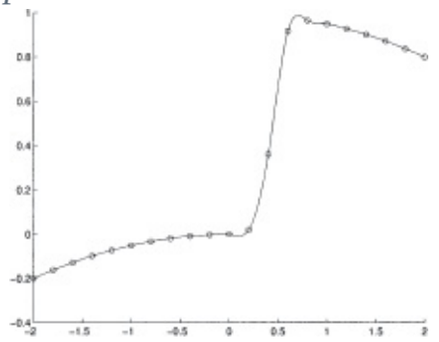


Figure 4.31 Difference between pure polynomial spline and tension spline, $p = 0.01$

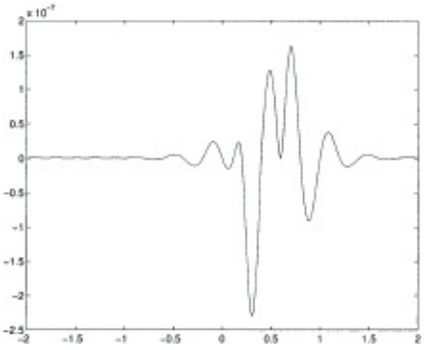


Figure 4.32 Tension spline fit, $p = 1$.

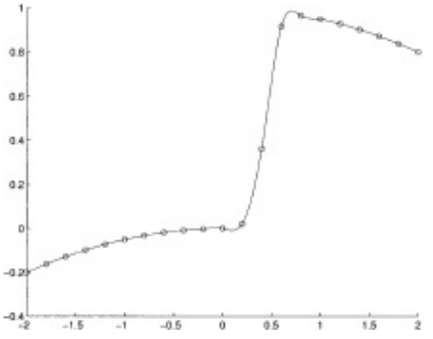


Figure 4.33 Difference between pure polynomial spline and tension spline, $p = 1$

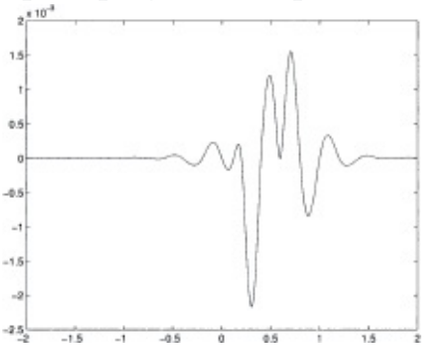


Figure 4.34 Tension spline fit, $p = 6$.

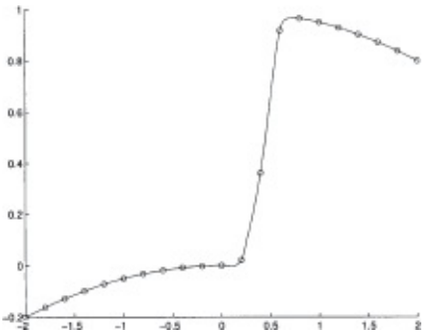
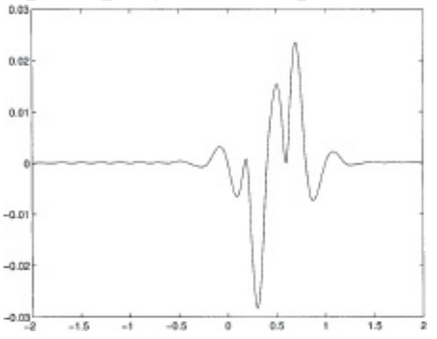


Figure 4.35 Difference between pure polynomial spline and tension spline, $p = 6$.



Exercises:

1. Show that the piecewise function defined in (4.56) is, indeed, continuous and has continuous first and second derivatives.

2. Fill in the details of the natural spline construction. In particular, confirm the expressions for $\delta_{2,0}(p)$ and $\delta_{2,1}(p)$ (and that $\tau''(1) = \tau''(-1)$) as well as the form of the final linear system (4.59).

3. Derive the linear system for the construction of a complete taut spline, by following what was done in §4.8.

4. Consider the dataset in Table 4.23:

Table 4.23 Data for Problem 4

x	600	650	700	750	800	850	900	950	1000	1050	1100
y	0.64	0.65	0.66	0.69	0.91	2.2	1.2	0.62	0.6	0.61	0.61

Plot the data, and construct a (natural) polynomial spline fit to it. Note the "wiggles" to the left of the peak, which appear to be contrary to the sense of the data, which is increasing monotonically towards the peak near $x = 850$. Find the smallest value of p in a taut natural spline fit to this data which yields a monotone curve to the left of the peak.

5. Repeat the previous problem using complete splines. Use a simple finite difference approximation based on the data to get the necessary derivative values.

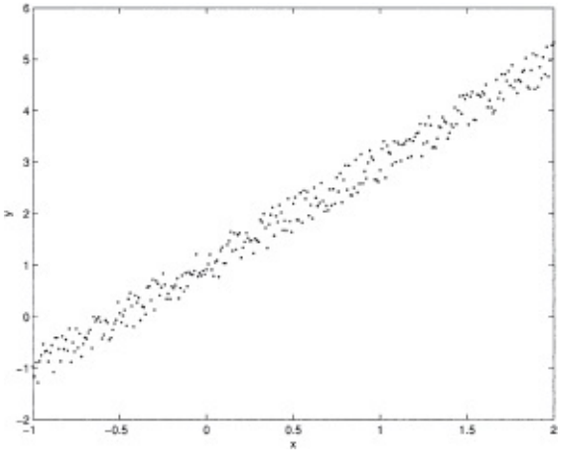
4.11 LEAST SQUARES CONCEPTS IN APPROXIMATION

4.11.1 An Introduction to Data Fitting

An important area in approximation is the problem of fitting a curve to experimental data. Since the data is experimental, we must assume that it is polluted with some degree of error, most commonly measurement error ("noise"), so we do not necessarily want to construct a curve that goes through every data point. (In fact, the material in §4.12.1 suggests that this would be a disastrous way to proceed.) Rather, we want to construct a function that represents the "sense of the data" and which is, in some sense, a close approximation to the data.

The most common approach is known as *least squares* data fitting. Consider Figure 4.36; this shows an example set of data for which the general trend is clearly a straight line. But *which* straight line do we use to represent the data?

Figure 4.36 Example plot of data.



The least squares approach defines the "correct" straight line as the line that minimizes the sum of the squares of the distances between the data points and the line. Let the experimental data be defined as pairs (x_k, y_k), $1 \le k \le n$ for some n. Thus, we want to find the coefficients m and b in the equation $y = mx + b$ such that

$$F(m,b) = \sum_{k=1}^{n} (y_k - (mx_k + b))^2$$

is minimized. This is a straight forward problem from multivariable calculus: We compute the partial derivatives F_m and F_b and find where they both vanish, and this will define a critical point. It can be shown that this critical point defines a global minimum for F. Thus, m and b are defined by the two equations

$$\frac{\partial F}{\partial m} = -2\sum_{k=1}^{n}\left(y_k - (mx_k + b)\right)x_k = 0,$$

$$\frac{\partial F}{\partial b} = -2\sum_{k=1}^{n}\left(y_k - (mx_k + b)\right) = 0,$$

which can be simplified to a system of two equations in two unknowns

$$\begin{aligned}\sum_{k=1}^{n}\left(y_k - (mx_k + b)\right)x_k &= 0,\\ \sum_{k=1}^{n}\left(y_k - (mx_k + b)\right) &= 0,\end{aligned}$$

or,

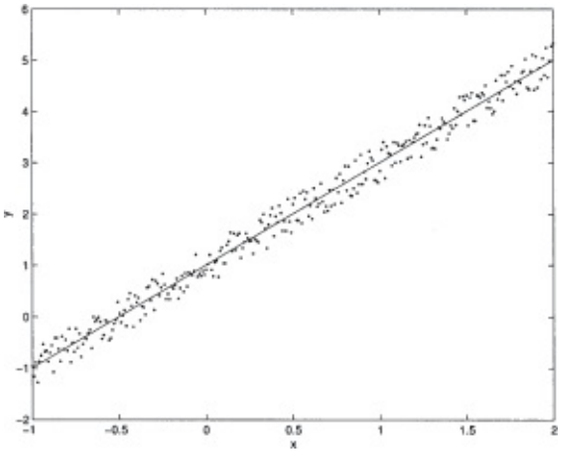
$$\begin{aligned}m\left(\sum_{k=1}^{n}x_k^2\right) + b\left(\sum_{k=1}^{n}x_k\right) &= \sum_{k=1}^{n}x_k y_k,\\ m\left(\sum_{k=1}^{n}x_k\right) + b\left(\sum_{k=1}^{n}1\right) &= \sum_{k=1}^{n}y_k.\end{aligned}$$

The solution here is then

$$\begin{aligned}m &= \frac{n\sum_{i=1}^{n}x_k y_k - \left(\sum_{i=1}^{n}x_k\right)\left(\sum_{i=1}^{n}y_k\right)}{n\sum_{k=1}^{n}x_k^2 - \left(\sum_{i=1}^{n}x_k\right)^2},\\ b &= \frac{\left(\sum_{i=1}^{n}x_k^2\right)\left(\sum_{i=1}^{n}y_k\right) - \left(\sum_{i=1}^{n}x_k\right)\left(\sum_{i=1}^{n}x_k y_k\right)}{n\left(\sum_{k=1}^{n}x_k^2\right) - \left(\sum_{i=1}^{n}x_k\right)^2},\end{aligned}$$

which, for the data in Figure 4.36, produces the straight line graph shown in Figure 4.37.

Figure 4.37 Straight line fit to data.



The notion of a least squares data fit can be generalized beyond simply fitting a straight line to data. We can look at higher degree polynomials and we can also look at higher dimensional data sets. The exercises include some examples of more involved least squares data fit problems.

■ EXAMPLE 4.12

Consider the data in Table 4.24. If we plot this, we get what appears to be a straight line as the general trend of the data, so we look for the equation of the line $y = mx + b$

which best fits this data in the least squares sense. Forming the separate sums gives us

$$\sum_{i=1}^{6} x_i = 15, \quad \sum_{i=1}^{6} y_i = 367, \quad \sum_{i=1}^{6} x_i^2 = 55, \quad \sum_{i=1}^{6} x_i y_i = 1303,$$

from which it follows that

$$m = 22.0286, \quad b = 6.0952$$

and the line is plotted, along with the data, in Figure 4.38.

Figure 4.38 Least squares fit to the data in Table 4.24.

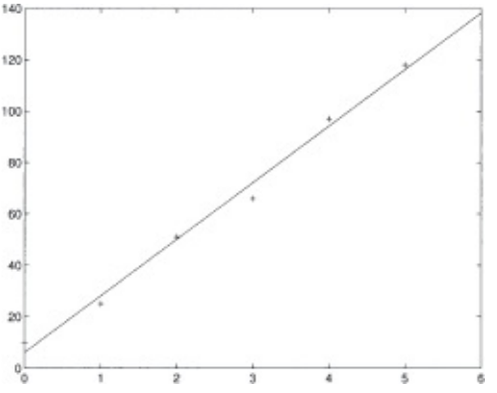


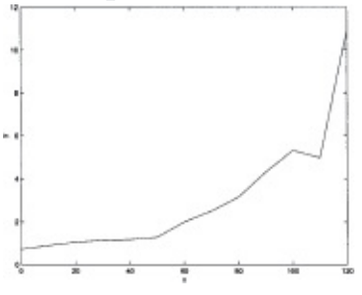
Table 4.24 Data for Example 4.12.

x	0.0	1.0	2.00	3.00	4.00	5.00
y	10.0	25.0	51.0	66.0	97.0	118

■ EXAMPLE 4.13

We don't have to restrict ourselves to linear or quadratic models to make good use of the idea of least squares data fits. Consider the y_k data in Table 4.25. When we plot this data, we get the curve shown in Figure 4.39. Generally, this looks like an exponential growth curve. Ordinarily this would require us to do a fit to a curve of the form $y_k = Ae^{rx_k}$, which will lead to a nonlinear system for the parameters A and r. However, if the raw data is exponential, then the logarithm of the data is linear, since we have

Figure 4.39 Plot of raw data for Example 4.13.

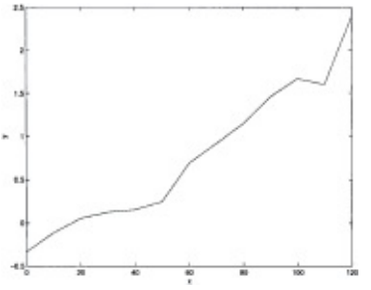


$$z_k = \ln y_k = rx_k + (\ln A),$$

and we can fall back on our existing algorithm to do a fit to the log data.

To verify this, we look at the logarithm of our example data; this data is plotted in Figure 4.40, and the general trend is indeed linear. So we do a least squares fit to the log data, getting the straight line

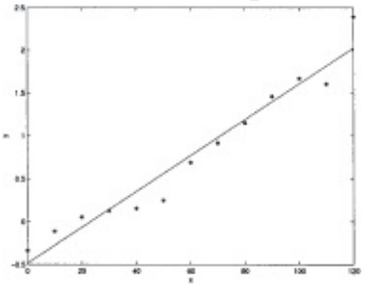
Figure 4.40 Plot of log data for Example 4.13.



$$z = 0.0209x - 0.4842.$$

This is plotted, along with the raw (logarithm) data, in Figure 4.41. It follows, then, that our curve fit to the original data is

Figure 4.41 Log data plus fitted curve for Example 4.13.



$$y = e^{(0.0209x - 0.4842)},$$

which is plotted in Figure 4.42, along with the original data. Note that, except for the last two points, this is a pretty good fit to the data set.

Figure 4.42 Raw data plus fitted curve for Example 4.13.

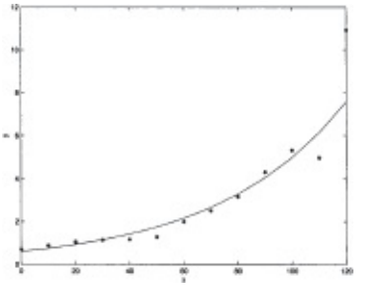


Table 4.25 Data for Example 4.13.

k	x_k	y_k	$\log y_k$	k	x_k	y_k	$\log y_k$
1	0	0.716	–0.3341	8	70	2.500	0.9163
2	10	0.893	–0.1132	9	80	3.151	1.1477
3	20	1.055	0.0535	10	90	4.300	1.4586
4	30	1.134	0.1258	11	100	5.308	1.6692
5	40	1.167	0.1544	12	110	4.966	1.6026
6	50	1.281	0.2476	13	120	10.919	2.3905
7	60	1.994	0.6901				

4.11.2 Least Squares Approximation and Orthogonal Polynomials

The notion of least squares approximation can be extended beyond the data-fitting

problem. Consider the problem of finding an approximation to a given function f in terms of a set of basis functions $\{\phi_k,\ 1 \le k \le n\}$. How do we find the coefficients in the expansion

$$f(x) \approx q_n(x) = \sum_{k=1}^{n} c_k \phi_k(x)?$$

One way to do this is to require that the coefficients c_k produce an approximation that minimizes the error

$$r_n = f - q_n$$

in the least squares sense, i.e., in the sense of the integral 2-norm. Thus we seek c_k such that

$$R_n = \|f - q_n\|_2^2$$

is minimized. Before doing this it will be convenient to introduce the notion of an *inner product*. It is best to do this thoroughly, so we pause in our development of approximations, but only briefly.

Inner Products of Functions The reader should be familiar with the notion of the dot product of two vectors in $\mathbb{R}^n$:

$$x \cdot y = \sum_{i=1}^{n} x_i y_i.$$

This is an example of a more general operation called an *inner product*, which can be defined on general vector spaces, including spaces of functions, rather than just Euclidean *n*-space. The formal definition is as follows.

Definition 4.2 (Inner Product on Real Vector Spaces) *Let f and g be elements of a real vector space V. Let* (f, g) *denote any operation on f and g that satisfies the following three properties:*

1. $(f, f) > 0$ *for all nonzero* $f \in V$;

2. $(f, \alpha g_1 + \beta g g_2) = \alpha(f, g_1) + \beta(g, g_2)$ *for all* $f, g \in V$ *and all scalars* α *and* β;

3. $(f, g) = (g, f)$ *for all* $f, g \in V$.

Then, (f, g) *is called an* inner product.

If we want to consider our overlying vector space to be $C([a, b])$, that is, continuous functions on a closed interval, then we can easily establish that the positively weighted integral of a product of two functions will be an inner product.

Theorem 4.8 *Let w be integrable on* $[a, b]$ *and non-negative, i.e.,* $\int_a^b w(x)dx$ *is defined and* $w(x) \ge 0$ *for all* $x \in [a, b]$. *For given f and g in* $C([a, b])$, *define* $(f, g)_w$ *as*

$$(f, g)_w = \int_a^b w(x) f(x) g(x) dx. \tag{4.60}$$

Then, $(\cdot, \cdot)_w$, *defines an inner product on* $C([a, b])$.

Proof: See Problem 6. •

Note that it therefore follows that if $(f, g)_w$ is an inner product, then $\|f\|_w = (f, f)_w^{1/2}$

defines a norm. In the common case when $w(x) \equiv 1$, we will simply write $(f, g)_w = (f, g)$; i.e., we will drop the subscript, and the norm is the ordinary 2-norm for functions that we defined earlier in this chapter. Problem 7 offers some practice with a norm defined by a weighted inner product.

This definition of inner product will allow us to apply a number of ideas from linear algebra to the construction of approximations, as we will soon see. Of more immediate interest is the fact that we can use the inner product notation to write the residual, R_n, in a very convenient form:

$$R_n = \|f\|_w^2 - 2\sum_{k=1}^{n} c_k(f, \phi_k) + \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j(\phi_i, \phi_j).$$

Note that we can regard R_n as a function of the n variables c_k, and thus apply ordinary calculus to the problem of minimizing R_n. After some manipulations (see Problem 8) we find that the c_k, are defined by simultaneously solving the set of equations

$$\begin{array}{ccccccc}
(\phi_1,\phi_1)c_1 & + & (\phi_1,\phi_2)c_2 & + & \cdots & (\phi_1,\phi_n)c_n & = & (f,\phi_1), \\
(\phi_2,\phi_1)c_1 & + & (\phi_2,\phi_2)c_2 & + & \cdots & (\phi_2,\phi_n)c_n & = & (f,\phi_2), \\
\vdots & & \vdots & & \cdots & \vdots & \vdots & \vdots \\
(\phi_n,\phi_1)c_1 & + & (\phi_n,\phi_2)c_2 & + & \cdots & (\phi_n,\phi_n)c_n & = & (f,\phi_n).
\end{array}$$

This system can be organized along matrix–vector lines as

$$\begin{bmatrix}
(\phi_1,\phi_1) & (\phi_1,\phi_2) & \cdots & (\phi_1,\phi_n) \\
(\phi_2,\phi_1) & (\phi_2,\phi_2) & \cdots & (\phi_2,\phi_n) \\
\vdots & \vdots & \cdots & \vdots \\
(\phi_n,\phi_1) & (\phi_n,\phi_2) & \cdots & (\phi_n,\phi_n)
\end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}
=
\begin{bmatrix} (f,\phi_1) \\ (f,\phi_2) \\ \vdots \\ (f,\phi_n) \end{bmatrix}. \tag{4.61}$$

Solving a system of linear equations is a problem that we do not encounter in the general case until Chapter 7. We can avoid it altogether at this point *if* our basis functions satisfy the *orthogonality* condition

$$(\phi_i, \phi_j) = 0, \quad \text{for all } i \neq j. \tag{4.62}$$

In this case, the matrix in (4.61) is a diagonal matrix and we very easily have

$$c_k = \frac{(f, \phi_k)}{(\phi_k, \phi_k)}.$$

So, to summarize what we have done so far, we can construct an approximation to a given function f from a given basis set $\{\phi_1, \phi_2, \ldots, \phi_n\}$, and the construction is very easy, *if the* basis satisfies the condition (4.62). So, the question becomes: When can we find a basis that satisfies (4.62), and how good is the resulting approximation?

The answer is that we can always find such a basis if we consider polynomial functions for our basis elements, and the resulting approximations are usually quite good. The special basis functions that satisfy (4.62) are called *orthogonal polynomials*. To be more specific with this, we have to introduce some new concepts and notation, and recall a major theorem from linear algebra. But first, one more definition.

Definition 4.3 (Vector Space of Polynomials of Degree $\leq N$) *For any $N \geq 0$ define $\mathcal{P}_N$ as the* vector space *of polynomials of degree $\leq N$. Note that this space has a standard basis*

consisting of $\{1, x, x^2, x^3, \ldots, x^N\}$, *and thus is an* $(N+1)$*-dimensional space.*

And, now, the theorem:

Theorem 4.9 *Let* w *be a given non-negative weight function on an interval* $[a, b]$, *and* $(\cdot, \cdot)_w$ *the associated inner product, defined as in (4.60). Then there exists a family of orthogonal polynomials* $\{\phi_k\}$, $\phi_k \in P_k$, $0 \le k \le N$, *such that*

$$(\phi_i, \phi_j)_w = 0, \quad i \neq j,$$

and

$$(\phi_i, \phi_i)_w > 0, \quad i \geq 0.$$

In addition, the ϕ_k *satisfy the following:*

1. *The set* $\{\phi_0, \phi_1, \ldots, \phi_N\}$ *is a basis for* $\mathcal{P}_N$;

2. *If* q_k *is an arbitrary element of* $\mathcal{P}_k$, *for* $k < N$, *then* $(q_k, \phi_N)_w = 0$ *for all* $N > k$ *(thus, orthogonal polynomials are orthogonal to all polynomials of strictly lower degree);*

3. *For* $j \geq 1$, *the roots of each* ϕ_j *are all in* $[a, b]$ *and are all distinct.*

Proof: The proof is somewhat lengthy, in part because of the length of the theorem, but it is not difficult.

To establish that the family $\{\phi_k\}$ exists, we will construct it directly, using the Gram–Schmidt process from linear algebra. Take $\phi_0(x) = 1$, and define $\xi_k(x) = x^k$, for $0 \le k \le N$. Then the subsequent ϕ_k can be found according to

$$\phi_k(x) = \xi_k(x) - \sum_{j=0}^{k-1} \frac{(\phi_j, \xi_k)_w}{(\phi_j, \phi_j)_w} \phi_j(x) \tag{4.63}$$

and an inductive argument shows very quickly that the orthogonality holds. In Problem 9 we ask the student to fill in the details of this part of the proof.

Having now proved that the family of orthogonal polynomials exists, we turn our attention to proving each of (1)–(3).

(1) The space $\mathcal{P}_N$ is finite-dimensional with dimension $N + 1$. It therefore follows that any set of $N + 1$ independent elements of $\mathcal{P}_N$ will be a basis. The orthogonality condition (4.62) forces the members of the family $\{\phi_k\}$ to be independent (see Problem 10); therefore, the set $\{\phi_0, \phi_1, \ldots, \phi_n\}$ is a basis for P_N.

(2) Let q_k be an arbitrary polynomial of degree $k < N$. Then we can write

$$q_k = \sum_{i=0}^{k} a_i \phi_i$$

because $\{\phi_0, \phi_1, \ldots, \varphi_k\}$ is a basis for $\mathcal{P}_k$ Therefore,

$$(q_k, \phi_N) = \sum_{i=0}^{k} a_i (\phi_i, \phi_N) = 0$$

since ϕ_N is orthogonal to each element of $\{\phi_0, \phi_1, \ldots, \phi_k\}$ In fact, we can write (Problem 11)

$$q_k = \sum_{i=0}^{k} \frac{(q_k, \phi_i)}{(\phi_i, \phi_i)} \phi_i. \tag{4.64}$$

(3) First, suppose that φ_j, $j \geq 1$, has no roots in $[a, b]$. This means that ϕ_j does not change sign on the interval, thus

$$(\phi_0, \phi_j) = \int_a^b w(x)\phi_0(x)\phi_j(x)dx \neq 0,$$

since the integrand does not change sign on $[a, b]$. But the orthogonality requires that $(\phi_0, \phi_j) = 0$; hence, we have a contradiction, so there must be at least one root in $[a, b]$.

Now, let x_1 be any root of ϕ_j that lies in $[a, b]$, and suppose that it is a multiple root. Then it follows that

$$\phi_j(x) = (x - x_1)^2 q(x)$$

for some polynomial q; thus,

$$q(x) = (x - x_1)^{-2}\phi_j(x)$$

is a polynomial of degree $j - 2$, therefore, $(q, \phi_j) = 0$, by another part of this theorem. But

$$(q, \phi_j) = \int_a^b w(x)(x - x_1)^{-2}(\phi_j(x))^2 dx > 0,$$

so we have another contradiction. Thus, any roots that lie in $[a, b]$ must be simple roots.

Suppose now that only some of the roots lie in $[a, b]$. Call these roots x_i, $1 \leq i < j$, and note that we can write ϕ_j as

$$\phi_j(x) = \Psi_j(x)(x - x_1) \cdots (x - x_i),$$

where $\Psi_j(x)$ does not change sign in $[a, b]$ and is a polynomial of degree $j - i$. Therefore,

$$\phi_j(x)(x - x_1) \cdots (x - x_i) = \Psi_j(x)(x - x_1)^2 \cdots (x - x_i)^2$$

is also a polynomial that does not change sign in $[a, b]$. Hence, the integral

$$I = \int_a^b \Psi_j(x)(x - x_1)^2 \cdots (x - x_i)^2 dx$$

cannot be zero. However,

$$I = \int_a^b \Psi_j(x)(x - x_1)^2 \cdots (x - x_i)^2 dx = (\phi_j, q)$$

where $q(x) = (x - x_1) \cdots (x - x_i)$, and q is a polynomial of degree $i < j$. Therefore, $(\phi_j, q) = 0$, and we have a contradiction. Thus, $i \geq j$, and since a polynomial of degree j cannot have more than j roots, we must have $i = j$. •

Families of Orthogonal Polynomials At this point it might be useful to look at some examples of orthogonal polynomial families. Four of the most common ones are discussed below.

Note that the orthogonality condition (4.62) means that an orthogonal polynomial can be multiplied by an arbitrary nonzero constant and still satisfy (4.62). To avoid the problems of nonuniqueness that this can lead to, it is common to impose a specific scaling on the elements of each family.

1. *Legendre polynomials:* The Legendre[13] polynomials are the orthogonal polynomials on

[−1, 1] with no weight function (more correctly, the unit weight function); thus, we have

$$\int_{-1}^{1} P_i(x)P_j(x)dx = 0, \quad i \neq j.$$

The usual scaling is to take $P_n(1) = 1$. The first five Legendre polynomials are:

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

2. *Chebyshev polynomials:* The common notation for the Chebyshev[14] polynomials is $T_n(x)$, and the interval and weight function are defined in the orthogonality relation

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}}dx = 0, \quad i \neq j.$$

The common scaling is to set the leading coefficient equal to 2^{n-1}. The first five Chebyshev polynomials are:

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

It can be shown that the Chebyshev polynomials are related in a very simple way to cosines; see Theorem 4.10.

3. *Hermite polynomials:* The Hermite polynomials are orthogonal on the entire real line,[15] using the weight function $w(x) = e^{-x2}$; that is,

$$\int_{-\infty}^{\infty} e^{-x^2} H_i(x)H_j(x)dx = 0, \quad i \neq j.$$

The common scaling is to set the leading coefficient equal to 2^n. The first five Hermite polynomials are:

$$\begin{aligned} H_0(x) &= 1, \\ H_1(x) &= 2x, \\ H_2(x) &= 4x^2 - 2, \\ H_3(x) &= 8x^3 - 12x, \\ H_4(x) &= 16x^4 - 48x^2 + 12. \end{aligned}$$

4. *Laguerre Polynomials:* The Laguerre[16] polynomials are orthogonal on the positive real line, using the weight function $w(x) = e^{-x}$, that is,

$$\int_{0}^{\infty} e^{-x} L_i(x)L_j(x)dx = 0, \quad i \neq j.$$

The common scaling is to set the leading coefficient equal to $\frac{(-1)^n}{n!}$. The first five Laguerre polynomials are then:

$$
\begin{aligned}
L_0(x) &= 1, \\
L_1(x) &= -x + 1, \\
L_2(x) &= \frac{1}{2}(x^2 - 4x + 2), \\
L_3(x) &= \frac{1}{6}(-x^3 + 9x^2 - 18x + 6), \\
L_4(x) &= \frac{1}{24}(x^4 - 16x^3 + 72x^2 - 96x + 24).
\end{aligned}
$$

We can use any of these orthogonal polynomial families to construct approximations to functions defined on the appropriate interval. These approximations are "best possible" in the sense that they minimize the error in the appropriate weighted 2-norm; i.e.,

$\|f - q_n\|_w < \|f - p_n\|_w$

for all $\frac{(-1)^n}{n!}$.

Consider, as illustrations, the following set of examples.

EXAMPLE 4.14

Let's construct the fourth-degree least squares approximation to the exponential function, $f(x) = e^x$, over the interval $[-1, 1]$, using Legendre polynomials. The approximation is defined by

$$p_4(x) = \sum_{k=0}^{4} \frac{(f, P_k)}{(P_k, P_k)} P_k,$$

where the P_k are the Legendre polynomials. We thus need to compute the integrals

$$I_0 = \int_{-1}^{1} e^x dx, \quad I_1 = \int_{-1}^{1} x e^x dx, \quad I_2 = \int_{-1}^{1} \frac{1}{2}(3x^2 - 1) e^x dx,$$

$$I_3 = \int_{-1}^{1} \frac{1}{2}(5x^3 - 3x) e^x dx, \quad I_4 = \int_{-1}^{1} \frac{1}{8}(35x^4 - 30x^2 + 3) e^x dx,$$

and

$$J_0 = \int_{-1}^{1} dx, \quad J_1 = \int_{-1}^{1} x^2 dx, \quad J_2 = \int_{-1}^{1} \frac{1}{4}(3x^2 - 1)^2 dx,$$

$$J_3 = \int_{-1}^{1} \frac{1}{4}(5x^3 - 3x)^2 dx, \quad J_4 = \int_{-1}^{1} \frac{1}{64}(35x^4 - 30x^2 + 3)^2 dx.$$

In practice, these integrals would be computed using some type of numerical integration routine, such as the trapezoid rule or the more accurate methods we discuss in Chapter 5. For this simple example, though, it is possible to use direct calculus methods or (much more attractive!) a computer algebra package such as Maple or Mathematica. However it is done, to eight digits the integrals are

$$I_0 = 2.3504024, \quad I_1 = 0.73575888, \quad I_2 = 0.14312574,$$
$$I_3 = 0.020130181, \quad I_4 = 0.0022144731,$$

and

$$J_0 = 2.00000000, \quad J_1 = 0.66666667, \quad J_2 = 0.40000000,$$
$$J_3 = 0.28571429, \quad J_4 = 0.22222222,$$

so the polynomial approximation is

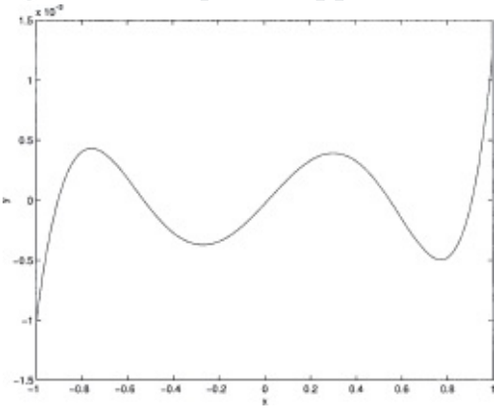
$$p_4(x) = \frac{2.3504024}{2.00000000}P_0(x) + \frac{0.73575888}{0.66666667}P_1(x) + \frac{0.14312574}{0.40000000}P_2(x) + \frac{0.020130181}{0.28571429}P_3(x) + \frac{0.0022144731}{0.22222222}P_4(x),$$

which simplifies to

$$p_4(x) = 1.0000309 + 0.99795487x + 0.49935229x^2 + 0.17613908x^3 + 0.043597439x^4.$$

Figure 4.43 shows a plot of the error $e^x - p_4(x)$. Compare this to the error plots for fourth-degree Taylor approximation and fourth degree Lagrange or Newton interpolation from earlier in this chapter. Note, in particular, that the least squares error oscillates back and forth between its maximum and minimum values (or nearly so), several times. It can be shown that this is a necessary and sufficient condition for the approximating polynomial to be the "best" approximation to the function, and is one reason why least squares approximations are considered valuable: they are close to being the best possible approximations.

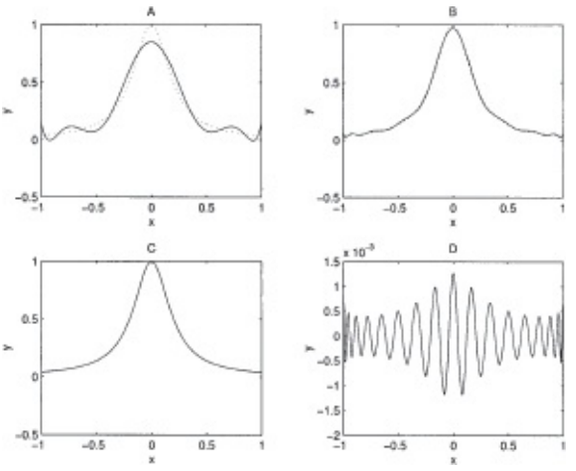
Figure 4.43 Error in fourth-degree least squares approximation to the exponential function.



EXAMPLE 4.15

Here we construct a Legendre polynomial approximation to the function $f(x) = (1 + 25x^2)^{-1}$ using $n = 8$, 16, and 32 degree polynomials. Figures 4.44A–C show plots of both f and the least squares approximation; Figure 4.44D shows the error in the 32-degree approximation.

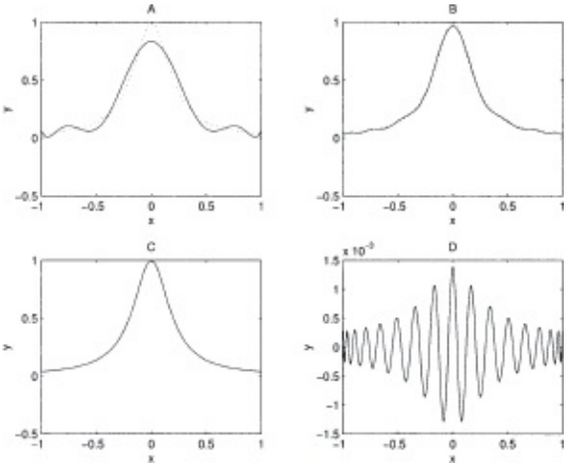
Figure 4.44 Legendre least squares approximation to $f(x) = 1/(1 + 25x^2)^{-1}$. A: $n = 8$; B: $n = 16$; C: $n = 32$; D: error in the $n = 32$ case. In A and B, $f(x)$ is denoted by the dotted curve.



EXAMPLE 4.16

This time, we use a Chebyshev polynomial approximation to the same f as in the previous example, again using $n = 8$, 16, and 32 degree polynomials. Figures 4.45 A–C show plots of both f and the least squares approximation; Figure 4.45D shows the error in the 32-degree approximation. The performance of the Chebyshev and Legendre approximations are very similar, with the Chebyshev being very slightly better.

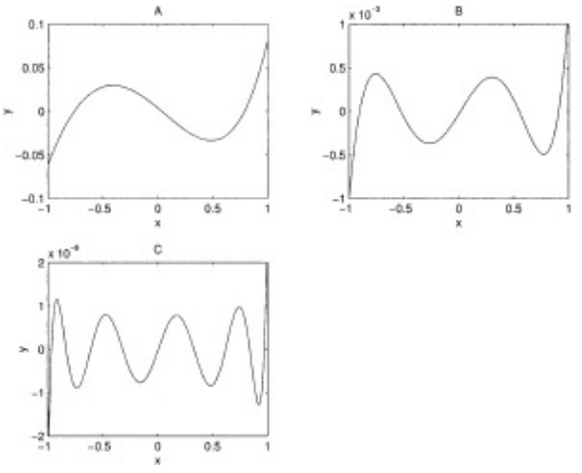
Figure 4.45 Chebyshev least squares approximation to $f(x) = 1/(1 + 25x^2)^{-1}$. A: $n = 8$; B: $n = 16$; C: $n = 32$; D: error in the $n = 32$ case. In A and B, $f(x)$ is denoted by the dotted curve.



EXAMPLE 4.17

Here we construct a Legendre polynomial approximation to e^x on the interval $[-1, 1]$, in much the same way as was done for $f(x) = (1 + 25x^2)^{-1}$ in Example 2. However, the accuracy here is so much greater that we plot the errors for the $n = 2$, 4, 8 cases in Figure 4.46.

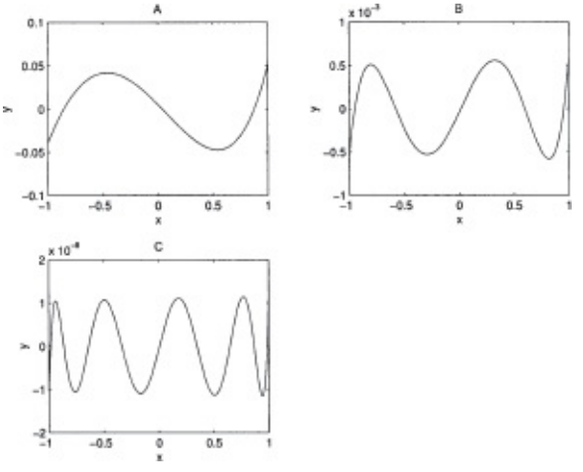
Figure 4.46 Error in Legendre least squares approximation to $f(x) = e^x$. A: error for $n = 2$; B: error for $n = 4$; C: error for $n = 8$.



EXAMPLE 4.18

Chebyshev approximation to e^x; This is the same as Example 4.17, except we use a Chebyshev expansion instead of a Legendre expansion.

Figure 4.47 Error in Chebyshev least squares approximation to $f(x) = e^x$. A: error for $n = 2$; B: error for $n = 4$; C: error for $n = 8$.



Several comments might be in order here. We first note how much easier it was to obtain a high degree of accuracy for the exponential than it was for $f(x) = (1 + 25x^2)^{-1}$. Second, although it might be difficult to discern in the plots, the results for the Chebyshev approximations were, in each case, very slightly better than for the Legendre approximations. A full discussion of this requires more mathematical machinery than we want to deal with right now, but it is generally true that Chebyshev least squares approximations are superior to those done with any other choice of basis.

Finally, note that to do a least squares approximation, we have to be able to compute the inner products, which are integrals. Thus, we need a tool like the trapezoid rule or, perhaps better, some of the more sophisticated methods to be developed in Chapter 5.

Exercises:

1. Modify the methods of §4.11.1 to compute the linear function of two variables that gives

the best least squares fit to the data for this exercise in Table 4.27.

2. The data in Table 4.26 gives the actual thermal conductivity data for the element iron. Construct a quadratic least squares fit to this data and plot both the curve and the raw data. How well does your curve represent the data? Is the fit improved any by using a cubic polynomial?

3. Repeat Problem 2, this time using the data for nickel from Problem 21 in §4.8.

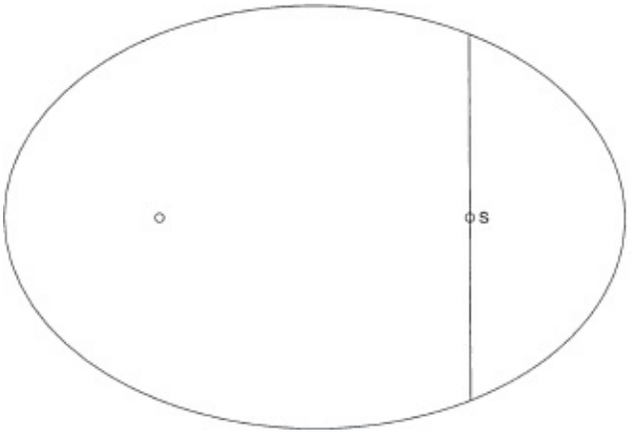
4. Modify the methods of §4.11.1 to compute the quadratic polynomial that gives the best least squares fit to the data in Table 4.27.

5. An astronomical tracking station records data on the position of a newly discovered asteroid orbiting the Sun. The data is reduced to measurements of the radial distance from the Sun (measured in millions of kilometers) and angular position around the orbit (measured in radians), based on knowledge of Earth's position relative to the Sun. In theory, these values should fit into the polar coordinate equation of an ellipse, given by

$$r = \frac{L}{2(1 + \epsilon \cos\theta)},$$

where $\in$ is the eccentricity of the elliptical orbit and L is the width of the ellipse (sometimes known as the *latus rectum* of the ellipse) at the focus. (See Figure 4.48.) However, errors in the tracking process and approximations in the transformation to (r, θ) values perturb the data. For the data in Table 4.28, find the eccentricity of the orbit by doing a least squares fit to the data. *Hint:* Write the polar equation of the ellipse as

Figure 4.48 Figure for Problem 5. The closed curve is the elliptical orbit, and the vertical line has length L.



$$2r(1 + \epsilon \cos\theta) = L,$$

which can then be written as

$$2r = \epsilon(-2r \cos\theta) + L,$$

so $y_k = 2r_k$ and $x_k = -2r_k \cos\theta_k$.

Table 4.26 Data for Problem 2.

Temperature (K), u	100	200	300	400	500
Conductivity (W/cm K), k	1.32	0.94	0.835	0.803	0.694
Temperature (K), u	600	700	800	900	1000
Conductivity (W/cm K), k	0.613	0.547	0.487	0.433	0.38

Table 4.27 Data for Problems 1 and 4.

Problem 4		Problem 1		
x_n	y_n	x_n	y_n	z_n
-1	0.9747	0	0	0.9573
0	0.0483	0	1	2.0132
1	1.0223	1	0	2.0385
2	4.0253	1	1	1.9773
3	9.0152	0.5	0.5	1.9936

6. Prove Theorem 4.8.

7. Let $w(x) = x$ on the interval [0, 1];compute $\|f\|_w$ for each of the following functions:

(a) e^{-x};

(b) $1/\sqrt{x^2+1}$;

(c) $1/\sqrt{x}$.

Compare to the values obtained using the unweighted 2-norm.

8. Derive the linear system (4.61) as the solution to the least squares approximation problem.

Table 4.28 Data for Problem 5.

θ_n	r_n
−0.1289	42895
−0.1352	42911
−0.1088	42851
−0.0632	42779
−0.0587	42774
−0.0484	42764
−0.0280	42750
−0.0085	42744
0.0259	42749
0.0264	42749
0.1282	42894

9. Provide the missing details to show that the family of polynomials defined in (4.63) is, indeed, orthogonal.

10. Let $\{\phi_k\}$ be a family of orthogonal polynomials associated with a general weight function w and an interval $[a, b]$. Show that the $\{\phi_k\}$ are independent in the sense that

$$0 = c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_n\phi_n(x)$$

holds for all $x \in [a, b]$ if and only if $c_k = 0$ for all k.

11. Prove the expansion formula (4.64) for a polynomial.

12. Construct the second-degree Legendre least squares approximation to $f(x) = \cos \pi x$ over the interval $[-1, 1]$.

13. Construct the second-degree Laguerre least squares approximation to $f(x) = e^x$ on the interval $[0, \infty)$.

14. Construct the third-degree Legendre least squares approximation to $f(x) - \sin \frac{1}{2}\pi x$ over the interval $[-1, 1]$.

◁ ● ● ● ▷

4.12 ADVANCED TOPICS IN INTERPOLATION ERROR

Based on the examples we have seen so far, the value of interpolation as an approximation tool seems unclear. When we applied it to the exponential function, we got excellent results, but when we applied it to the example $f(x) = (1 + 25x^2)^{-1}$ (known as the *Runge*[17] *example*), we got substantially less accuracy for the same number of nodes. This is not simply a case of using insufficiently many points in a particular example. It can be shown that if we took more and more nodes and used higher-degree interpolating polynomials, the error in interpolating to the Runge example would continue to get larger and larger, when measured in the norm $\|f - p_n\|_\infty$. In fact, the trend that is suggested in Figures 4.3B and C continues: For values of x in the middle of the interval $[-1, 1]$, $|f(x) - p_n(x)|$ actually goes to zero, but near the ends of the interval the polynomial interpolates oscillate wildly and do not converge. What is going on here?

A complete answer, although mathematically very interesting, is beyond our scope here, and so we will skip the details and present only the broader ideas. We also look at a several ways in which the potential problems with interpolation at equally spaced nodes can affect a calculation, and derive a better set of interpolating nodes.

4.12.1 Stability of Polynomial Interpolation

One problem with polynomial interpolation using high-degree polynomials is that it really is a potentially unstable process, in the sense that small changes to the data can lead to large changes in the interpolating polynomial. To see this, consider the effects of rounding error on the interpolation problem. Let $f(x)$ be the exact function that we wish to interpolate, and let $\hat{f}(x)$ be the function polluted by rounding error, and assume that $f(x) - \hat{f}(x) = \epsilon(x)$, where we assume that

$$\|f - \hat{f}\|_\infty = \|\epsilon\|_\infty = \epsilon_0$$

for some small ϵ_0.

The polynomial interpolate that we compute is based on the function that is polluted by rounding error, so we have the Lagrange form

$$\hat{p}_n(x) = \sum_{i=0}^{n} L_i^{(n)}(x)\hat{f}(x_i),$$

and we want to estimate the error $f(x) - \hat{p}(x)$. For convenience we define the "ideal" interpolate, based on the exact function, f, as

$$p_n(x) = \sum_{i=0}^{n} L_i^{(n)}(x) f(x_i).$$

Theorem 4.3 gives us the error between f and p_n:

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

We can relate this to the error we want to bound as follows:

$$f(x) - \hat{p}_n(x) = \underbrace{(f(x) - p_n(x))}_{\text{Error due to interpolation}} + \underbrace{(p_n(x) - \hat{p}_n(x))}_{\text{Error due to rounding}}.$$

We now turn our attention to analyzing the error due to rounding. We have

$$\begin{aligned} |p_n(x) - \hat{p}_n(x)| &= \left| \sum_{i=0}^{n} L_i^{(n)}(x)(f(x_i) - \hat{f}(x_i)) \right|, \\ &\leq \|\epsilon\|_\infty \sum_{i=0}^{n} \|L_i^{(n)}\|_\infty, \\ &= \epsilon_0 \sum_{i=0}^{n} \Lambda_i, \end{aligned}$$

where the norms are both taken over the interval defined by the nodes, and $\Lambda_i = \|L_i^{(n)}\|_\infty$. Let's assume that the rounding error is bounded and small: a very reasonable assumption based on our work in Chapter 1. Thus, we take $\|\epsilon\|_\infty = \epsilon_0 = \mathcal{O}(\mathbf{u})$.

We appear to be in good shape, having bounded the error due to rounding by something small times a sum depending on the norm of the Lagrange functions. The problem is that this sum can grow quite large as n increases. A general theorem (see Theorem 4.6 of [15]) is not necessary, since we can get the essentials by being experimental. Assume that the nodes are equidistant on the interval $[a, b]$, with $x_0 = a$ and $x_n = b$, and with $x_{j+1} - x_j = h$ being the uniform mesh spacing. Then we can write

$$x_j = a + jh, \quad 0 \leq j \leq n,$$

for each node, and

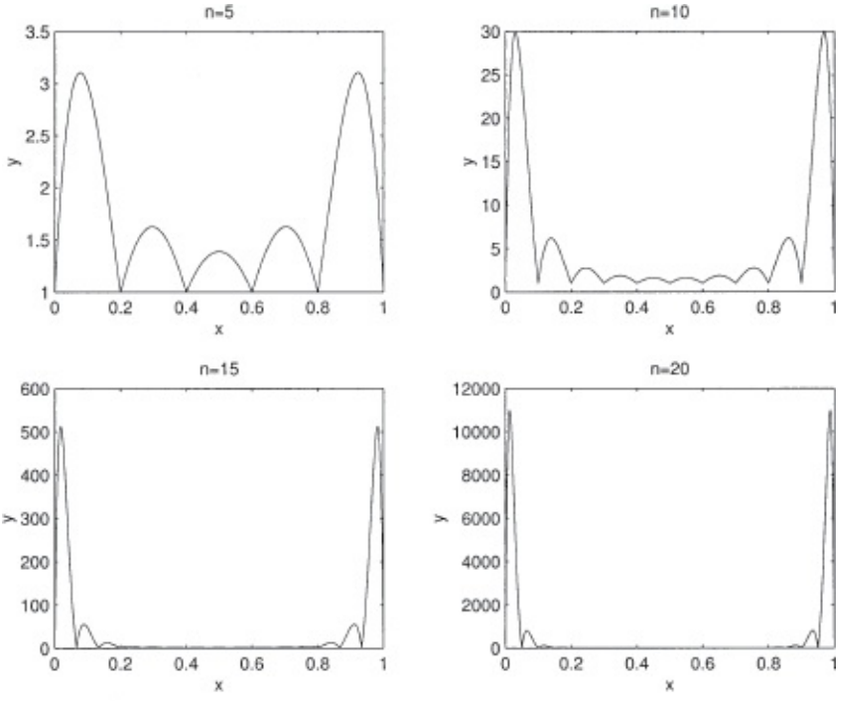
$$x = a + \eta_x h, \quad 0 \leq \eta_x \leq n,$$

where $\eta_x = (x - a)/h$ takes on *real* values between 0 and n. Therefore,

$$\begin{aligned} L_i^{(n)}(x) &= \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k}, \\ &= \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{\eta_x - k}{i - k}. \end{aligned}$$

Hence, the Lagrange functions (and, therefore, the Λ_i) are not dependent on the choice of a, b, or h. They depend entirely on n, η_x (which depends on x), and the distribution of the nodes. This means that we ought to be able to plot them rather easily and get some idea of how large that sum term can be. Figure 4.49 shows plots of

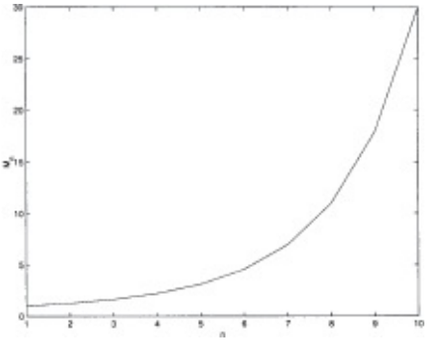
Figure 4.49 Plots of $L(x)$ versus x for $n = 5, 10, 15, 20$.



$$L(x) = \sum_{i=0}^{n} |L_i^{(n)}(x)|$$

for various values of n (assuming equally spaced points on the interval [0, 1]), and Figure 4.50 shows a plot of

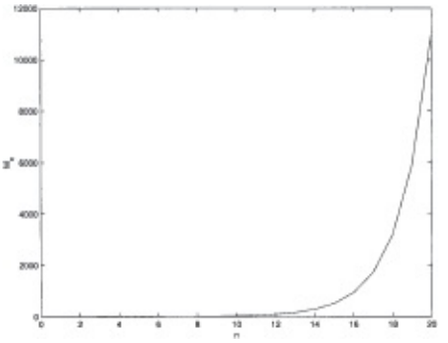
Figure 4.50 Plot of M_n versus n, for $n \leq 10$.



$$M_n = \sum_{i=0}^{n} \Lambda_i$$

as a function of n. Note the *rapid* growth of M_n. If we take larger values of n, the trend continues and the growth becomes clearly exponential, as Figure 4.51 shows.

Figure 4.51 Plot of M_n versus n, for $n \leq 20$.



What does this mean for our original interpolation problem? Recall how the overall error was related to the error in interpolation and the error in rounding:

$$f(x) - \hat{p}_n(x) \quad = \quad (f(x) - p_n(x)) + (p_n(x) - \hat{p}_n(x)) \, ;$$

from which it follows that

(4.65) $$\|f - \hat{p}_n\|_\infty \leq \|f - p_n\|_\infty + \|p_n - \hat{p}_n\|_\infty \leq \|f - p_n\|_\infty + \epsilon_0 M_n .$$

Since we now know that M_n can become quite large, we know that the presence of the small ϵ_0 multiplying the rounding-error term is *not* enough to guarantee that the overall computation will not be badly corrupted by rounding error. On the other hand, if we keep the degree of the polynomial interpolation less than, say, seven, then we have that $M_n \leq 10$ (see Figure 4.50) and we know the overall error is not seriously corrupted by rounding error.

Note that we are not claiming that polynomial interpolation of high degree *has* to be affected by large amounts of rounding error; the inequality in (4.65) goes the wrong way to support that conclusion. However, the estimate (4.65) does *allow* for large amounts of rounding error whenever n is large enough that M_n is very large, and we can avoid this potential problem by taking n so that M_n is small. Thus, we avoid using polynomial interpolates of degree much higher than 7 or so.

4.12.2 The Runge Example

(*Note:* This section of the text requires a bit more background than the others in this chapter, especially in complex arithmetic.)

One is tempted to look at the function $f(x) = (1 + 25x^2)^{-1}$ and say that it is a smooth, well-behaved functon for all x, and this is indeed the case, so long as x is real. But if we allow x to be imaginary, then problems can occur; consider the value of $f(i/5)$, where $i = \sqrt{-1}$. Why does this matter, if we are considering the problem of interpolating to f as a function of the real variable x?

It turns out that it matters a great deal. It can be shown (see §3.4 of [11]) that for functions of the form $f(x) = (1 + a^2x^2)^{-1}$, the error in interpolation can be expressed as

$$f(x) - p_n(x) = \left(\frac{r_n}{1 + a^2 x^2} \right) \left(\frac{w_n(x)}{w_n(ia^{-1})} \right) ,$$

where $r_n = x$ if n is even, $r_n = ia^{-1}$ if n is odd, and

$$w_n(x) = \prod_{k=0}^{n}(x - x_k).$$

Since

$$\left|\frac{r_n}{1 + a^2x^2}\right| \leq C$$

for all x and n, the question of convergence then comes down to the size of the ratio

$$R_n(x) = \left|\frac{w_n(x)}{w_n(ia^{-1})}\right|.$$

Now we can write

$$R_n(x) = \left|\frac{\sigma_n(x)}{\sigma_n(ia^{-1})}\right|^{n+1}$$

for

$$\sigma_n(z) = \left(\prod_{k=0}^{n}|z - x_k|\right)^{\frac{1}{n+1}}.$$

But we can take logarithms of both sides to get that

$$\log \sigma_n(z) = \frac{1}{n+1}\sum_{k=0}^{n}\log|z - x_i|.$$

Now, if the nodes are equally spaced on the interval [−1, 1], the sum can be interpreted as a Riemann sum for the integral

$$I(f) = \frac{1}{2}\int_{-1}^{1}\log|z - t|dt;$$

therefore, we can write

$$\log \sigma_n(z) \approx \frac{1}{2}\int_{-1}^{1}\log|z - t|dt,$$

and we define the right side of this as log σ(z), thus implicitly defining σ(z):

$$\frac{1}{2}\int_{-1}^{1}\log|z - t|dt = \log \sigma(z).$$

The integral can be evaluated, but we have to be careful about doing it, since $z = x + iy$ is complex. Thus, we have

$$\frac{1}{2}\int_{-1}^{1}\log|z - t|dt = \frac{1}{2}\int_{-1}^{1}\log\sqrt{(x-t)^2 + y^2}dt = \frac{1}{4}\int_{-1}^{1}\log[(x-t)^2 + y^2]dt.$$

Careful application of integration by parts yields

$$\begin{aligned}\frac{1}{2}\int_{-1}^{1}\log|z - t|dt &= \frac{1}{4}(1-x)\log((x-1)^2 + y^2) + \frac{1}{4}(1+x)\log((x+1)^2 + y^2)\\ &- \frac{1}{2}y\arctan\frac{x-1}{y} + \frac{1}{2}y\arctan\frac{x+1}{y}.\end{aligned}$$

Thus,

$$\begin{aligned}\sigma_n(z) \approx \sigma(z) &= e^{-1}\left((x-1)^2 + y^2\right)^{(1-x)/4}\left((x+1)^2 + y^2\right)^{(1+x)/4}\\ &\times \exp\left\{\frac{1}{2}y\left(\arctan\frac{x+1}{y} - \arctan\frac{x-1}{y}\right)\right\}.\end{aligned}$$

Now the error satisfies

$$|f(x) - p_n(x)| = |R_n(x)| \approx C_n \left| \frac{\sigma(x)}{\sigma(ia^{-1})} \right|^{n+1},$$

so convergence depends, in the limit as $n \to \infty$, on the ratio $|\sigma(x)/\sigma(ia^{-1})|$. For real x we get that

$$\sigma(x) = e^{-1}(x-1)^{(1-x)/2}(x+1)^{(1+x)/2},$$

whereas for imaginary arguments we get

$$\sigma(iy) = e^{-1}\sqrt{1+y^2}\exp\left\{ y \arctan\frac{1}{y} \right\}.$$

To get convergence on the entire interval $[-1, 1]$, we need to have

$$\max_{x\in[-1,1]} |\sigma(x)| = 2e^{-1} = 0.7357588824 < |\sigma(ia^{-1})|.$$

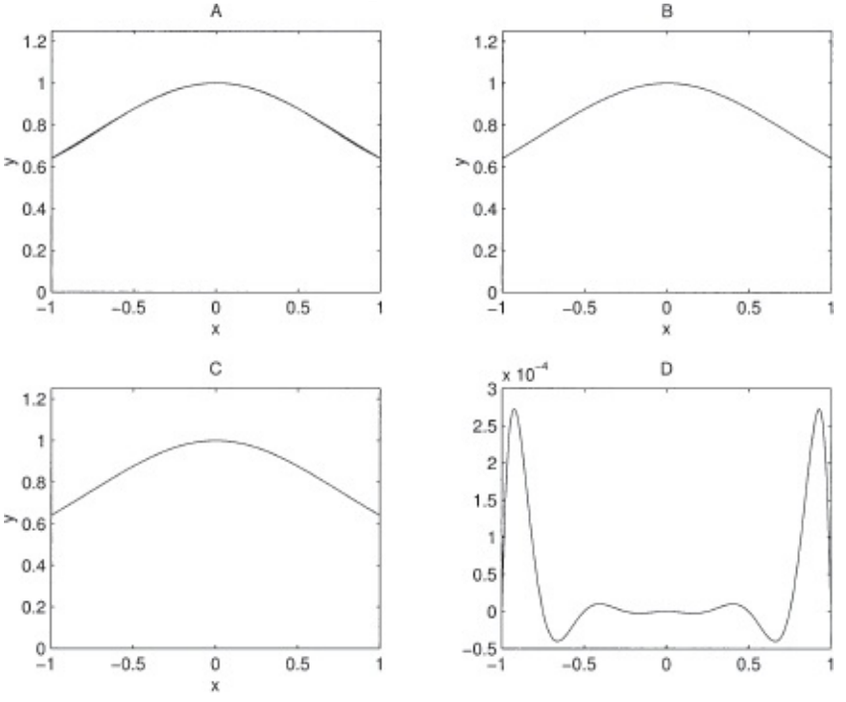
An elementary computation shows that $|\sigma(i/5)| = 0.4937581336\ldots$; therefore, we will not get convergence on the entire interval $[-1, 1]$ with $a = 5$, because

$$\frac{\max_{x\in[-1,1]} |\sigma(x)|}{\sigma(i/5)} = \frac{0.7357588824}{0.4937581336} > 1.$$

By decreasing a, we can finally get $0.7357588824 < |\sigma(ia^{-1})|$. For example, for $a = 3/4$ we have $\sigma(i/a) = 1.446\ldots$.

What this shows is that the problem really is that the *singularities* of the function (i.e., the points $\pm i/5$) are "too close" to the interval of interpolation. If we looked at $g(x) = (1 + a^2x^2)^{-1}$ for a smaller value of a, then we would get convergence on the entire interval from $[-1, 1]$. An example of this is given in Figure 4.52, using g with $a = 3/4$, and $n = 4$, 8, 16 nodes; the last plot shows the error for the $n = 16$ case. See also Problem 3. Note that the error still shows the development of "spikes" near the endpoints; the difference is that in this case, as $n \to \infty$, the amplitude of the spikes will eventually decay to zero.

Figure 4.52 Successful interpolation to the Runge-like function, $g(x) = 1/(1 + (9/16)x^2)^{-1}$. A: 4 nodes; B: 8 nodes; C: Error in 16-node interpolation.



It is also possible to work on a more arbitrary interval $[-b, b]$, in which case we learn that the length of this interval also plays a role in the convergence of the interpolation. (See [9] for a discussion of this.) Essentially, convergence will occur if $1 > Cab$, where $C = 0.5255\ldots$.

4.12.3 The Chebyshev Nodes

In §4.12.1 and §4.12.2 we saw that high-degree polynomial interpolation *at equidistant points* can be a bad idea. Is there an alternate choice for the distribution of nodes that will produce better results? The answer is "yes," and the nodes in question are called the *Chebyshev nodes*.

Define the family of functions $T_n(x)$, $n \geq 0$, by the formula

(4.66) $$T_n(x) = \cos(n \arccos x), \quad x \in [-1, 1].$$

Remarkably, these functions are polynomials (in fact, they are the Chebyshev polynomials introduced back in §4.11.2).

Theorem 4.10 *The functions $T_n(x)$ satisfy the following:*

1. *Each T_n is a polynomial of degree n;*
2. *For $n \geq 1$, $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$;*
3. *$T_n(x) = 2^{n-1}x^n$ + lower-order terms.*

Proof: We observe first that both (1) and (3) will follow quickly from (2), so we start by proving (2). Write $x = \cos\theta$ so that (4.66) becomes

$$T_n(x) = \cos n\theta.$$

Elementary trigonometry tells us that

$$\cos n\theta \cos\theta = \frac{1}{2}\left(\cos(n+1)\theta + \cos(n-1)\theta\right),$$

which can be solved to get

$$\cos(n+1)\theta = 2\cos\theta\cos n\theta - \cos(n-1)\theta,$$

from which (2) follows, using $T_n(x) = \cos n\theta$, and so on.

Now, to prove (1), we note that we trivially have

$$\cos(n \arccos x) = 1$$

for $n = 0$, and

$$\cos(n \arccos x) = x$$

for $n = 1$. Now assume that $\cos(n \arccos x)$ is a polynomial for all $n \leq k$, and an inductive argument based on the recursion in (2) quickly finishes the proof.

The final part follows more or less directly from (2). We have that $T_0(x) = 1$, and $T_1(x) = x$, so that the recursion yields

$$T_2(x) = 2xT_1(x) - T_0(x) = 2x(x) - 1 = 2x^2 - 1,$$

and we can use induction to formally prove it for all n (see Problem 5). •

It should be noted that all orthogonal polynomial families satisfy a three-term recurrance relation similar to the one in Theorem 4.10, part 2.

Assume now (for convenience) that we are only interested in constructing approximations on

the interval [−1, 1]. As a polynomial of degree n, each T_n will have exactly n roots; we can show (it actually follows from Theorem 4.9) that these roots will be distinct and all lie in the interval [−1, 1]. Denote, then, the roots of the n-degree polynomial T_n as $z_k^{(n)}$, and note that (4.66) implies that

$$z_k^{(n)} = \cos\left(\frac{(2k-1)\pi}{2n}\right), \quad 1 \le k \le n. \tag{4.67}$$

The Chebyshev nodes are simply the roots of the polynomials T_n. To construct an n-degree polynomial interpolate, use the $n + 1$ roots of T_{n+1}; i.e., take the interpolation nodes to be

$$x_k = z_{k+1}^{(n+1)}, \quad 0 \le k \le n,$$

so that we have the following.

Definition 4.4 (Chebyshev Nodes)

$$x_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right), \quad 0 \le k \le n. \tag{4.68}$$

Below, we will show why this works, but first let's look at some examples.

EXAMPLE 4.19

Let $f(x) = e^x$ on the interval [−1, 1]. The Chebyshev nodes for the case $n = 1$ (linear interpolation) are given by

$$x_0 = \cos\left(\frac{(2\times 0+1)\pi}{2(1+1)}\right) = \cos\frac{\pi}{4} = \frac{1}{2}\sqrt{2}$$

and

$$x_1 = \cos\left(\frac{(2\times 1+1)\pi}{2(1+1)}\right) = \cos\frac{3\pi}{4} = -\frac{1}{2}\sqrt{2},$$

so the linear interpolate to the exponential is given by

$$p_1(x) = 1.260591837 + 1.085441641x.$$

EXAMPLE 4.20

Let's now look at higher-degree interpolation to the exponential; specifically, we will consider fourth-degree interpolation. The Chebyshev nodes now are

$$x_0 = \cos\left(\frac{(2\times 0+1)\pi}{2(4+1)}\right) = \cos\frac{\pi}{10}$$

$$x_1 = \cos\left(\frac{(2\times 1+1)\pi}{2(4+1)}\right) = \cos\frac{3\pi}{10}$$

$$x_2 = \cos\left(\frac{(2\times 2+1)\pi}{2(4+1)}\right) = \cos\frac{5\pi}{10}$$

$$x_3 = \cos\left(\frac{(2\times 3+1)\pi}{2(4+1)}\right) = \cos\frac{7\pi}{10}$$

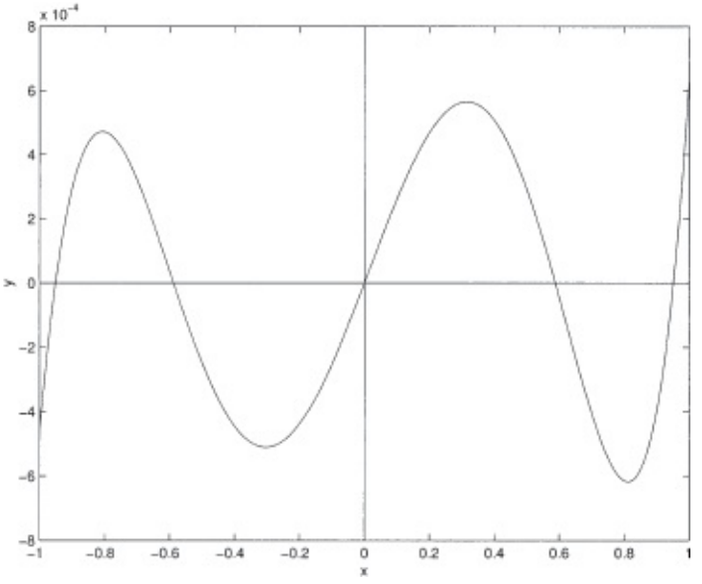
$$x_4 = \cos\left(\frac{(2\times 4+1)\pi}{2(4+1)}\right) = \cos\frac{9\pi}{10}.$$

Note that these values are somewhat less simple to work with than the integer or simple fraction nodes we have generally used. But the computer doesn't really care about that, and the Newton algorithm yields the polynomial

$$p_4(x) = 1 + 0.997317240x + 0.4995561859x^2 + 0.177334621x^3 + 0.043434107x^4.$$

Figure 4.53 shows the plot of the error in this interpolation, over the interval [−1, 1]. Compare it to Figure 4.2B, which shows the plot of the same error for fourth degree interpolation using equally spaced points. It is difficult to tell because of the scales used, but the Chebyshev interpolation is slightly better in terms of the maximum error.

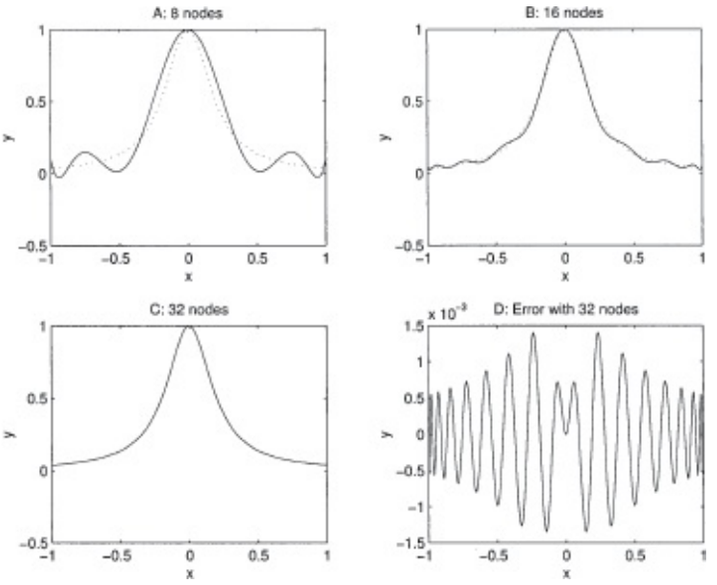
Figure 4.53 Error in fourth degree Chebyshev interpolation to the exponential.



EXAMPLE 4.21

Consider again $f(x) = (1 + 25x^2)^{-1}$, the Runge example. If we use the Chebyshev nodes to compute interpolating approximations of degree 8, 16, and 32, we get the plots shown in Figure 4.54. Note that the accuracy of the approximation is much better than was the case using equally spaced nodes, as in Figure 4.3. Figure 4.54C shows that, with only 32 nodes, we cannot distinguish by eye between the interpolate and the original function. Figure 4.54D shows the error in the 32-degree approximation.

Figure 4.54 Chebyshev interpolation to the Runge example.



Why do these nodes work better? A complete answer would have to be phrased in the vague terms that we used in §4.12.2 to discuss the Runge example, or we would have to bring in a lot of additional mathematical machinery.

A fairly decent answer, however, can be based on the basic interpolation error theorem.

Theorem 4.11 *Let $f \in C^{n+1}([-1, 1])$ be given, and let p_n be the n-degree polynomial interpolate to f, using the Chebyshev nodes (4.68). Then,*

$$\|f - p_n\|_\infty \leq \frac{1}{2^n(n+1)!}\|f^{(n+1)}\|_\infty.$$

Proof: Recall that we have

$$f(x) - p_n(x) = \frac{w_n(x)}{(n+1)!}f^{(n+1)}(\xi_x),$$

where

$$w_n(x) = \prod_{i=0}^{n}(x - x_i).$$

Since polynomials are uniquely defined by their roots (up to a constant multiple), and since w_n and T_{n+1} have the same roots, it follows that for some constant c_n. Now, w_n is what is known as a *monic polynomial*, i.e., the coefficient of the highest-order term is 1:

$$w_n(x) = x^{n+1} + \text{lower-order terms.}$$

Therefore, c_n is the *reciprocal* of the leading coefficient for T_{n+1}. Therefore, from Theorem 4.10, we have that

$$c_n = 2^{-n}.$$

But each Chebyshev polynomial is a cosine on the interval $[-1, 1]$; this means that

$$|T_{n+1}(x)| \leq 1, \quad -1 \leq x \leq 1.$$

Therefore,

$$|w_n(x)| = |c_n T_{n+1}(x)| \leq 2^{-n}$$

for all $x \in [-1, 1]$, and the proof is complete. •

What the theorem (and its proof) shows is that the choice of the Chebyshev nodes allows us

to bound the factor in the error theorem that depends on w_n. It can be shown, in fact, that any other choice of nodes would lead to a larger $\|w_n\|_\infty$. so in this sense the Chebyshev nodes are optimal. They are not "perfect," however. One can find functions for which the Chebyshev nodes fail, but these examples are much more involved than the Runge example.

Exercises:

1. Use your computer's random number function to generate a set of random values r_k, $0 \le k \le 8$, with $|r_k| \le 0.01$. Construct the interpolating polynomial to $f(x) = \sin x$ on the interval $[-1, 1]$, using nine equally spaced nodes; call this $p_8(x)$. Then construct the polynomial that interpolates $f(x_k) + r_k$; call this $\hat{p}_8(x)$. How much difference is there between the divided difference coefficients for the two polynomials? Plot both p_8 and $\hat{p}_8$ and comment on your results. (*Note:* It is important here that you look at x values *between* the nodes. Do not produce your plots based simply on the values at the nodes.)

2. Repeat the above, using $f(x) = e^x$.

3. Use the Newton interpolating algorithm to construct interpolating polynomials of degree 4, 8, 16, and 32 using equally spaced nodes on the interval $[-1, 1]$, to the function $g(x) = (1 + 4x^2)^{-1}$. Is the sequence of interpolates converging to g throughout the interval?

4. Use the Newton interpolating algorithm to construct interpolating polynomials of degree 4, 8, 16, and 32 using equally spaced nodes on the interval $[-1, 1]$, to the function $g(x) = (1 + 100x^2)^{-1}$. Is the sequence of interpolates converging to g throughout the interval?

5. Write up a complete proof of Theorem 4.10, providing all the details omitted in the text.

6. Construct interpolating polynomials of degree 4, 8, 16, and 32 on the interval $[-1, 1]$ to $f(x) = e^x$ using equidistant and Chebyshev nodes. Sample the respective errors at 500 equally spaced points and compare your results.

7. The Chebyshev nodes are defined on the interval $[-1, 1]$. Show that the change of variable

$$t_k = a + \frac{1}{2}(b - a)(x_k + 1)$$

will map the Chebyshev nodes to the interval $[a, b]$.

8. Use the formula in Problem 7 to find the Chebyshev nodes for linear interpolation on the interval $[\frac{1}{4}, 1]$. Use them to construct a linear interpolate to $f(x) = \sqrt{x}$. Plot the error in this interpolation. How does it compare to the error estimate used in §3.7?

9. Repeat the above for $f(x) = x^{-1}$ on the interval $[\frac{1}{2}, 1]$.

10. What is the error estimate for Chebyshev interpolation on a general interval $[a, b]$? In other words, how does the change of variable affect the error estimate?

4.13 LITERATURE AND SOFTWARE DISCUSSION

The subject of interpolation and approximation is a very broad one, with an extensive literature. In fact, approximation theory is a significant area of mathematical research, generally distinct from (although obviously related to) numerical analysis. The classic text on interpolation and approximation is Davis's book [6]; more recent and equally good works are those by Cheney [3], Powell [13], and Rivlin [15]. The recent book by Trefethen [18] should also be considered. Generally, approximation theory is distinct from interpolation, which is one reason that Davis's book is so useful, since it covers both. Modern work in what might be called computational approximation tends to center on splines, which is why de Boor's book [8], although old, is a good reference. A rather large collection of FORTRAN codes is given there, which can be found at NETLIB under the library `pppack`; MATLAB has incorporated many of them into their Splines Toolbox. Another standard spline reference is [1].

Several interesting ideas were not discussed here (the subject of interpolation and approximation is rich enough that an entire course could be based on this one subject). In Padé approximation, we look for *rational function* (i.e., ratios of polynomials) approximations to given functions. This is only marginally more involved than polynomial approximation, but of course allows for more general functional behavior to be approximated accurately, since rational functions can exhibit asymptotic behavior, but polynomials cannot. A good introduction to Padé approximation is given by Cheney in Chapter 5 of [3].

Another missing topic is Fourier (i.e., trigonometric) approximation. Again, an entire course could be based on this subject, and this is done in many engineering departments. Wavelets, related to trigonometric approximation, is also probably not appropriate for a text at this level. Future editions may have to re-think these two omissions.

REFERENCES

1. Ahlberg, J. H., Nilson, E. N., and Walsh, J. L., *The Theory of Splines and Their Applications*, Academic Press, New York, 1967.

2. Birkhoff, Garrett; "Fluid dynamics, reactor computations, and surface representation," in *A History of Scientific Computation* (Steve Nash, editor), 1990.

3. Cheney, E.W., *Approximation Theory*, Chelsea, New York, 1966.

4. Cline, A. K. "Scalar- and planar-valued curve fitting using splines under tension," Commun. ACM vol. 17, pp. 218–220 (1974).

5. Davis, Paul; "B-splines and geometric design," *SIAM News*, vol. 29, no. 5, 1996.

6. Davis, Philip, *Interpolation and Approximation*, Dover, New York, 1975; originally published by Blaisdell Publishing Company (1963).

7. de Boor, Carl, “On uniform approximation by splines,” *J. Approx. Theory*, vol. 1, pp. 219–235, 1968.

8. de Boor, Carl, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

9. Epperson, James, “On the Runge example,” *Am. Math. Monthly*, pp. 329–341, 1987.

10. Hall, C.A., “On error bounds for spline interpolation,” *J. Approx. Theory*, vol. 1, pp. 209–218, 1968.

11. Isaacson, E., and Keller, H. B., *Analysis of Numerical Methods*, Dover, New York, 1994; originally published by John Wiley & Sons, Inc., (1966).

12. Marušić, Miljenko, and Rogina, Mladen, “B-spline in tension,” *VII Conference on Applied Mathematics*, Univ. Osijek, Osijek, Yugoslavia, 1990, pp. 129–134.

13. Powell, M. J. D., *Approximation Theory and Methods*, Cambridge University Press, Cambridge, UK, 1981.

14. Prenter, Paddy, *Splines and Variational Methods*, Wiley-Interscience, New York, 1975.

15. Rivlin, Theodore J., *An Introduction to the Approximation of Functions*, Dover, New York, 1981; originally published by Blaisdell Publishing Company (1969).

16. Schoenberg, I. J., “Contributions to the problem of approximation of equidistant data by analytic functions,” *Quart. Appl. Math.*, vol. 4, pp. 45–99 and 112–141, 1946.

17. Schweikert, D. G., “An interpolating curve using a spline in tension,” *J. Math. Phys.*, vol. 45, pp. 312–317, 1966.

18. Trefethen, L. N., *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2012.

19. Young, David, “Garrett Birkhoff and applied mathematics,” *Notices AMS*, vol. 44, no. 11, pp. 1446–1449, 1997.

[1] The reader is reminded that the symbol Π is used for a product in the same way that $\sum$ is used for a sum. Thus

$$\prod_{k=1}^{n} a_k = a_1 \times a_2 \times a_3 \times \cdots \times a_n.$$

[2] Leopold Kronecker (1823–1891) was born in Liegnitz, Prussia, and educated in local schools and the University of Berlin, graduating with a doctorate in 1845. Most of Kronecker’s mathematical research was in what would now be called modem algebra and the theory of equations. In 1845, he left academia briefly to enter the business world, but he remained mathematically active even during this interlude. The fortune that he made while working as a banker enabled him to continue as a mathematician despite not having a regular professorship until 1883.

In his later years, Kronecker advocated a somewhat extreme philosophy of mathematics in which only finite numbers and finite operations were valid. The Kronecker delta notation came out of his work on determinants and matrices, done in 1869, under the title “Bemerkungen zur Determinanten-Theorie,” published in the *Journal für die reine und*

angewandte Mathematik.

[3]Joseph-Louis Lagrange (1736–1813) was born in Turin, Italy, of French parents. At a very young age he was made a professor at the Royal Artillery School in Turin. In 1766, he succeeded his friend and mentor Euler as the Director of the Mathematics Section of the Berlin Academy. In 1787 he accepted a similar position at the Paris Academy of Sciences, where he was active in establishing two very influential French schools, the École Polytechnique, and the École Normale. In his later years, he was granted a number of honors by the French emperor, Napoleon Bonaparte.

His results on interpolation were a major part of his work while in Berlin, but the interpolation formula ascribed to him here was not published until 1795, after he had moved to France. Ironically, Lagrange claims to have gotten the idea from some of Newton's work.

[4]The reader should note that all of these examples used equally spaced points to define the nodes. Other choices are possible, and there is a particular choice that is indeed better. (See §4.12.3.

[5]We have broken with convention and tradition in our divided difference notation here. Most texts use $f[x_k]$ for the zero-th divided difference, and $f[x_k, x_{k+1}]$ for the first divided difference formed by the nodes x_k and x_{k+1}, $f[x_k, x_{k+1}, x_{k+1}]$ for the second divided difference formed by x_k, x_{k+1}, and x_{k+2}, and so on. This becomes unwieldy when talking about higher and higher differences, and while it is slightly more general, most texts (including this one) do very little to justify that extra generality.

[6]Taken from tables in *Introduction to Thermodynamics: Classical and Statistical*, by R. E. Sonntag and G. J. Van Wylen, John Wiley & Sons, Inc., New York, 1971.

[7]Note that here we are treating x_2 as the "first" node, instead of x_0. This is possible because the order of the nodes is irrelevant in the computation, although this does affect the formulas somewhat.

[8]Note that we have assumed here that ξ_x is a differentiable function of x. This is indeed the case, but actually proving it is a nontrivial exercise that we omit for the sake of a brief and concise presentation.

[9]Note the change in indexing convention: We are now indexing from 1 to n, instead of from 0 to n.

[10]Charles Hermite (1822–1901) was born in Dieuze, France, on Christmas Eve. A poor test-taker, Hermite was not able to pass the exams for his bachelor's degree until the age of nearly 26. He held professional positions at a number of French schools, most notably the École Polytechnique, the École Normale, and the Sorbonne.

Perhaps his best-known mathematical result is the first proof that e is a transcendental

number (published in 1873). His name is attached to a number of mathematical ideas and concepts, including the Hermite differential equation, Hermite polynomials, and Hermitian matrices. The idea of interpolating to the derivative as well as to the function values was part of a paper he published in 1878, *Sur la formule d'interpolation de Lagrange*, which actually considers interpolation not only of the first derivative, but of higher derivatives as well. Hermite interpolation is sometimes called *oscillatory interpolation*.

[11]The history of the development of splines—and even the origin of the word itself—might be of interest to readers.

Many years ago, one of the author's students suggested that the word "splines" was constructed from "spliced lines;" while this is indeed an interesting observation, the truth of the matter is that the mathematical use of the word "spline" comes from the days when being an engineer often meant being an accomplished draftsman. One draftsman's tool was known as a "spline," a thin, very flexible piece of metal that was used to help draw a smooth curved line based on a few discrete points marked on a drawing.

One of the historical uses of splines was in the automobile industry, where they were used to define the shape of sheet-metal pieces for car bodies. Much of the mathematical development work was done at General Motors in the early 1960s, and also at Renault and Citroen in France, at about the same time. Curiously, one of the original papers on splines [16] was written with applications to actuarial data in mind. Some of the impetus for the development of the theory also came from the British aircraft industry during World War II, as well as the ship-building industry.

The work at GM is detailed nicely by Birkhoff [2] and in the retrospective by Young [19]. Paul Davis summarized some of this material in *SIAM News* in 1996; see [5]. Carl DeBoor maintains a spline bibliography on the Internet at http://www.cs.wisc.edu/~deboor/bib/bib.html.

[12]The correct terminology is to say that B is a function of "compact support," but using the phrase "locally defined" conveys the meaning more clearly at this level.

[13]Adrien-Marie Legendre (1752–1833) was born and educated in Paris. He contributed greatly to what we now call number theory as well as elliptic function theory. He was the first to publish, in 1805, a description of the method of least squares, although it appears that Gauss had previously worked out much of the same material.

Legendre introduced the polynomials that bear his name in a 1785 paper on the gravitational attraction of spherical bodies. They arise in this context because they are the solutions to an ordinary differential equation that occurs as part of the solution process for the equations of motion in a spherical coordinate system.

[14]Pafnuty Lvovich Chebyshev (1821–1894) was born near Borovsk, Russia, southwest of Moscow, and educated at the University of Moscow, from which he was graduated in 1841. From 1847 until his death he lived in St. Petersburg. Many areas of mathematics, from

number theory to the theory of equations, were touched by Chebyshev, but he is perhaps best known, at least in applied mathematics, for his work in approximation theory.

Because of the many different ways that the Russian alphabet can be transliterated into the Roman alphabet, there are several different ways to spell Chebyshev's name. The most common alternative is "Tschebyscheff." A marvelously engaging discussion of the perils of transliterating between the Roman and the Russian (Cyrillic) alphabets, as well as a substantial treatment of Chebyshev's life, is contained in *The Thread*, by Philip J. Davis, a leading mathematician in the area of interpolation and approximation.

[15]Charles Hermite (1822–1901) was born in Dieuze, France, on Christmas Eve. A poor test-taker, Hermite was not able to pass the exams for his bachelor's degree until the age of nearly 26. He held professional positions at a number of French schools, most notably the École Polytechnique, the École Normale, and the Sorbonne.

Perhaps his best-known mathematical result is the first proof that e is a transcendental number (published in 1873). His name is attached to a number of mathematical ideas and concepts, including the Hermite differential equation, Hermite polynomials, and Hermitian matrices. The idea of interpolating to the derivative as well as to the function values was part of a paper he published in 1878, *Sur la formule d'interpolation de Lagrange*, which actually considers interpolation not only of the first derivative, but of higher derivatives as well. Hermite interpolation is sometimes called *osculatory interpolation*.

[16]Edmond Nicolas Laguerre (1834 – 1886) was born and died in Bar-le-Duc, France. He was educated at the École Polytechnique but did not graduate with a high ranking. He served in the military as an artillery officer for 10 years before returning to the École as a faculty member, where he remained for the rest of his life. Although best known for the orthogonal polynomials that bear his name, and their associated differential equation, Laguerre also published work in analysis, geometry, and abstract linear spaces.

[17]Carl Runge (1856–1927) was born in Bremen and educated at the University of Munich. He originally intended to study literature but after less than two months switched to mathematics and physics. In 1877 he began attending the University of Berlin, where he received a doctorate in 1880, on differential geometry. He held academic positions at Hanover and Göttingen and retired in 1925. Much of his professional work was more in physics than in mathematics, but he did make contributions in the numerical solution of differential equations (Runge–Kutta methods) and polynomial interpolation theory. The paper in which he outlined the theory behind the so-called "Runge example" appeared in 1901, under the title *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten*.