Chapter 6
Interpolation and Splines

6.1 Introduction

In basic terms, interpolate means to estimate function values between known values. Prior to modern technology, interpolation in standard tables of tabulated functions was used to compute values for arguments not tabulated. Today, table interpolation of known functions has been virtually eliminated by modern calculators and computers; however, the concept of interpolation is still important both in the analysis of experimental data and in the mathematical development of various numerical schemes.

As an example, consider the following data which describes the saturated vapor pressure of water as a function of temperature.

Temperature (°C)	30	40	60	80	100	150
Pressure (atm)	0.0418	0.0728	0.196	0.467	1.00	4.69

Estimating the pressure at other temperatures is the goal of interpolation. If we wish to estimate the pressure at 90°C, a line between the points $(80,\ 0.467)$ and $(100,\ 1.00)$,

$$P - 0.467 = \left(\frac{1.00 - 0.467}{100 - 80}\right)(T - 80),$$

will provide an approximation. In this chapter we will focus our efforts on polynomial methods, lines, parabolas, etc., to compute interpolated values.

Suppose we have a set of data points $(x_i,\ y_i)$ for $i = 1$ to n. The data may be from an experiment or a known function so that $y_i = f(x_i)$. Our goal is to build an *interpolating* function, say $F(x)$, to approximate or model the given data. The interpolation requirements are specified by the equations $F(x_i) = y_i$ for $i = 1$ to n. In basic geometric terms, a graph of the function $F(x)$ must pass through the n data points.

Typically, $F(x)$ has the form

$$F(x) = c_1 g_1(x) + c_2 g_2(x) + c_3 g_3(x) + \ldots + c_n g_n(x) \tag{6.1}$$

where the $g_j(x)$'s are known linearly independent functions.

A set of functions $\{g_j(x)\}$ is said to be *linearly dependent* on an interval if there exists a set of constants $\{c_j\}$, not all zero, such that

$$c_1 g_1(x) + c_2 g_2(x) + c_3 g_3(x) + \ldots + c_n g_n(x) = 0 \tag{6.2}$$

for all x in the interval. This means that at least one function is a linear combination of others. For example, $g_3(x)$ may equal $4g_1(x) + 7g_5(x)$. The set of functions $\{g_j(x)\}$ is said to be *linearly independent* on the interval if the set is not linearly dependent. This means that (6.2) will be satisfied for all x only if all of the $c_i = 0$. This implies that no function in the set may be expressed as a linear combination of other functions in the set.

Matrix notation will simplify our development of interpolation. Equation (6.1) may be written in matrix form as follows:

$$[\,F(x)\,] = [\,g_1(x) \quad g_2(x) \quad g_3(x) \quad \ldots \quad g_n(x)\,]\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix}. \tag{6.3}$$

With the $g_j(x)$'s specified, the interpolation requirement $F(x_i) = y_i$ results in

$$[\,F(x_i)\,] = [\,g_1(x_i) \quad g_2(x_i) \quad g_3(x_i) \quad \ldots \quad g_n(x_i)\,]\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = [\,y_i\,].$$

Since $[\,F(x_i)\,]$ and $[\,y_i\,]$ are both 1×1, the matrix notation could be eliminated; however, combining the results for each x_i will result in an $n \times n$ linear system as follows:

$$\begin{bmatrix} F(x_1) \\ F(x_2) \\ F(x_3) \\ \vdots \\ F(x_n) \end{bmatrix} = \begin{bmatrix} g_1(x_1) & g_2(x_1) & g_3(x_1) & \cdots & g_n(x_1) \\ g_1(x_2) & g_2(x_2) & g_3(x_2) & \cdots & g_n(x_2) \\ g_1(x_3) & g_2(x_3) & g_3(x_3) & \cdots & g_n(x_3) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_1(x_n) & g_2(x_n) & g_3(x_n) & \cdots & g_n(x_n) \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

In matrix form we have

$$\boldsymbol{Gc} = \boldsymbol{y} \tag{6.4}$$

where $G_{ij} = g_j(x_i)$. The solution of (6.4) provides the coefficients for our interpolating function, $F(x)$, specified in (6.1). Keep in mind that any appropriate functions may be selected for the $g_j(x)$'s. In this chapter we will discuss how to construct several families of $g_j(x)$'s.

6.2 Lagrange Interpolating Polynomials

The development in Section 6.1 was in very general terms. To simplify our development, assume that the data set consists of two points, $(\,x_1,\, y_1)$ and $(\,x_2,\, y_2\,)$, and that $F(x)$ is a polynomial of first degree, $P_1(x) = c_1 x + c_2 1$. In matrix form $P_1(x)$ may be written as

$$[\,P_1(x)\,] = [\,x \;\; 1\,]\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \tag{6.5}$$

The interpolation requirements, $F(x_1) = P_1(x_1) = y_1$ and $F(x_2) = P_1(x_2) = y_2$, result in the linear system

$$\begin{bmatrix} P_1(x_1) \\ P_1(x_2) \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{6.6}$$

The 2×2 matrix $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}$, in (6.6), is called a *Vandermonde* matrix. The solution of (6.6) may be expressed in terms of the inverse of the Vandermonde matrix.

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{6.7}$$

Substituting (6.7) into (6.5) gives

$$[\, P_1(x) \,] = [\, x \ 1 \,] \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{6.8}$$

The inverse of the 2×2 matrix $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}^{-1}$ is obtained by interchanging the elements on the main diagonal, negating the off-diagonal terms and dividing by the determinant, $x_1 - x_2$ (recall Problem 1-12); thus, (6.8) becomes

$$[\, P_1(x) \,] = \left([\, x \ 1 \,] \frac{\begin{bmatrix} 1 & -1 \\ -x_2 & x_1 \end{bmatrix}}{x_1 - x_2} \right) \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}. \tag{6.9}$$

Associating the matrices as indicated in (6.9) will give the *Lagrange* form of a linear interpolating polynomial

$$P_1(x) = \left(\frac{x - x_2}{x_1 - x_2} \right) y_1 + \left(\frac{x - x_1}{x_2 - x_1} \right) y_2. \tag{6.10}$$

The two linear polynomials $\frac{x - x_2}{x_1 - x_2}$ and $\frac{x - x_1}{x_2 - x_1}$ on the right side of (6.10) are called Lagrange polynomials. They are denoted $l_1(x)$ and $l_2(x)$, so that (6.10) may be written as

$$P_1(x) = l_1(x) y_1 + l_2(x) y_2.$$

Lagrange polynomials have many interesting properties. Convince yourself that the linear polynomials satisfy the relationship $l_1(x) + l_2(x) = 1$.

Using the Lagrange form to interpolate at $x = \overline{x}$ we find

$$P_1(\overline{x}) = \left(\frac{\overline{x} - x_2}{x_1 - x_2} \right) y_1 + \left(\frac{\overline{x} - x_1}{x_2 - x_1} \right) y_2 = l_1(\overline{x}) y_1 + l_2(\overline{x}) y_2. \tag{6.11}$$

The values of the Lagrange polynomials at $\overline{x}$ represent the weights associated with y_1 and y_2. Observe that if $\overline{x}$ is the midpoint of the interval $[\, x_1, \, x_2 \,]$, each of the weights will be $\frac{1}{2}$ giving equal weighting to both y_1 and y_2, an intuitive result for linear interpolation.

MATLAB's **interp1q** will perform linear interpolation in a table of values by building a linear interpolating polynomial for each interval and then selecting the proper interval for a given $\overline{x}$. **interp1q** requires that data be entered as columns. Consider the following data:

x	y
2.0	1.623
2.5	1.855
4.0	2.041
4.5	2.333
6.0	2.561

The following commands will guarantee that **x** and **y** are columns (for **interp1**) and will interpolate at the three values $\overline{x} = 2.2,\ 3.1$ and 5.2.

```
>> x = x(:);y = y(:);
>> interp1q(x,y,[2.2, 3.1,5.2]')

ans =
  1.7158e+000
  1.9294e+000
  2.4394e+000
```

The output shows that the value of the a linear interpolating polynomial at 3.1 is 1.9294. With 3.1 in the interval $[\,2.5, 4.0\,]$ you should be able to verify that

$$1.9293 = l_1(3.1)\cdot 1.855 + l_2(3.1)\cdot 2.041$$

by constructing the appropriate $l_1(x)$ and $l_2(x)$. Since the original data contains three decimal places, it is standard practice to retain only three places, 1.929, for the interpolated value.

Quadratic Interpolating Polynomials

The procedures applied to two data points may be extended easily to three data points resulting in a quadratic interpolating polynomial. In matrix form, a quadratic polynomial, $c_1x^2 + c_2x + c_3 1$, may be written as

$$[\,P_2(x)\,] = [\,x^2 \quad x \quad 1\,]\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = [\,x^2 \quad x \quad 1\,]\boldsymbol{c}.$$

Interpolation requirements yield the 3×3 linear system

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \text{or} \quad \boldsymbol{Vc} = \boldsymbol{y}.$$

The structure of the Vandermonde matrix, $\boldsymbol{V}$, begins to emerge. The third column consists of ones, the second column contains the values of $\boldsymbol{x}$ and the first column contains the squares of $\boldsymbol{x}$. >> **help vander** provides the specific details.

With $\boldsymbol{c} = \boldsymbol{V}^{-1}\boldsymbol{y}$, the quadratic interpolating polynomial becomes

$$[\,P_2(x)\,] = ([\,x^2 \quad x \quad 1\,]\boldsymbol{V}^{-1})\boldsymbol{y}$$

where the terms have been grouped as in (6.8). The symbolic computation of

$$[\,x^2 \quad x \quad 1\,]\boldsymbol{V}^{-1} = [\,l_1(x) \quad l_2(x) \quad l_3(x)\,]$$

is a daunting task. Fortunately, the symbolic capabilities of MATLAB provide the result. The **factor** command is used to simplify the matrix expression. In addition, for display purposes, MATLAB's non-conjugate transpose, .' , is used on the symbolic expression.

```
>> syms x x1 x2 x3
>> v=[x1^2,x1,1;x2^2,x2,1;x3^2,x3,1];
>> Lagrange = factor([x^2,x,1]*inv(v)).'
Lagrange =
[  (x-x3)*(x-x2)/(x1-x3)/(x1-x2)]
[ -(x-x3)*(x-x1)/(x2-x3)/(x1-x2)]
[  (x-x2)*(x-x1)/(x2-x3)/(x1-x3)]
```

With the MATLAB output the structure of $[\,l_1(x) \quad l_2(x) \quad l_3(x)\,]$ should be clear. Rearranging terms slightly produces the standard forms

$$l_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)},$$

$$l_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)},$$

$$l_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}.$$

Note the symmetry in the quadratic Lagrange polynomials. Multiplying each of the corresponding terms by the appropriate y_i completes the Lagrange form of the quadratic interpolating polynomial

$$P_2(x) \,= l_1(x)y_1 + l_2(x)y_2 + l_3(x)y_{3.} \tag{6.12}$$

Each $l_i(x)$ has two zeros. For example, $l_2(x)$ has zeros at both $x_1$and x_3. In addition, the graph of $l_2(x)$ goes through the point $(x_2,\, 1)$. There are similar properties for the other two terms. MATLAB shows that the sum of the quadratic Lagrange polynomials, the weighting values for interpolation, is one. **Lagrange** is defined in the MATLAB output above.

```
>> factor(Lagrange(1)+Lagrange(2)+Lagrange(3))
ans =
1
```

It is a simple task to generalize Lagrange's polynomial structure. The Lagrange form for n data points, a polynomial of degree $n-1$, is given by

$$P_{n-1}(x) = l_1(x)y_1 + l_2(x)y_2 + l_3(x)y_3 + \ldots + l_n(x)y_n, \tag{6.13}$$

where each $l_i(x)$ is also a polynomial of degree $n-1$ with the form

$$l_i(x) = \frac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}.$$

Note that the term $(x-x_i)$ is missing from the numerator and that x_i appears in each term of the denominator. Lagrange interpolating polynomials are easy to understand; however, the many differences and products makes the Lagrange procedure cumbersome for large n. Moreover, if one more data point is added, a new set of $l_i(x)$'s must be computed to construct $P_n(x)$.

6.3 Newton Interpolating Polynomials

Newton suggested an alternative scheme to systematically construct interpolating polynomials. The polynomials are structured so that adding one additional data point, which increases the degree, requires the addition of only one new term. Beginning with $P_1(x)$ the first few polynomials are

2 data points:	$P_1(x) = a_0 + a_1(x-x_1)$
3 data points:	$P_2(x) = P_1(x) + a_2(x-x_1)(x-x_2)$
4 data points:	$P_3(x) = P_2(x) + a_3(x-x_1)(x-x_2)(x-x_3)$

The structure of Newton's form allows for nested evaluation. For example,

$$P_2(x) = a_0 + a_1(x-x_1) + a_2(x-x_1)(x-x_2)$$

may be rewritten as

$$P_2(x) = [a_2(x-x_2) + a_1](x-x_1) + a_0. \tag{6.14}$$

To evaluate (6.14) we begin with $(x-x_2)$, followed by multiplication by a_2, addition of a_1, multiplication by $(x-x_1)$ and, finally, addition of a_0. The nested method for polynomial evaluation reduces the number of arithmetic operations and typically improves numerical accuracy.

The coefficients in the Newton polynomial form are determined by the interpolation requirements, beginning with $P_1(x_1) = a_0 = y_1$. At x_2, $P_1(x_2) = a_0 + a_1(x_2-x_1) = y_2$. It is easy to see that $a_1 = \frac{y_2-a_0}{x_2-x_1} = \frac{y_2-y_1}{x_2-x_1}$. The coefficient $a_1$is called a *first divided difference* for the data at $x_1$and x_2. If there is a third data point we use $P_2(x_3)$ to compute a_2. Substituting gives $P_2(x_3) = a_0 + a_1(x_3-x_1) + a_2(x_3-x_1)(x_3-x_2) = y_3$. It may be shown that

$$a_2 = \frac{\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}}{x_3-x_1}. \tag{6.15}$$

The coefficient a_2 given in (6.15) is called a second divided difference for the data. The quotients in the numerator are both first order divided differences using different data points. Elaborate notations are used to identify various divided differences. For example the coefficient $a_0 = y_1$, a zeroth divided difference, may be denoted $DD_0[x_1]$. Continuing this scheme we find

$$a_1 = DD_1[x_1, x_2] = \frac{DD_0[x_2] - DD_0[x_1]}{x_2 - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

and $$a_2 = DD_2[x_1, x_2, x_3] = \frac{DD_1[x_2, x_3] - DD_1[x_1, x_2]}{x_3 - x_1} = \frac{\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}}{x_3 - x_1}.$$

At this point the structure of the coefficients emerges with a_3 given by a third divided difference

$$a_3 = DD_3[x_1, x_2, x_3, x_4] = \frac{DD_2[x_2, x_3, x_4] - DD_2[x_1, x_2, x_3]}{x_4 - x_1}$$

Typically, the various divided differences are displayed in a table, for example,

x_i	y_i	First DD	Second DD	Third DD
x_1	$y_1 = DD_0[x_1]$			
		$DD_1[x_1, x_2]$		
x_2	$y_2 = DD_0[x_2]$		$DD_2[x_1, x_2, x_3]$	
		$DD_1[x_2, x_3]$		$DD_3[x_1, x_2, x_3, x_4]$
x_3	$y_3 = DD_0[x_3]$		$DD_2[x_2, x_3, x_4]$	
		$DD_1[x_3, x_4]$		
x_4	$y_4 = DD_0[x_4]$			

The coefficients for Newton's interpolating polynomial are found at the top of each column of differences. Most advanced texts on numerical analysis provide additional information about divided differences and their properties. A divided difference table for the data in Section 6.2 is given by

x_i	y_i	First DD	Second DD	Third DD	Fourth DD
2.0	1.623				
		0.464			
2.5	1.855		-0.17		
		0.124		0.16	
4.0	2.041		0.23		-0.071856
		0.584		-0.127433	
4.5	2.333		-0.216		
		0.152			
6.0	2.561				

Test your knowledge of the divided difference formulas by verifying the values in the table. Pay particular attention to the values used in the denominators of each difference. We may now identify the coefficients for the fourth degree Newton interpolating polynomial. They are as follows: $a_0 = 1.623,\ a_1 = 0.464, a_2 = -0.17,\ a_3 = 0.16$ and $a_4 = -0.071856.$

The M-file divdif.m will compute the divided difference coefficients for use in Newton interpolating polynomials. MATLAB's **length** will determine the number of data points.

M-file divdif.m

```
function dd=divdif(x,y)
%DIVDIF Newton divided differences
x = x(:);y = y(:);
n = length(x);
d = y;
for i = 2:n
        for j = n:-1:i
                d(j)=(d(j)-d(j-1))/(x(j)-x(j-i+1));
        end
end
dd = d;
```

Using the data from Section 6.2, the coefficients for the Newton form of $P_4(x)$ may be computed as follows:

```
>> a=divdif(x,y)
a =
  1.6230e+000
  4.6400e-001
 -1.7000e-001
  1.6000e-001
 -7.1857e-002
```

The Newton interpolating polynomial is given by

$$\begin{aligned} P_4(x) = 1.623 + 0.464(x-2) - 0.17(x-2)(x-2.5) \\ + 0.16(x-2)(x-2.5)(x-4) \\ -0.071857(x-2)(x-2.5)(x-4)(x-4.5). \end{aligned} \tag{6.16}$$

Although $x_5 = 6$ does not appear explicitly in $P_4(x)$, the fifth data point, (6, 2.561), has been used in the divided difference computations. You may wish to verify that $P_4(6) = 2.561.$ Begin by rewriting (6.16) in nested form

$$P_4(x) = \left\{ \left[\Big(a_4(x-x_4) + a_3 \Big)(x-x_3) + a_2 \right](x-x_2) + a_1 \right\}(x-x_1) + a_0$$

before you start the actual computations.

6.4 Polynomial Interpolation - MATLAB

If we wish to express the interpolating polynomial in the power form,

$$P_{n-1}(x) = c_1 x^{n-1} + c_2 x^{n-2} + c_3 x^{n-3} + \ldots + c_{n-1} x + c_n, \tag{6.17}$$

neither the Lagrange nor the Newton structure permit easy identification of the coefficients. To resolve this problem the general approach of Section 6.1 may be used. With $F(x) = P_{n-1}(x)$ and $g_1(x) = x^{n-1}, g_2(x) = x^{n-2}, \ldots, g_j(x) = x^{n-j}, \ldots$ and $g_n(x) = 1$ we may find the desired coefficients using the interpolation requirements on (6.17).

$$P_{n-1}(x_i) = c_1 x_i^{n-1} + c_2 x_i^{n-2} + c_3 x_i^{n-3} + \ldots + c_{n-1} x_i + c_n = y_i,$$

for $i = 1$ to n. In matrix form, we have an $n \times n$ Vandermonde system $\boldsymbol{Vc} = \boldsymbol{y}$.

Constructing the matrix $\boldsymbol{V}$ from the data is not a difficult task; however, MATLAB provides a command, **polyfit**, which will construct $\boldsymbol{V}$ and also solve the linear system, $\boldsymbol{Vc} = \boldsymbol{y}$, returning the coefficients in descending order. Once the coefficient vector $\boldsymbol{c}$ has been computed, the interpolating polynomial (6.17) may be evaluated with **polyval**.

As an example, let's use the data from Section 6.2. Recall that the degree of the polynomial is one less than the number of data points. With five data points given, a fourth degree polynomial is required.. The inputs to **polyfit** are the x and y data along with the degree. It is important to enter the degree correctly. If a degree error is made, **polyfit** will return coefficient values for a *least squares* polynomial. (See Chapter 7.) The degree of the polynomial may be entered explicitly or computed by MATLAB using **length**.

```
>> c4 = polyfit(x,y,length(x)-1)
c4 =
 -7.1857e-002  1.0941e+000 -5.9312e+000  1.3783e+001 -9.8221e+000
```

In power form, the interpolating polynomial is

$$P_4(x) = -0.071857x^4 + 1.0941x^3 - 5.9312x^2 + 13.783x - 9.8221. \tag{6.18}$$

Observe that the coefficient of x^4 in (6.18), -0.071857, is also found in the last term of (6.16).

To confirm that $P_4(x)$ does indeed interpolate, the command **polyval(c4,x)** will return the data values for y.

```
>> polyval(c4,x)'
ans =
  1.6230e+000
  1.8550e+000
  2.0410e+000
  2.3330e+000
  2.5610e+000
```

Between data points the behavior of interpolating polynomials may be less than satisfactory. A plot of $P_4(x)$ will expose serious problems with polynomial interpolation. To produce a smooth graph we evaluate $P_4(x)$ at many points on the interval $[\,2,\ 6\,]$ and plot the results with the following commands.

```
>> xi=2:.01:6;yi=polyval(c4,xi);
>> plot(x,y,'ko',xi,yi,'k')
```

As seen in the left side of Figure 6.1 the plot of $P_4(x)$ on the interval $[\,2,\ 4.5\,]$ seems reasonable; however, on the interval $[\,4.5, 6\,]$ the plot has a local maximum that appears to be inconsistent with the data. The plot of $P_4(x)$ is typical of interpolating polynomials; however, the data points play a major part in the results. As a second example consider a revised data set in which the last three y values have been increased from the data set in Section 6.2.

x	yr
2.0	1.623
2.5	1.855
4.0	2.425
4.5	2.806
6.0	3.966

Using the revised data values, a different $P_4(x)$ is computed. The graphs of both polynomials are plotted in the same figure window using **subplot**. As given below **subplot** produces one row and two columns of smaller figure windows. See >> **help subplot** for additional information about the parameter list. Note also the use of MATLAB's **title** command.

```
>> subplot(1,2,1),plot(x,y,'ko',xi,yi,'k'),title('P4(x), Original Data')
>> subplot(1,2,2),plot(x,yr,'ko',xi,yri,'k'),title('P4(x), Revised Data')
```

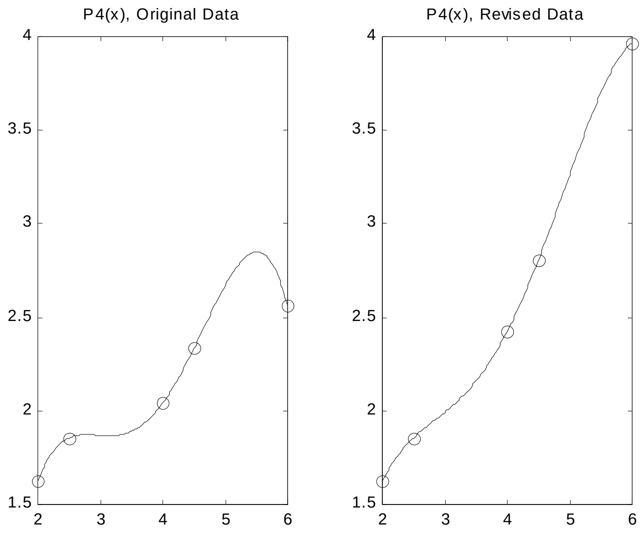

Figure 6.1 Interpolating Polynomials

It should be clear that polynomial interpolation is highly dependent on the data set. Sometimes the results are very reasonable. In other cases poor results are found.

6.5 Error in Polynomial Interpolation

Attempts to quantify the accuracy of interpolating polynomials based on experimental data are usually limited to a graphical evaluation as in Figure 6.1. There are no methods to compute or estimate the accuracy of results. It is perhaps best to use more than one method to estimate a value between data points.

On the other hand, there are situations in which we wish to approximate a known function, $f(x)$, by an interpolating polynomial. In Chapter 3 we approximated a function by a Taylor polynomial centered at a specified point, asking for interpolation of the function and various derivatives at that point. Constructing a Taylor polynomial or an interpolating polynomial have similar goals $-$ replace a complicated function with a polynomial.

In this chapter, the x_i's are selected from the interval $[a, b]$ with the corresponding y_i's computed from $f(x_i) = y_i$. Similar to Taylor polynomials, there is an error formula for interpolation. The formula assumes that $f(x)$ has n continuous derivatives on the interval $[a, b]$ and that the x_i's are distinct values also in $[\,a,\ b\,]$. The result for n points is stated without proof.

$$\text{Error}(x) = f(x) - P_{n-1}(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})(x-x_n)}{n!}\, f^{(n)}(c), \quad (6.19)$$

where x is in $[a, b]$ and c is an unknown number in the open interval (a, b). The remainder term for a Taylor expansion looks similar. See (3.8) with n replaced by $n-1$.

The implications of (6.19) are difficult to comprehend without a specific example. As noted at the beginning of this chapter linear interpolation in tables of known functions, using two adjacent data points, was once a common procedure. With $n = 2$, (6.19) becomes

$$f(x) - P_1(x) = \frac{(x-x_1)(x-x_2)}{2}\, f''(c). \quad (6.20)$$

The geometric implication of (6.20) rests on the fact that concavity is measured by the value of the second derivative. If the concavity of $f(x)$ on the interval $[a, b]$ is small, i.e. the graph of $f(x)$ resembles a line, we should expect the error in linear interpolation to be small as well. Since c is unknown, an analysis of (6.20) makes use of a bound for $f''(x)$ as follows:

$$|\, f(x) - P_1(x)\,| \le \frac{1}{2}\,|\,(x-x_1)(x-x_2)\,| \cdot \max_{[\,x_1,\, x_2]} |\, f''(\text{x})\,|. \quad (6.21)$$

The other terms in (6.21) are related to the spacing of the data points. In other words, with x between x_1 and x_2 a smaller interval, $[\,x_1, x_2]$, should produce better results. For x in the interval $[\,x_1, x_2]$, we may show that $|\,(x-x_1)(x-x_2)\,| \le \frac{1}{4}(x_2-x_1)^2$ so that the error in linear interpolation is bounded by

$$|\, f(x) - P_1(x)\,| \le \frac{1}{8}\,(x_2-x_1)^2 \cdot \max_{[\,x_1,\, x_2]} |\, f''(\text{x})\,|. \quad (6.22)$$

The error bound (6.22) may be used to establish a relationship between the spacing of the x_i's and the number of decimal places for the y_i's in a table which will be used for linear interpolation. For example, standard tables of the exponential function, e^x, on the interval $[0,1]$, typically have a spacing of 0.01 and provide four decimal place values. Since the maximum value of the second derivative of e^x on the interval $[0,\ 1]$ is e, the error bound becomes $|e^x - P_1(x)| \le \frac{1}{8}(0.01)^2 e \approx 0.000034$. In other words, we may expect linear interpolation in this table to provide at least four decimal places of accuracy.

Polynomial Models for Known Functions

Constructing interpolating polynomials to approximate known functions suffers the same fate as noted with experimental data. An example will illustrate the situation. Let us approximate $f(x) = \sin(x)$ on the interval $[0,\pi]$ with $P_4(x)$ by selecting five equally spaced points using **linspace**. The following MATLAB commands construct and evaluate $P_4(x)$. Figure 6.2 shows the data points, the interpolating polynomial $P_4(x)$, and the error term $(\sin(x) - P_4(x))$, multiplied by 100.

```
>> x = linspace(0,pi,5);
>> y = sin(x);
>> c4 = polyfit(x,y,4);
>> xi = 0:.01:3.14;
>> p4i = polyval(c4,xi);
>> e4i = sin(xi)-p4i;
>> subplot(121),plot(x,y,'ko',xi,p4i,'k',xi,100*e4i,'k'),title('UNIFORM'), grid on
```

Figure 6.2 reveals that the error is largest near the end points of the interval. The magnitude of the error peaks may be reduced using techniques which select different points in the interval. The second subplot shows the effect of non-uniformly spaced data points specified in the vector $xc = [.08, .65, 1.57, 2.49, 3.06]$.

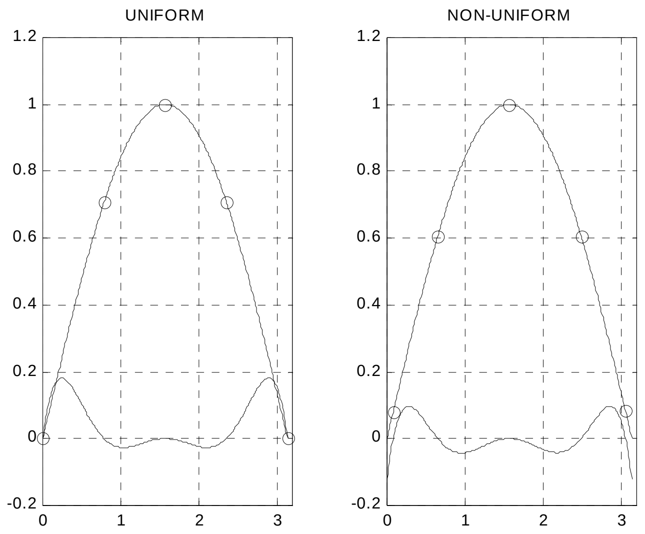

Figure 6.2 $\sin(x)$ and Interpolation Error

In either case, the errors are largest near the end points of the interval. The source of the problem may be found in the error expression(6.19), with $n = 5$:

$$\sin(x) - P_4(x) = (x - x_1)(x - x_2)(x - x_3)(x - x_4)(x - x_5)\cos(c)/5!. \tag{6.23}$$

The fifth degree polynomial in (6.23) controls the shape of the error curve. Note that both error curves in the figure have five zeros. With proper choice of interpolation points, the shape of the error curve may be adjusted to minimize the error peaks. The technique for selecting the values used in xc may be found in more advanced texts on numerical analysis.

6.6 Piecewise Interpolating Polynomials

Examples in the previous sections have illustrated problems associated with interpolating polynomials of modest degree. Another possible approach treats the data by grouping data points so that lower order polynomials may be used.

The data set in Section 6.2 contained five points leading to an interpolating polynomial of fourth degree. Piecewise linear interpolation will construct four linear polynomials, one for each subinterval. This is exactly what **interp1q** produces. If **plot** is used to plot a data set, MATLAB automatically draws lines between the points.

Continuing the piecewise approach, quadratic interpolating polynomials may be constructed using groupings of three data points. If we use the first three data points a quadratic polynomial may be constructed on the interval $[\,x_1,\, x_3\,]$. Likewise, using points three, four and five, a second quadratic may be constructed on the interval $[\,x_3,\, x_5\,]$.

The following MATLAB commands may be used to plot both the piecewise linear and quadratic polynomials using the data set of Section 6.2. Study the sequence of commands carefully.

```
>> x = [2, 2.5, 4, 4.5, 6];
>> y = [1.623, 1.855, 2.041, 2.333, 2.561];
>> subplot(121),plot(x,y,'k',x,y,'ko'),title('LINEAR')
>> x1 = [2 2.5 4]; x2 = [4 4.5 6];
>> y1 = [1.623 1.855 2.041];y2 = [2.041 2.333 2.561];
>> c21 = polyfit(x1,y1,2); c22 = polyfit(x2,y2,2);
>> x1i = 2:.01:4; x2i = 4:.01:6;
>> y1i = polyval(c21,x1i); y2i = polyval(c22,x2i);
>> subplot(122),plot(x1,y1,'ko',x1i,y1i,'k',x2,y2,'ko',x2i,y2i,'k'), ...
title('QUADRATIC')
```

The plot in Figure 6.3 shows both piecewise quadratics to be concave down; thus we expect the quadratic scheme to give higher interpolated values compared to the linear approach. Some interpolated values for the data are given in the following table.

x	Linear	Quadratic	$P_4(x)$
3.5	1.979	2.058	1.890
4.3	2.216	2.229	2.203
5.5	2.485	2.593	2.850

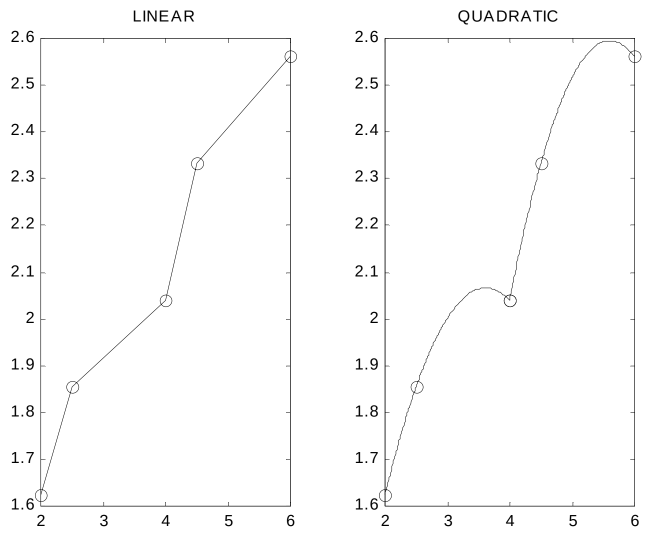

Figure 6.3 Piecewise Interpolation

One of the most obvious differences between the piecewise polynomials and $P_4(x)$ occurs where the piecewise segments are joined together. At these point, the interpolating function makes an abrupt change. In mathematical terms, the derivative does not exist at these points. In the next section we will consider a procedure to construct an interpolating function that will eliminate the abrupt changes.

6.7 Interpolating Splines

Given a set of data points $(\,x_i,\ y_i)$ for $i = 1$ to n, an interpolating *spline* is a piecewise mathematical function designed to model the data. It is customary to call the x_i's *nodes.*

Definition: A spline, $S(x)$, is a piecewise interpolating polynomial of low degree, $m = 1, 2,$ or 3, between each pair of consecutive nodes, joined together at the interior nodes, $x_2, x_3, \ldots, x_{i-1}$, so that $m - 1$ derivatives of $S(x)$ are continuous on the interval $(\,x_1, x_n)$.

$S(x)$ is composed of $n - 1$ polynomials, one for each subinterval.

$$S(x) = \begin{cases} S_1(x), & x_1 \le x \le x_2 \\ S_2(x), & x_2 \le x \le x_3 \\ \quad\vdots & \\ S_j(x), & x_j \le x \le x_{j+1} \\ \quad\vdots & \\ S_{n-2}(x), & x_{n-2} \le x \le x_{n-1} \\ S_{n-1}(x), & x_{n-1} \le x \le x_n \end{cases} \tag{6.24}$$

The special case of $m = 1$ is easy to understand. In a first degree spline each $S_j(x)$ will be a line segment resulting in the piecewise linear model discussed earlier. While interesting, quadratic splines, $m = 2$, will not be discussed. See Problem 6-19 in the chapter problems.

Details may be found in more advanced texts. Do not confuse a quadratic spline with the piecewise quadratic scheme discussed in the previous section. The major difference is in how the pieces are joined together.

The main focus of this section is on third degree or *cubic* splines, $m = 3$. In our development each $S_j(x)$ will be expressed in descending powers of $(x - x_j)$.

$$S_j(x) = a_j(x - x_j)^3 + b_j(x - x_j)^2 + c_j(x - x_j) + d_j,\ x_j \le x \ \le\ x_{j+1} \tag{6.25}$$

To comprehend the task at hand, note that there are four unknown coefficients in each $S_j(x)$, multiplied by the number of subintervals, $n - 1$, gives total of $4n - 4$ unknowns to be computed.

The unknowns will be determined from the definition of a cubic spline that requires interpolation of the data and continuity of the first and second derivatives of $S(x)$ on the interval $(\,x_1, x_n)$. It should be clear that interpolation and the piecewise structure of $S(x)$ guarantees the continuity of $S(x)$ itself. Although not obvious, the requirements in the definition will result in $4n - 6$ equations. Two additional equations, usually called endpoint conditions, will be necessary to compute the unknowns. Depending on the nature of the endpoint conditions various types of cubic splines may be constructed.

Using (6.25) and the interpolation requirement $S_j(x_j) = y_j$ shows that $d_j = y_j$ for all j. Unfortunately, the other coefficients are not so easy to determine. The remaining coefficients, the a_j's, b_j's and c_j's, may be found from three equations derived from the continuity requirements at the interior nodes, $x_2, x_3, \ldots, x_{n-1}$ as follow:

$$S_{j-1}(x_j) = S_j(x_j),\ \text{for}\ \ j = 2, 3, \ldots, n-1$$

$$S'_{j-1}(x_j) = S'_j(x_j),\ \text{for}\ \ j = 2, 3, \ldots, n-1$$

$$S''_{j-1}(x_j) = S''_j(x_j),\ \text{for}\ \ j = 2, 3, \ldots, n-1$$

Using (6.25), these three equations become

$$a_{j-1}(x_j - x_{j-1})^3 + b_{j-1}(x_j - x_{j-1})^2 + c_{j-1}(x_j - x_{j-1}) + d_{j-1} = d_j,$$

$$3a_{j-1}(x_j - x_{j-1})^2 + 2b_{j-1}(x_j - x_{j-1}) + c_{j-1} = c_j,$$

$$6a_{j-1}(x_j - x_{j-1}) + 2b_{j-1} = 2b_j.$$

To simplify the development, assume that the nodes are equally spaced, so that each $(x_j - x_{j-1}) = h$. The three equations may now be written as

$$a_{j-1}h^3 + b_{j-1}h^2 + c_{j-1}h + d_{j-1} = d_j \tag{6.26}$$

$$3a_{j-1}h^2 + 2b_{j-1}h + c_{j-1} = c_j \tag{6.27}$$

$$6a_{j-1}h + 2b_{j-1} = 2b_j \tag{6.28}$$

Ignoring subscripts for the moment, (6.26), (6.27) and (6.28) may be viewed as three equations in three unknowns. Keep in mind that the d_j's and h are known. Elimination will allow us to construct one equation from the three. We begin by solving (6.28) for $a_{j-1} = \frac{1}{3}(b_j - b_{j-1})/h$. Substituting into (6.26) and (6.27) gives, after simplification,

$$\frac{1}{3}(2b_{j-1} + b_j)h^2 + c_{j-1}h + d_{j-1} = d_j \tag{6.29}$$

$$(b_{j-1} + b_j)h + c_{j-1} = c_j \tag{6.30}$$

Solving (6.29) for c_{j-1} yields

$$c_{j-1} = -\frac{1}{3}(2b_{j-1} + b_j)h + (- d_{j-1} + d_j)/h. \tag{6.31}$$

For use in (6.30) the subscript in (6.31) may be incremented by one to give

$$c_j = -\frac{1}{3}(2b_j + b_{j+1})h + (-d_j + d_{j+1})/h. \tag{6.32}$$

Finally, substituting c_{j-1}, (6.31), and c_j, (6.32), into (6.30) gives, after collecting terms,

$$b_{j-1} + 4b_j + b_{j+1} = 3(d_{j-1} - 2d_j + d_{j+1})/h^2, j = 2, \ldots, n-2. \tag{6.33}$$

Equation (6.33) is the key result for finding the coefficients of the spline. The equations are

$$\begin{array}{ll} \text{For } j = 2: & b_1 + 4b_2 + b_3 = 3(d_1 - 2d_2 + d_3)/h^2 \\ \text{For } j = 3: & b_2 + 4b_3 + b_4 = 3(d_2 - 2d_3 + d_4)/h^2 \\ \quad\vdots & \quad\vdots \\ \text{For } j = n-3: & b_{n-4} + 4b_{n-3} + b_{n-2} = 3(d_{n-4} - 2d_{n-3} + d_{n-2})/h^2 \\ \text{For } j = n-2: & b_{n-3} + 4b_{n-2} + b_{n-1} = 3(d_{n-3} - 2d_{n-2} + d_{n-1})/h^2 \end{array} \tag{6.34}$$

The linear system in (6.34) contains $n-3$ equations with $n-1$ unknowns, the b_j's. Recall that the d_j's equal the known y_j's. So that $y_n = d_n$ is represented in the system, we create one additional equation with $j = n-1$,

$$b_{n-2} + 4b_{n-1} + b_n = 3(d_{n-2} - 2d_{n-1} + d_n)/h^2 \tag{6.35}$$

Adding (6.35) does not alter the fact that there are more unknowns than equations. In matrix form we have $n-2$ equations in n unknowns.

$$\begin{bmatrix} 1 & 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{bmatrix} = \frac{3}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-3} - 2y_{n-2} + y_{n-1} \\ y_{n-2} - 2y_{n-1} + y_n \end{bmatrix} \tag{6.36}$$

Two additional equations are provided by the endpoint conditions. This will allow the b_j's to be computed. With the b_j's known, the a_j's and c_j's may be found from

$$a_j = \frac{1}{3}(b_{j+1} - b_j)/h \tag{6.37}$$

and

$$c_j = -\frac{1}{3}(2b_j + b_{j+1})h + (-d_j + d_{j+1})/h. \tag{6.38}$$

<u>Endpoint Conditions: Natural Cubic Spline</u>

The endpoint conditions for a *natural* cubic spline are defined to make the graph of $S(x)$ resemble a line near the two exterior nodes, $x_1$and x_n, by asking that the concavity be zero at both $x_1$and x_n. In other words, $S''(x_1) = 0$ and $S''(x_n) = 0$. The second derivative of (6.25) is

$$S_j''(x) = 6a_j(x - x_j) + 2b_j. \tag{6.39}$$

Using (6.39), $S''(x_1) = S_1''(x_1) = 2b_1 = 0$. A similar result holds with $j = n$ so that b_1 and b_n are both zero. With $b_1 = b_n = 0$, the system (6.36) becomes a tridiagonal linear system with $n-2$ equations and $n-2$ unknowns $b_2, \ldots, b_{n-1}$.

$$\begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} = \frac{3}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-3} - 2y_{n-2} + y_{n-1} \\ y_{n-2} - 2y_{n-1} + y_n \end{bmatrix} \tag{6.40}$$

Remember that (6.40) is based on equally spaced nodes. In the case of unequally spaced nodes the key equation, (6.33), will reflect the values of $h_{j-1} = x_j - x_{j-1}$and $h_j = x_{j+1} - x_j$. Details may be found in more advanced texts.

Endpoint Conditions: Not-a-Knot Cubic Spline

The extra conditions for a *not-a-knot* spline are not actually endpoint conditions. Instead they impose additional continuity requirements on $S(x)$ at the interior nodes x_2 and x_{n-1}. In particular, the third derivative, $S'''(x)$, must be continuous at both x_2 and x_{n-1}. Since we are working with cubic splines, the third derivative, $S_j'''(x) = 6a_j$, will be a constant. Continuity of $S'''(x_2)$ means $S_1'''(x_2) = S_2'''(x_2)$ so that $6a_1 = 6a_2$ or $a_1 = a_2$. Similar equations at x_{n-1} result in $a_{n-2} = a_{n-1}$.

Equation (6.37) may be used to determine the effect of these equalities on the unknown b_j's. For example, $a_1 = a_2$ implies $\frac{1}{3}(b_2 - b_1)/h = \frac{1}{3}(b_3 - b_2)/h$ or $b_1 = 2b_2 - b_3$. Substituting for b_1 in the first equation of (6.34) gives

$$6b_2 = 3(d_1 - 2d_2 + d_3)/h^2.$$

Likewise, $a_{n-2} = a_{n-1}$ will give $b_n = -b_{n-2} + 2b_{n-1}$. Substituting into (6.35) gives

$$6b_{n-1} = 3(d_{n-2} - 2d_{n-1} + d_n)/h^2.$$

With these new equations, the unknowns for the not-a-knot cubic spline are found by solving the tridiagonal system

$$\begin{bmatrix} 6 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} = \frac{3}{h^2} \begin{bmatrix} y_1 - 2y_2 + y_3 \\ y_2 - 2y_3 + y_4 \\ y_3 - 2y_4 + y_5 \\ \vdots \\ y_{n-3} - 2y_{n-2} + y_{n-1} \\ y_{n-2} - 2y_{n-1} + y_n \end{bmatrix}. \tag{6.41}$$

MATLAB contains a command, **spline,** that will assemble and solve the tridiagonal system (6.41) and compute the remaining coefficients, (6.37) and (6.38). Examples will be discussed in Section 6.8.

Endpoint Conditions: Clamped Cubic Spline

A *clamped* spline specifies values for the slope at both endpoints, $S'(x_1) = M_1$ and $S'(x_n) = M_n$. The first derivative of (6.25) is

$$S_j'(x) = 3a_j(x - x_j)^2 + 2b_j(x - x_j) + c_j. \tag{6.42}$$

With (6.41), $S'(x_1) = S_1'(x_1) = c_1 = M_1$. Using $j = 1$ in (6.38) we find

$$c_1 = -\frac{1}{3}(2b_1 + b_2)h + (-d_1 + d_2)/h = M_1. \tag{6.43}$$

Equation (6.43) may be solved for b_1 with the result used in the first equation of (6.34). There is a comparable development at x_n. The computations are omitted. Once again, a tridiagonal linear system must be solved. MATLAB provides an option to construct a clamped spline with the endslopes specified. Simply use the vector $[\,M_1\, y_1\, y_2 \ldots y_n\, M_n\,]$ in place of the usual data values.

The clamped first derivative values, $M_1$and M_n, may be chosen with some goal in mind or estimated from the known data. In a later chapter, methods to approximate derivatives based on data will be studied. These methods will provide the equations needed to produce values for $M_1$and M_n.

Further information and other ways to construct splines may be found in more advanced texts on numerical analysis.

6.8 MATLAB and Splines

As noted in the previous section, MATLAB's **spline** is the basic command for computations of not-a-knot splines. If we wish to model a set of data, $(\,x_i,\, y_i\,)$ for $i = 1$ to n, with $S(x)$ and in turn compute $S(xi)$ the MATLAB command >> **sofxi = spline(x,y,xi)** will return the desired value. To generate a plot of $S(x)$ the single value x may replaced by a vector. As an example, consider the MATLAB commands

```
>> x=[1 2.3 3.1 4 5.2 5.9 6.7]';
>> y=[1.7 2.8 3.6 4.5 3.4 3.1 3.1]';
>> xi=1:.1:6.7;
>> si=spline(x,y,xi);
>> c6=polyfit(x,y,6);
>> yi=polyval(c6,xi);
>> plot(x,y,'ko',xi,yi,'k.',xi,si,'k'),title('S(x) and P6(x)')
```

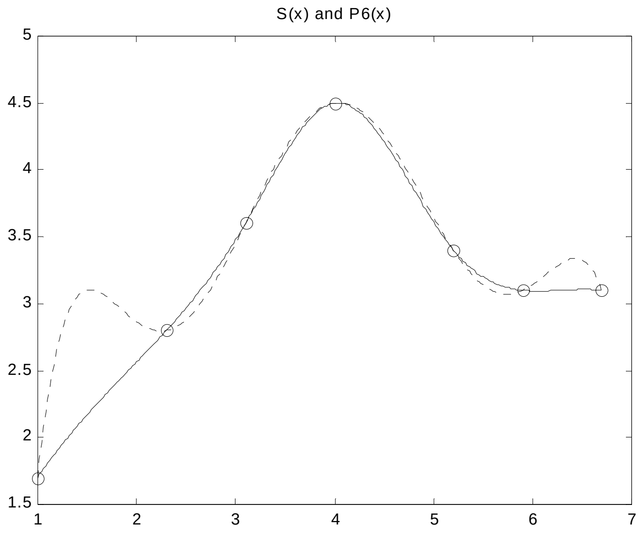

Figure 6.4 Spline and Polynomial Approximation

Figure 6.4 shows a plot of both the spline and the sixth degree interpolating polynomial, $P_6(x)$. Between $x = 2.3$ and $x = 6$ both interpolating approximations appear satisfactory; however, near the end points, the spline provides a better approximation for the data. To represent the data with a clamped spline, estimates for the endslopes are needed. Using a line between the first two data points, $(1,\ 1.7)$ and $(2.3,\ 2.8)$, gives an approximate value of $0.85 = M_1$ for the slope at x_1. Since both y_6 and y_7 are 3.1, a slope value of $0.0 = M_7$ may be used at x_7. The data vector y is now replaced with $yc = [0.85,\ 1.7,\ 2.8,\ 3.6,\ 4.5,\ 3.4,\ 3.1,\ 3.1, 0.0]$. In this example, the differences between the not-a-knot and clamped splines will be slight. You may wish to plot the clamped spline and compare the results with the not-a-knot spline shown in Figure 6.4.

Information about the piecewise cubic polynomials that make up $S(x)$ may be obtained from **spline**. If the third argument in **spline** is omitted, MATLAB provides the spline data in a piecewise form. To actually obtain the coefficients, the $a_j\ b_j\ c_j\ d_j$'s, it will be necessary to disassemble the spline data. Using the example data of this section we find the following MATLAB data.

```
>> spdata = spline(x,y)
spdata =
      form: 'pp'
    breaks: [1 2.3000e+000 3.1000e+000 4 5.2000e+000 5.9000e+000 6.7000e+000]
     coefs: [6x4 double]
    pieces: 6
     order: 4
       dim: 1
```

To view the coefficient data in the 6 by 4 array, **coefs**, it is necessary to take apart the spline information in **spdata** using the MATLAB command **unmkpp** as follows:

```
>> [breaks,spcoef,pieces,order,dim]=unmkpp(spdata);
```

The spline coefficients are contained in **spcoef** in the order $a_j,\ b_j,\ c_j, d_j$. Recall that $d_j = y_j$, the known data values.

```
>> spcoef
spcoef =

  1.1973e-001 -3.3383e-001  1.0778e+000  1.7000e+000
  1.1973e-001  1.3313e-001  8.1687e-001  2.8000e+000
 -7.8789e-001  4.2048e-001  1.2598e+000  3.6000e+000
  7.1490e-001 -1.7068e+000  1.0205e-001  4.5000e+000
 -2.6415e-001  8.6684e-001 -9.0593e-001  3.4000e+000
 -2.6415e-001  3.1213e-001 -8.0650e-002  3.1000e+000
```

As expected for a not-a-knot spline, the first column shows $a_1 = a_2$ and $a_5 = a_6$. The fourth column gives the y_i values. A typical $S_j(x)$ may be constructed using (6.25). For example, with $j = 3$,

$$S_3(x) = -0.79(x-3.1)^3 + 0.42(x-3.1)^2 + 1.26(x-3.10) + 3.6, \quad 3.1 \le x \le 4.$$

With **spdata** known, the spline $S(x)$ may be evaluated at the vector xi using **ppval**. As the name suggests, **ppval** will evaluate piecewise polynomials. As a practical matter the MATLAB commands

```
>> spdata=spline(x,y);
>> si=ppval(spdata,xi);
```

or

```
>> si=spline(x,y,xi);
```

will produce identical results.

Interpolating polynomials and splines may also be used to estimate rates of change from the given data. MATLAB's **polyder** will easily treat the polynomial case. Recall the fourth degree interpolating polynomial (6.18)

$$P_4(x) = -0.071857x^4 + 1.0941x^3 - 5.9312x^2 + 13.783x - 9.8221.$$

The coefficients of $P_4(x)$ are contained in **c4**. For example, $P_4'(3)$ may be computed using the MATLAB commands

```
c4 =
 -7.1857e-002  1.0941e+000 -5.9312e+000  1.3783e+001 -9.8221e+000
>> c4prime=polyder(c4)
c4prime =
 -2.8743e-001  3.2824e+000 -1.1862e+001  1.3783e+001
>> p4prime=polyval(c4prime,3)
p4prime =
 -2.2857e-002
```

Note that **c4prime** contains the coefficients of the derivative.

Extra effort is needed to determine the derivative of the spline $S(x)$. To begin, the first derivative of (6.25) is

$$S_j'(x) = 3a_j(x-x_j)^2 + 2b_j(x-x_j) + c_j, \; x_j \le x \;\le\; x_{j+1}. \tag{6.43}$$

As noted above the spline coefficients are in **spcoef** and it is a simple task to use the data in the array to compute the coefficients in the piecewise derivative. Multiply the first column of **spcoef** by 3, the second column by 2, the third column by 1(no change) and delete the fourth column.

The following M-file computes the coefficients for the derivative of the spline. MATLAB's **mkpp** will build (make) the piecewise polynomial form **dspdata** for use by **ppval**.

<u>M-file dspline.m</u>

```
function dspdata=dspline(spdata)
%DSPLINE Computes coefficients for the derivative of S(x)
% spdata  = spline(x,y), the spline data
% take apart spdata:
[breaks, spcoef, m, order, dim]=unmkpp(spdata);
% compute coefficients of derivative
  dspcoef(:,1) = 3*spcoef(:,1);        % 3*ai's
  dspcoef(:,2) = 2*spcoef(:,2);        % 2*bi's
  dspcoef(:,3) = 1*spcoef(:,3);        % 1*ci's
% construct piecewise data for the derivative
  dspdata = mkpp(breaks,dspcoef);
```

The derivative of the spline representing our example data may be evaluated and plotted as follows: (Recall **xi=1:.1:6.7**.)

```
>> spdata=spline(x,y);
>> dspdata=dspline(spdata);
>> dsi=ppval(dspdata,xi);
>> plot(x,y,'ko',xi,si,'k',xi,dsi,'k'),title('S(x) and Derivative'),grid on
```

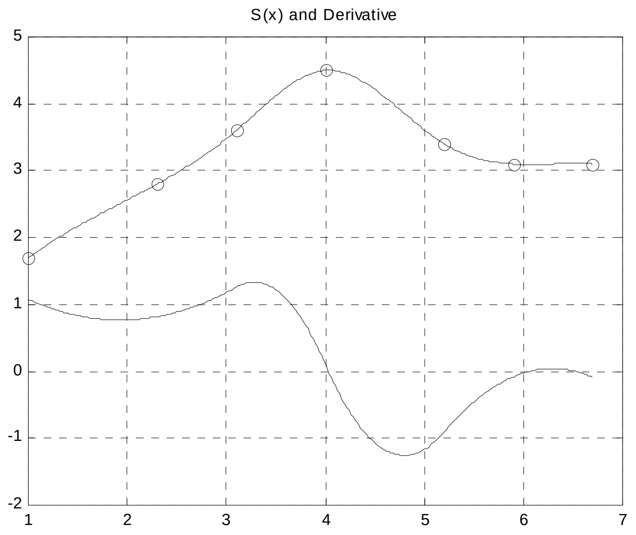

Figure 6.5 $S(x)$ and $S'(x)$

6.9 Inverse Interpolation

As mentioned in Section 4.4, MATLAB's root finding command **fzero** uses inverse quadratic interpolation. At first thought, polynomial interpolation and root finding seem unrelated. Actually the ideas are similar. For example, a linear polynomial may be used for interpolation and, in root finding, lines may be used in various methods to estimate the root.

Consider a function $f(x)$ on the interval $[\,x_1,\,x_2\,]$ with $f(x_1)f(x_2)\ <0$. We know that there must be a root, α, in the interval. To reflect the known function values, (6.10) may be rewritten as

$$P_1(x)=\Big(\frac{x-x_2}{x_1-x_2}\Big)f(x_1)+\Big(\frac{x-x_1}{x_2-x_1}\Big)f(x_2). \tag{6.44}$$

Equation (6.44) may be interpreted in two ways. First, with $\overline{x}$ in the interval $[\,x_1,\,x_2\,]$, $P_1(\overline{x})$ will provide an interpolated estimate of $f(\overline{x})$. Secondly, if we ask that $P_1(x_3)=0$, x_3 is the value where the line intersects the x-axis, in other words, an estimate of the root α.

Solving $P_1(x_3)=0$, may be viewed from the concept of inverse functions. Assuming that f has an inverse, f^{-1}, on the interval $[\,x_1,\,x_2\,]$, we may construct a linear interpolating polynomial for f^{-1} on the interval $[\,y_1,\,y_2\,]=[\,f(x_1),\,f(x_2)\,]$, as follows

$$Q_1(y)=\Big(\frac{y-y_2}{y_1-y_2}\Big)f^{-1}(y_1)+\Big(\frac{y-y_1}{y_2-y_1}\Big)f^{-1}(y_2). \tag{6.45}$$

With $y=0$, (6.45) becomes

$$Q_1(0)=\Big(\frac{-y_2}{y_1-y_2}\Big)f^{-1}(y_1)+\Big(\frac{-y_1}{y_2-y_1}\Big)f^{-1}(y_2). \tag{6.46}$$

The numerical value of $Q_1(0)$, say x_3, may be used to estimate the root.

Using the inverse relationships, (6.46) may be rewritten as

$$x_3=\Big(\frac{-f(x_2)}{f(x_1)-f(x_2)}\Big)x_1+\Big(\frac{-f(x_1)}{f(x_2)-f(x_1)}\Big)x_2. \tag{6.47}$$

Algebra will reveal the secant method, see (4.17),

$$x_3=x_2-f(x_2)\cdot[\frac{x_2-x_1}{f(x_2)-f(x_1)}]. \tag{6.48}$$

This development shows how inverse linear interpolation may be used to compute an estimate for the root of a function. Inverse quadratic interpolation implemented by MATLAB uses quadratic rather than linear functions. The procedure is complicated by the possibility of two estimates for the root. Details may be found in more advanced texts on numerical analysis.

6.10 Problems

6-1. Determine c_1 and c_2 so that $F(x)=c_1x^2+c_2e^x$ interpolates the data $(\,-1,\,2\,)$ and $(\,2,\,5\,)$. Use matrix methods.

6- 2. Does $F(x)=3+(x-5)^2-2\sin(\pi x/2)$ interpolate the data $(\,4,\,4)$, $(5,\,1)$ and (7, 5)? Explain your reasoning.

6-3. Construct the Lagrange interpolating polynomials through the points.
a. $(2,-1),\ (4,5)$

b. $(2,-1)$, $(4,5)$, $(7,2)$
c. $(1,-2)$, $(3,0)$, $(4,3)$

6-4. Construct the Lagrange interpolating polynomial for the function and the given x-values. Graph the function and interpolating polynomial on the given interval.
a. $f(x) = \tan x,\ x = \pi/4, \pi/3, [\pi/4, \pi/3]$
b. $f(x) = \frac{1}{x}$, $x = 1,\ 3,\ 4,\ [1,4]$

6-5. Using the data of Problem 6-3 determine the Newton form of the interpolating polynomial. Construct the divided difference tables and verify the coefficients with the M-file divdif.m.

6-6. Convert the interpolating polynomial $P_2(x) = -\frac{5}{2}x^2 + \frac{19}{2}x - 8$ to the Newton form $P_2(x) = a_0 + a_1(x-1) + a_2(x-1)(x-2)$.

6-7. Construct the cubic interpolating polynomial for the data ($x_1,\ y_1$), $(x_1 + h,\ y_2$), $(x_1 + 2h,\ y_3$) and $(x_1 + 3h,\ y_4)$. Use the Lagrange form. Simplify the denominators of each term and then compute the derivative of the cubic interpolating polynomial. Remember the product rule:

$$\frac{d}{dx}[f(x)g(x)h(x)] = f'(x)g(x)h(x) + f(x)g'(x)h(x) + f(x)g(x)h'(x).$$

6-8. The error in linear interpolation is bounded. See equation (6.22). If you are using MATLAB's **interp1q** to estimate values using linear interpolation in data recorded from an experiment, what can you say about the interpolation errors?

6-9. Find bounds on the error of approximation for the two Lagrange interpolating polynomials in 6-4.

6-10. Suppose that we wish to construct a table of ln(x) on the interval $[\,1,\ 2\,]$. Determine the spacing, h, of x-values in the table so that we can be sure that linear interpolation errors are less than 0.0001.

6-11. Use MATLAB to approximate cos(x) with $P_4(x)$ using $\boldsymbol{x} = [\,0, \pi/2, \pi, 3\pi/2, 2\pi\,]$. Graph cos$(x)$, the approximation, $P_4(x)$, and the error cos$(x) - P_4(x)$.

6-12. Given n data points represented by the vectors x and y. State the MATLAB commands to compute the coefficients of an interpolating polynomial using all of the data and to evaluate the polynomial at π.

6-13. Determine which of the following piecewise functions are cubic splines. If the function is a spline, does it satisfy the conditions for a natural cubic spline?

a. $$f(x) = \begin{cases} \frac{19}{2} - \frac{81}{4}x + 15x^2 - \frac{13}{4}x^3 & 1 \le x \le 2 \\ \frac{-77}{2} + \frac{207}{4}x - 21x^2 + \frac{11}{4}x^3 & 2 \le x \le 3 \end{cases}$$

b. $f(x) = \begin{cases} 11 - 24x + 18x^2 - 4x^3 & 1 \le x \le 2 \\ -54 + 72x - 30x^2 + 4x^3 & 2 \le x \le 3 \end{cases}$

c $f(x) = \begin{cases} 18 - \frac{75}{2}x + 26x^2 - \frac{11}{2}x^3 & 1 \le x \le 2 \\ -70 + \frac{189}{2}x - 40x^2 + \frac{11}{2}x^3 & 2 \le x \le 3 \end{cases}$

d. $f(x) = \begin{cases} 13 - 31x + 23x^2 - 5x^3 & 1 \le x \le 2 \\ -35 + 51x - 22x^2 + 3x^3 & 2 \le x \le 3 \end{cases}$

6-14. A mysterious experiment produces the following data: $\boldsymbol{x} = [\,0,\ .05,\ .15, .2, 1.2, 1.5, 2\,]$ and $\boldsymbol{y} = [-1, 0, .8, 1.1, .8, .5, -.3]$.

a. Interpolate at $x = 1$ using **interp1q**, $P_6(x)$ and $S(x)$.

b. Plot a piecewise linear model for the data along with $P_6(x)$ and $S(x)$.

6-15. The force needed to stretch a spring from its normal length is given in the table.

$x\,(cm)$	3.0	3.5	4.0	4.5	5.0	6.0	7.0	8.0
$F\,(dynes)$	2.37	8.25	14.10	20.00	22.45	24.82	26.27	28.61

a. Interpolate at $x = 4.25$ and 6.7 using $P_7(x)$ and $S(x)$.

b. Estimate the spring constant dF/dx at $x = 4.25$ and 6.7 using derivatives of $P_7(x)$ and $S(x)$.

c. Are the results of parts a) and b) reasonable? Explain your reasoning.

6-16. Calibration data for a thermocouple is

$T\,(°F)$	45	50	55	60	70	80	90	100
$V\,(mV)$	3.49	3.17	2.93	2.73	1.76	1.63	1.41	1.36

a. Interpolate at $T = 66$ and 76 using $P_7(T)$ and $S(T)$.

b. Estimate dV/dT at $T = 66$ and 76 using derivatives of $P_7(T)$ and $S(T)$.

c. Are the results of parts a) and b) reasonable? Explain your reasoning.

6-17. The thermocouple in Problem 6-16 was defective. Repeat Problem 6-16 with revised data.

$T\,(°F)$	45	50	55	60	70	80	90	100
$V\,(mV)$	3.49	3.17	2.93	2.73	2.37	2.08	1.85	1.65

6-18. Consider the data $\boldsymbol{x} = [\,1,\ 3,\ 5,\ 7,\ 9\,]$ and $\boldsymbol{y} = [2,\ 4,\ 2,\ 3,\ 5\,]$. Construct the system of equations, (6.41), needed to solve for the not-a-knot spline coefficients. Solve the system using the usual backslash command and then compute the remaining coefficients for the spline. Verify your results by extracting the coefficient data from **spline(x,y)**.

6-19. A piecewise quadratic spline will consist of parabolas of the form $S_j(x) = a_j(x - x_j)^2 + b_j(x - x_j) + c_j,\ x_j \leq x \ \leq \ x_{j+1}$. For n data points determine the total number of unknowns. Count the number of equations if each $S_j(x)$ is required to interpolate at both ends of $x_j \leq x \ \leq \ x_{j+1}$ and $S'(x)$ is continuous at all interior nodes. How many additional requirements may be imposed?

6-20. A natural cubic spline requires that the second derivative of $S(x)$ be zero at both ends. Instead of zero at both endpoints other values may be chosen, for example, $S''(x_1) = K_1$ and $S''(x_n) = K_n$. Construct the tridiagonal system for this case showing how the values for K_1 and K_n enter the computations.

6-21. Show that the secant method (6.48) follows from (6.47).

6-22. Use equations (6.26), (6.27) and (6.28) with h replaced by $h_{j-1} = x_j - x_{j-1}$ to derive an equation for unequally spaced data that may be used in place of (6.33).