Chapter 3
Interpolation

Interpolation is the process of defining a function that takes on specified values at specified points. This chapter concentrates on two closely related interpolants, the piecewise cubic spline and the shape-preserving piecewise cubic named "pchip".

3.1 The Interpolating Polynomial

We all know that two points determine a straight line. More precisely, any two points in the plane, (x_1, y_1) and (x_2, y_2), with $x_1 \neq x_2$, determine a unique first degree polynomial in x whose graph passes through the two points. There are many different formulas for the polynomial, but they all lead to the same straight line graph.

This generalizes to more than two points. Given n points in the plane, $(x_k, y_k), k = 1, \ldots, n$, with distinct x_k's, there is a unique polynomial in x of degree less than n whose graph passes through the points. It is easiest to remember that n, the number of data points, is also the number of coefficients, although some of the leading coefficients might be zero, so the degree might actually be less than $n - 1$. Again, there are many different formulas for the polynomial, but they all define the same function.

This polynomial is called the *interpolating* polynomial because it exactly reproduces the given data.

$$P(x_k) = y_k, \;\; k = 1, \ldots, n$$

Later, we examine other polynomials, of lower degree, that only approximate the data. They are *not* interpolating polynomials.

The most compact representation of the interpolating polynomial is the *Lagrange* form.

$$P(x) = \sum_k \left(\prod_{j \neq k} \frac{x - x_j}{x_k - x_j} \right) y_k$$

There are n terms in the sum and $n-1$ terms in each product, so this expression defines a polynomial of degree at most $n-1$. If $P(x)$ is evaluated at $x = x_k$, all the products except the kth are zero. Furthermore, the kth product is equal to one, so the sum is equal to y_k and the interpolation conditions are satisfied.

For example, consider the following data set:

```
x = 0:3;
y = [-5  -6  -1  16];
```

The command

```
disp([x;  y])
```

displays

```
     0     1     2     3
    -5    -6    -1    16
```

The Lagrangian form of the polynomial interpolating this data is

$$\begin{aligned} P(x) &= \frac{(x-1)(x-2)(x-3)}{(-6)}(-5) + \frac{x(x-2)(x-3)}{(2)}(-6) + \\ &= \frac{x(x-1)(x-3)}{(-2)}(-1) + \frac{x(x-1)(x-2)}{(6)}(16) \end{aligned}$$

We can see that each term is of degree three, so the entire sum has degree at most three. Because the leading term does not vanish, the degree is actually three. Moreover, if we plug in $x = 0, 1, 2$, or 3, three of the terms vanish and the fourth produces the corresponding value from the data set.

Polynomials are usually not represented in their Lagrangian form. More frequently, they are written as something like

$$x^3 - 2x - 5$$

The simple powers of x are called *monomials* and this form of a polynomial is said to be using the *power form*.

The coefficients of an interpolating polynomial using its power form,

$$P(x) = c_1 x^{n-1} + c_2 x^{n-2} + \cdots + c_{n-1} x + c_n$$

can, in principle, be computed by solving a system of simultaneous linear equations

$$\begin{pmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \cdots & \cdots & \cdots & \cdots & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

The matrix V of this linear system is known as a *Vandermonde* matrix. Its elements are

$$v_{k,j} = x_k^{n-j}$$

The columns of a Vandermonde matrix are sometimes written in the opposite order, but polynomial coefficient vectors in MATLAB always have the highest power first.

The MATLAB function `vander` generates Vandermonde matrices. For our example data set,

```
V = vander(x)
```

generates

```
V =
     0     0     0     1
     1     1     1     1
     8     4     2     1
    27     9     3     1
```

Then

```
c = V\y'
```

computes the coefficients

```
c =
    1.0000
    0.0000
   -2.0000
   -5.0000
```

In fact, the example data was generated from the polynomial $x^3 - 2x - 5$.

One of the exercises asks you to show that Vandermonde matrices are nonsingular if the points x_k are distinct. But another one of the exercises asks you to show that a Vandermonde matrix can be very badly conditioned. Consequently, using the power form and the Vandermonde matrix is a satisfactory technique for problems involving a few well-spaced and well-scaled data points. But as a general-purpose approach, it is dangerous.

In this chapter, we describe several MATLAB functions that implement various interpolation algorithms. All of them have the calling sequence

```
v = interp(x,y,u)
```

The first two input arguments, `x` and `y`, are vectors of the same length that define the interpolating points. The third input argument, `u`, is a vector of points where the function is to be evaluated. The output, `v`, is the same length as `u` and has elements `v(k) = interp(x,y,u(k))`

Our first such interpolation function, `polyinterp`, is based on the Lagrange form. The code uses MATLAB array operations to evaluate the polynomial at all the components of `u` simultaneously.

```
function v = polyinterp(x,y,u)
n = length(x);
v = zeros(size(u));
```

```
for k = 1:n
   w = ones(size(u));
   for j = [1:k-1 k+1:n]
      w = (u-x(j))./(x(k)-x(j)).*w;
   end
   v = v + w*y(k);
end
```

To illustrate `polyinterp`, create a vector of densely spaced evaluation points.

```
u = -.25:.01:3.25;
```

Then

```
v = polyinterp(x,y,u);
plot(x,y,'o',u,v,'-')
```

creates figure 3.1.

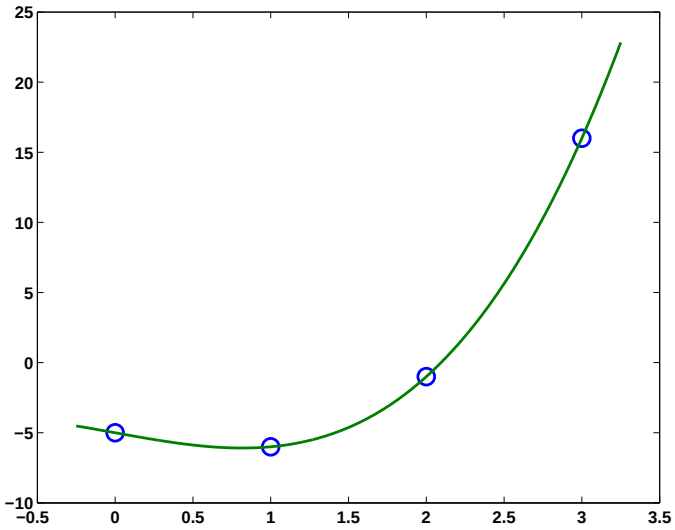

Figure 3.1. `polyinterp`

The `polyinterp` function also works correctly with symbolic variables. For example, create

```
symx = sym('x')
```

Then evaluate and display the symbolic form of the interpolating polynomial with

```
P = polyinterp(x,y,symx)
pretty(P)
```

produces

```
-5 (-1/3 x + 1)(-1/2 x + 1)(-x + 1) - 6 (-1/2 x + 3/2)(-x + 2)x
-1/2 (-x + 3)(x - 1)x + 16/3 (x - 2)(1/2 x - 1/2)x
```

This expression is a rearrangement of the Lagrange form of the interpolating polynomial. Simplifying the Lagrange form with

```
P = simplify(P)
```

changes `P` to the power form

```
P =
   x^3-2*x-5
```

Here is another example, with a data set that is used by the other methods in this chapter.

```
x = 1:6;
y = [16 18 21 17 15 12];
disp([x; y])
u = .75:.05:6.25;
v = polyinterp(x,y,u);
plot(x,y,'o',u,v,'-');
```

produces

```
 1     2     3     4     5     6
16    18    21    17    15    12
```

and figure 3.2.

Already in this example, with only six nicely spaced points, we can begin to see the primary difficulty with full-degree polynomial interpolation. In between the data points, especially in the first and last subintervals, the function shows excessive variation. It overshoots the changes in the data values. As a result, full-degree polynomial interpolation is hardly ever used for data and curve fitting. Its primary application is in the derivation of other numerical methods.

3.2 Piecewise Linear Interpolation

You can create a simple picture of the data set from the last section by plotting the data twice, once with circles at the data points and once with straight lines connecting the points. The following statements produce figure 3.3.

```
x = 1:6;
y = [16 18 21 17 15 12];
plot(x,y,'o',x,y,'-');
```

To generate the lines, the MATLAB graphics routines use *piecewise linear* interpolation. The algorithm sets the stage for more sophisticated algorithms. Three quantities are involved. The *interval index* k must be determined so that

$$x_k \le x < x_{k+1}$$

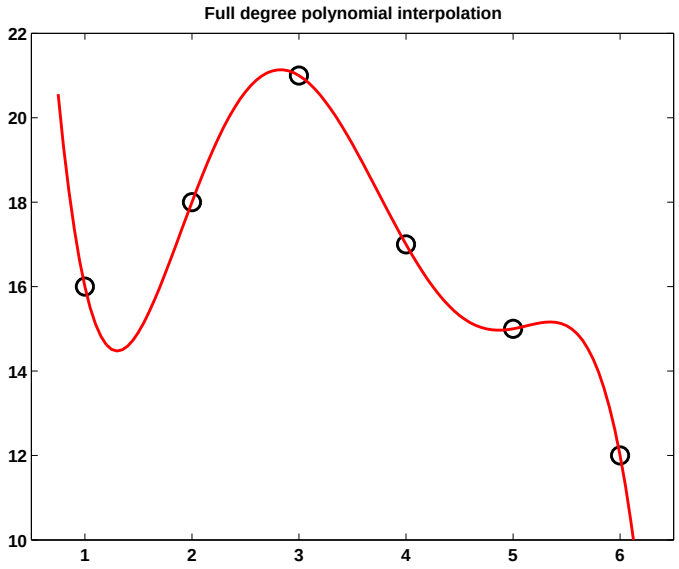

Figure 3.2. *Full degree polynomial interpolation*

The *local variable*, s, is

$$s = x - x_k$$

The *first divided difference* is

$$\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

With these quantities in hand, the interpolant is

$$\begin{aligned} L(x) &= y_k + (x - x_k)\frac{y_{k+1} - y_k}{x_{k+1} - x_k} \\ &= y_k + s\delta_k \end{aligned} \tag{3.1}$$

This is clearly a linear function that passes through (x_k, y_k) and (x_{k+1}, y_{k+1}).

The points x_k are sometimes called *breakpoints* or *breaks*. The piecewise linear interpolant $L(x)$ is a continuous function of x, but its first derivative, $L'(x)$, is not continuous. The derivative has a constant value, δ_k, on each subinterval and jumps at the breakpoints.

Piecewise linear interpolation is implemented in `piecelin.m`. The input `u` can be a vector of points where the interpolant is to be evaluated. In this case, the index `k` is actually a vector of indices. Read this code carefully to understand how `k` is computed.

```
function v = piecelin(x,y,u)
%PIECELIN  Piecewise linear interpolation.
```

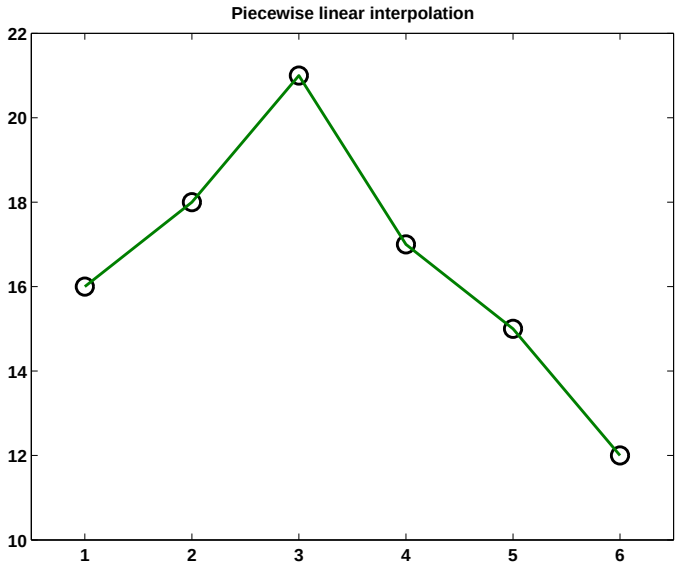

Figure 3.3. *Piecewise linear interpolation*

```
%  v = piecelin(x,y,u) finds the piecewise linear L(x)
%  with L(x(j)) = y(j) and returns v(k) = L(u(k)).

%  First divided difference

   delta = diff(y)./diff(x);

%  Find subinterval indices k so that x(k) <= u < x(k+1)

   n = length(x);
   k = ones(size(u));
   for j = 2:n-1
      k(x(j) <= u) = j;
   end

%  Evaluate interpolant

   s = u - x(k);
   v = y(k) + s.*delta(k);
```

3.3 Piecewise Cubic Hermite Interpolation

Many of the most effective interpolation techniques are based on piecewise cubic polynomials. Let h_k denote the length of the kth subinterval,

$$h_k = x_{k+1} - x_k$$

Then the first divided difference, δ_k, is

$$\delta_k = \frac{y_{k+1} - y_k}{h_k}$$

Let d_k denote the slope of the interpolant at x_k.

$$d_k = P'(x_k)$$

For the piecewise linear interpolant, $d_k = \delta_{k-1}$ or δ_k, but this is not necessarily true for higher order interpolants.

Consider the following function on the interval $x_k \le x \le x_{k+1}$, expressed in terms of local variables $s = x - x_k$ and $h = h_k$

$$\begin{aligned} P(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} \quad &+ \quad \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \\ \frac{s^2(s-h)}{h^2} d_{k+1} \quad &+ \quad \frac{s(s-h)^2}{h^2} d_k \end{aligned}$$

This is a cubic polynomial in s, and hence in x, that satisfies four interpolation conditions, two on function values and two on the possibly unknown derivative values.

$$\begin{aligned} P(x_k) = y_k, \;\; P(x_{k+1}) = y_{k+1} \\ P'(x_k) = d_k, \;\; P'(x_{k+1}) = d_{k+1} \end{aligned}$$

Functions that satisfy interpolation conditions on derivatives are known as *Hermite* or *osculatory* interpolants, because of the higher order contact at the interpolation sites. ("Osculari" means "to kiss" in Latin.)

If we happen to know both function values and first derivative values at a set of data points, then piecewise cubic Hermite interpolation can reproduce that data. But if we are not given the derivative values, we need to define the slopes d_k somehow. Of the many possible ways to do this, we will describe two, which MATLAB calls `pchip` and `spline`.

3.4 Shape Preserving Piecewise Cubic

The acronym `pchip` abbreviates "piecewise cubic Hermite interpolating polynomial." Although it is fun to say, the name does not specify which of the many possible interpolants is actually being used. In fact, spline interpolants are also piecewise cubic Hermite interpolating polynomials, but with different slopes. Our particular `pchip` is a shape-preserving, "visually pleasing" interpolant that was introduced into MATLAB fairly recently. It is based on an old Fortran program by

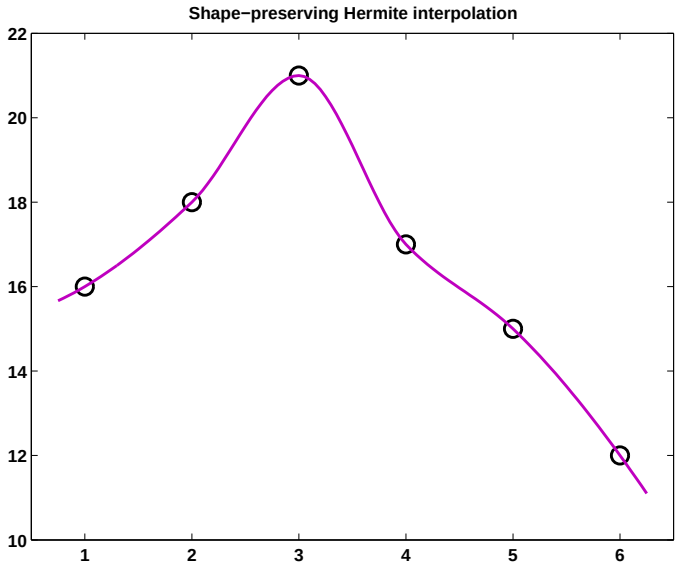

Figure 3.4. *Shape-preserving piecewise cubic Hermite interpolation*

Fritsch and Carlson that is described by Kahaner, Moler, and Nash [2]. Figure 3.4 shows how `pchip` interpolates our sample data.

The key idea is to determine the slopes d_k so that the function values do not overshoot the data values, at least locally. If δ_k and δ_{k-1} have opposite signs, or if either of them is zero, then x_k is a discrete local minimum or maximum, so we set

$$d_k = 0$$

This is illustrated in the first half of figure 3.5. The lower solid line is the piecewise linear interpolant. Its slopes on either side of the breakpoint in the center have opposite signs. Consequently, the dashed line has slope zero. The curved line is the shape-preserving interpolant, formed from two different cubics. The two cubics interpolate the center value and their derivatives are both zero there. But there is a jump in the second derivative at the breakpoint.

If δ_k and δ_{k-1} have the same signs and the two intervals have the same length, then d_k is taken to be the harmonic mean of the two discrete slopes.

$$\frac{1}{d_k} = \frac{1}{2}\left(\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k}\right)$$

In other words, at the breakpoint, the reciprocal slope of the Hermite interpolant is the average of the reciprocal slopes of the piecewise linear interpolant on either side. This is shown in the other half of figure 3.5. At the breakpoint, the reciprocal slope of the piecewise linear interpolant changes from 1 to 5. The reciprocal slope of the dashed line is 3, the average of 1 and 5. The shape-preserving interpolant is formed

from the two cubics that interpolate the center value and that have slope equal to 1/3 there. Again, there is a jump in the second derivative at the breakpoint.

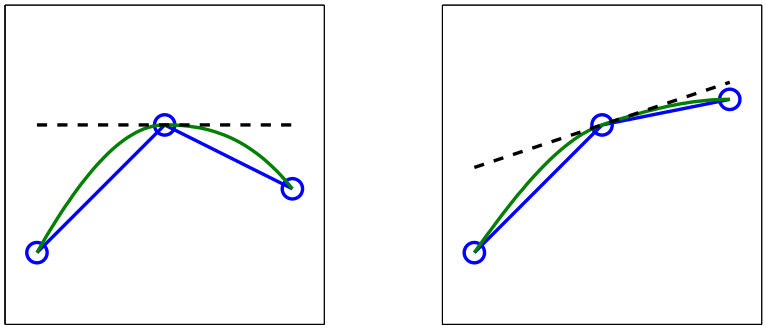

Figure 3.5. *Slopes for pchip*

If δ_k and δ_{k-1} have the same signs, but the two intervals have different lengths, then d_k is a weighted harmonic mean, with weights determined by the lengths of the two intervals.

$$\frac{w_1 + w_2}{d_k} = \frac{w_1}{\delta_{k-1}} + \frac{w_2}{\delta_k}$$

where

$$w_1 = 2h_k + h_{k-1}, \;\; w_2 = h_k + 2h_{k-1}$$

This defines the `pchip` slopes at interior breakpoints, but the slopes d_1 and d_n at either end of the data interval are determined by a slightly different, one-sided, analysis. The details are in `pchiptx.m`.

3.5 Cubic Spline

Our other piecewise cubic interpolating function is a *cubic spline*. The term "spline" refers to an instrument used in drafting. It is a thin, flexible wooden or plastic tool that is passed through given data points and that defines a smooth curve in between. The physical spline minimizes potential energy, subject to the interpolation constraints. The corresponding mathematical spline must have a continuous second derivative, and satisfy the same interpolation constraints. The breakpoints of a spline are also referred to as its *knots*.

The world of splines extends far beyond the basic one-dimensional, cubic, interpolatory spline we are describing here. There are multidimensional, high-order, variable knot, and approximating splines. A valuable expository and reference text for both the mathematics and the software is *A Practical Guide to Splines* by Carl de Boor [4]. De Boor is also the author of the `spline` function and the Spline Toolbox for MATLAB.

Figure 3.6 shows how `spline` interpolates our sample data.

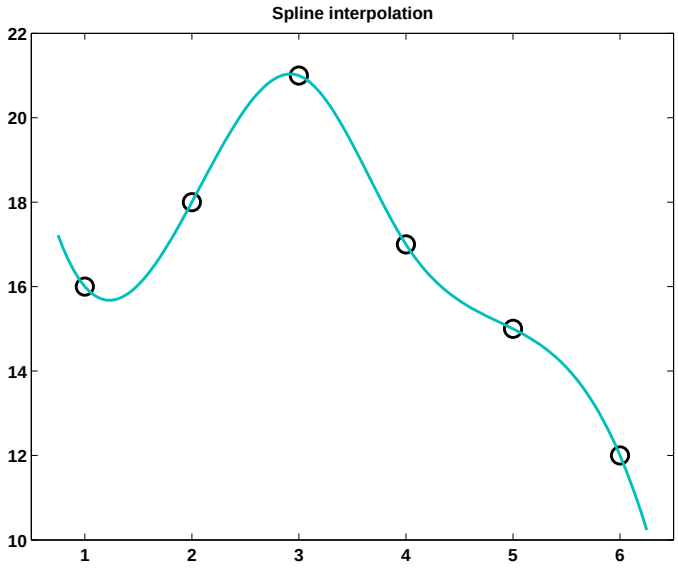

Figure 3.6. *Cubic spline interpolation*

The first derivative $P'(x)$ of our piecewise cubic function is defined by different formulas on either side of a knot, x_k. Both formulas yield the same value d_k at the knots, so $P'(x)$ is continuous.

On the kth subinterval, the second derivative is a linear function of $s = x - x_k$,

$$P''(x) = \frac{(6h - 12s)\delta_k + (6s - 2h)d_{k+1} + (6s - 4h)d_k}{h^2}$$

If $x = x_k$, $s = 0$ and

$$P''(x_k+) = \frac{6\delta_k - 2d_{k+1} - 4d_k}{h_k}$$

The plus sign in x_k+ indicates that this is a one-sided derivative. If $x = x_{k+1}$, $s = h_k$ and

$$P''(x_{k+1}-) = \frac{-6\delta_k + 4d_{k+1} + 2d_k}{h_k}$$

On the $(k-1)$st interval, $P''(x)$ is given by a similar formula involving δ_{k-1}, d_k and d_{k-1}. At the knot x_k,

$$P''(x_k-) = \frac{-6\delta_{k-1} + 4d_k + 2d_{k-1}}{h_{k-1}}$$

Requiring $P''(x)$ to be continuous at $x = x_k$ means that

$$P''(x_k+) = P''(x_k-)$$

This leads to the condition

$$h_k d_{k-1} + 2(h_{k-1} + h_k)d_k + h_{k-1}d_{k+1} = 3(h_k\delta_{k-1} + h_{k-1}\delta_k)$$

If the knots are equally spaced, so that h_k does not depend on k, this becomes

$$d_{k-1} + 4d_k + d_{k+1} = 3\delta_{k-1} + 3\delta_k$$

Like our other interpolants, the slopes d_k of a spline are closely related to the differences δ_k. In the spline case, they are a kind of running average of the δ_k's.

The preceding approach can be applied at each interior knot, x_k, $k = 2, \ldots, n-1$ to give $n-2$ equations involving the n unknowns d_k. As with `pchip`, a different approach must be used near the ends of the interval. One effective strategy is known as "not-a-knot." The idea is to use a single cubic on the first two subintervals, $x_1 \le x \le x_3$, and on the last two subintervals, $x_{n-2} \le x \le x_n$. In effect, x_2 and x_{n-1} are not knots. If the knots are equally spaced with all $h_k = 1$, this leads to

$$d_1 + 2d_2 = \frac{5}{2}\delta_1 + \frac{1}{2}\delta_2$$

and

$$2d_{n-1} + d_n = \frac{1}{2}\delta_{n-2} + \frac{5}{2}\delta_{n-1}$$

The details if the spacing is not equal to one are in `splinetx.m`.

With the two end conditions included, we have n linear equations in n unknowns:

$$Ad = r$$

The vector of unknown slopes is

$$d = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

The coefficient matrix A is tridiagonal.

$$A = \begin{pmatrix} h_2 & h_2 + h_1 & & & & \\ h_2 & 2(h_1 + h_2) & h_1 & & & \\ & h_3 & 2(h_2 + h_3) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-1} & 2(h_{n-2} + h_{n-1}) & h_{n-2} \\ & & & & h_{n-1} + h_{n-2} & h_{n-2} \end{pmatrix}$$

The right-hand side is

$$r = 3\begin{pmatrix} r_1 \\ h_2\delta_1 + h_1\delta_2 \\ h_3\delta_2 + h_2\delta_3 \\ \vdots \\ h_{n-1}\delta_{n-2} + h_{n-2}\delta_{n-1} \\ r_n \end{pmatrix}$$

The two values r_1 and r_n are associated with the end conditions.

If the knots are equally spaced with all $h_k = 1$, the coefficient matrix is quite simple.

$$A = \begin{pmatrix} 1 & 2 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ & & & & 2 & 1 \end{pmatrix}$$

The right-hand side is

$$r = 3 \begin{pmatrix} \frac{5}{6}\delta_1 + \frac{1}{6}\delta_2 \\ \delta_1 + \delta_2 \\ \delta_2 + \delta_3 \\ \vdots \\ \delta_{n-2} + \delta_{n-1} \\ \frac{1}{6}\delta_{n-2} + \frac{5}{6}\delta_{n-1} \end{pmatrix}$$

In our textbook function, `splinetx`, the linear equations defining the slopes are solved with the `tridisolve` function introduced in the chapter on linear equations. In the spline functions distributed with MATLAB and the Spline Toolbox, the slopes are computed by the MATLAB backslash operator,

$$d = A\backslash r$$

Because most of the elements of A are zero, it is appropriate to store A in a *sparse* data structure. The backslash operator can then take advantage of the tridiagonal structure and solve the linear equations in time and storage proportional to n, the number of data points.

In summary, figure 3.7 shows how the four interpolants handle our example data.

These figures illustrate the tradeoff between smoothness and a somewhat subjective property that we might call *local monotonicity* or *shape preservation*.

The piecewise linear interpolant is at one extreme. It has hardly any smoothness. It is continuous, but there are jumps in its first derivative. On the other hand, it preserves the local monotonicity of the data. It never overshoots the data and it is increasing, decreasing, or constant on the same intervals as the data.

The full-degree polynomial interpolant is at the other extreme. It is infinitely differentiable. But it often fails to preserve shape, particularly near the ends of the interval.

The `pchip` and `spline` interpolants are in between these two extremes. The spline is smoother than `pchip`. The spline has two continuous derivatives, while `pchip` has only one. A discontinuous second derivative implies discontinuous curvature. The human eye can detect large jumps in curvature in graphs and in mechanical parts made by numerically controlled machine tools. On the other hand, `pchip` is guaranteed to preserve shape, but the spline might not.

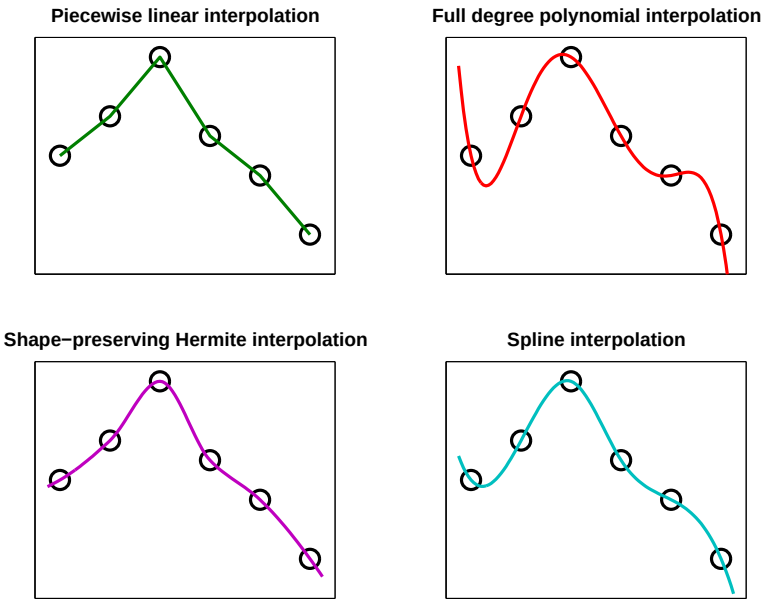

Figure 3.7. *Four interpolants*

3.6 `pchiptx`, `splinetx`

The M-files `pchiptx` and `splinetx` are both based on piecewise cubic Hermite interpolation. On the kth interval, this is

$$P(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where $s = x - x_k$ and $h = h_k$. The two functions differ in the way they compute the slopes, d_k. Once the slopes have been computed, the interpolant can be efficiently evaluated using the power form with the local variable s.

$$P(x) = y_k + sd_k + s^2 c_k + s^3 b_k$$

where the coefficients of the quadratic and cubic terms are

$$c_k = \frac{3\delta_k - 2d_k - d_{k+1}}{h}$$
$$b_k = \frac{d_k - 2\delta_k + d_{k+1}}{h^2}$$

Here is the first portion of code for `pchiptx`. It calls an internal subfunction to compute the slopes, then computes the other coefficients, finds a vector of interval

indices, and evaluates the interpolant. After the preamble, this part of the code for `splinetx` is the same.

```
function v = pchiptx(x,y,u)
%PCHIPTX  Textbook piecewise cubic Hermite interpolation.
%  v = pchiptx(x,y,u) finds the shape-preserving piecewise
%  P(x), with P(x(j)) = y(j), and returns v(k) = P(u(k)).
%
%  See PCHIP, SPLINETX.

%  First derivatives

   h = diff(x);
   delta = diff(y)./h;
   d = pchipslopes(h,delta);

%  Piecewise polynomial coefficients

   n = length(x);
   c = (3*delta - 2*d(1:n-1) - d(2:n))./h;
   b = (d(1:n-1) - 2*delta + d(2:n))./h.^2;

%  Find subinterval indices k so that x(k) <= u < x(k+1)

   k = ones(size(u));
   for j = 2:n-1
      k(x(j) <= u) = j;
   end

%  Evaluate interpolant

   s = u - x(k);
   v = y(k) + s.*(d(k) + s.*(c(k) + s.*b(k)));
```

The code for computing the `pchip` slopes uses the weighted harmonic mean at interior breakpoints and a one-sided formula at the endpoints.

```
function d = pchipslopes(h,delta);
%  PCHIPSLOPES  Slopes for shape-preserving Hermite cubic.
%  pchipslopes(h,delta) computes d(k) = P'(x(k)).

   n = length(h)+1;
   d = zeros(size(h));
   k = find(sign(delta(1:n-2)).*sign(delta(2:n-1)) > 0) + 1;
   w1 = 2*h(k)+h(k-1);
   w2 = h(k)+2*h(k-1);
   d(k) = (w1+w2)./(w1./delta(k-1) + w2./delta(k));
```

```
%  Slopes at endpoints

   d(1) = pchipendpoint(h(1),h(2),delta(1),delta(2));
   d(n) = pchipendpoint(h(n-1),h(n-2),delta(n-1),delta(n-2));

function d = pchipendpoint(h1,h2,del1,del2)
%  Noncentered, shape-preserving, three-point formula.
   d = ((2*h1+h2)*del1 - h1*del2)/(h1+h2);
   if sign(d) ~= sign(del1)
      d = 0;
   elseif (sign(del1) ~= sign(del2)) & (abs(d) > abs(3*del1))
      d = 3*del1;
   end
```

The `splinetx` M-file computes the slopes by setting up and solving a tridiagonal system of simultaneous linear equations.

```
function d = splineslopes(h,delta);
%  SPLINESLOPES  Slopes for cubic spline interpolation.
%  splineslopes(h,delta) computes d(k) = S'(x(k)).
%  Uses not-a-knot end conditions.

%  Diagonals of tridiagonal system

   n = length(h)+1;
   a = zeros(size(h)); b = a; c = a; r = a;
   a(1:n-2) = h(2:n-1);
   a(n-1) = h(n-2)+h(n-1);
   b(1) = h(2);
   b(2:n-1) = 2*(h(2:n-1)+h(1:n-2));
   b(n) = h(n-2);
   c(1) = h(1)+h(2);
   c(2:n-1) = h(1:n-2);

%  Right-hand side

   r(1) = ((h(1)+2*c(1))*h(2)*delta(1)+h(1)^2*delta(2))/c(1);
   r(2:n-1) = 3*(h(2:n-1).*delta(1:n-2)+ ...
                 h(1:n-2).*delta(2:n-1));
   r(n) = (h(n-1)^2*delta(n-2)+ ...
                 (2*a(n-1)+h(n-1))*h(n-2)*delta(n-1))/a(n-1);

%  Solve tridiagonal linear system

   d = tridisolve(a,b,c,r);
```

3.7 interpgui

The M-file `interpgui` allows you to experiment with the four interpolants discussed in this chapter.

- Piecewise linear interpolant
- Full-degree interpolating polynomial
- Piecewise cubic spline
- Shape-preserving piecewise cubic

The program can be initialized in several different ways.

- With no arguments, `interpgui` starts with six zeros.
- With a scalar argument, `interpgui(n)` starts with `n` zeros.
- With one vector argument, `interpgui(y)` starts with equally spaced `x`'s.
- With two arguments, `interpgui(x,y)` starts with a plot of `y` vs. `x`.

After initialization, the interpolation points can be varied with the mouse. If `x` has been specified, it remains fixed. Figure 3.8 is the initial plot generated by our example data.

Exercises

3.1. Reproduce figure 3.7, with four subplots showing the four interpolants discussed in this chapter.

3.2. Tom and Ben are twin boys born on October 27, 2001. Here is a table of their weights, in pounds and ounces, over their first few months.

```
%       Date       Tom     Ben
W = [10 27 2001    5 10    4  8
     11 19 2001    7  4    5 11
     12 03 2001    8 12    6  4
     12 20 2001   10 14    8  7
     01 09 2002   12 13   10  3
     01 23 2002   14  8   12  0
     03 06 2002   16 10   13 10];
```

You can use `datenum` to convert the date in the first three columns to a serial date number measuring time in days.

```
t = datenum(W(:,[3 1 2]));
```

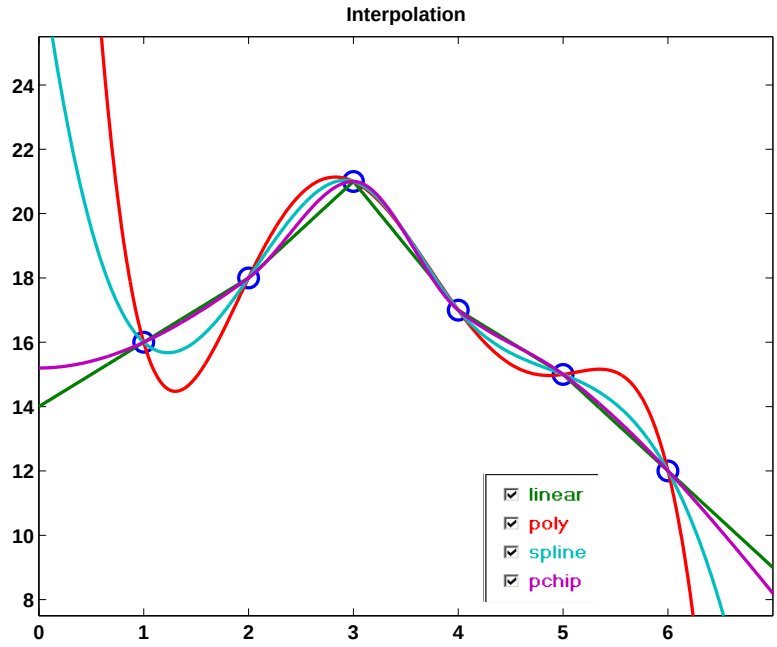

Figure 3.8. `interpgui`

Make a plot of their weights versus time, with circles at the data points and the `pchip` interpolating curve in between. Use `datetick` to relabel the time axis. Include a `title` and a `legend`. The result should look something like figure 3.9.

3.3. (a) Interpolate this data by each of the four interpolants discussed in this chapter, `piecelin`, `polyinterp`, `splinetx`, and `pchiptx`. Plot the results for $-1 \le x \le 1$.

```
    x         y
-1.00   -1.0000
-0.96   -0.1512
-0.65    0.3860
 0.10    0.4802
 0.40    0.8838
 1.00    1.0000
```

(b) What are the values of each of the four interpolants at $x = -.3$? Which of these values do you prefer? Why?
(c) The data was actually generated from a low-degree polynomial with integer coefficients. What is that polynomial?

3.4. Make a plot of your hand. Start with

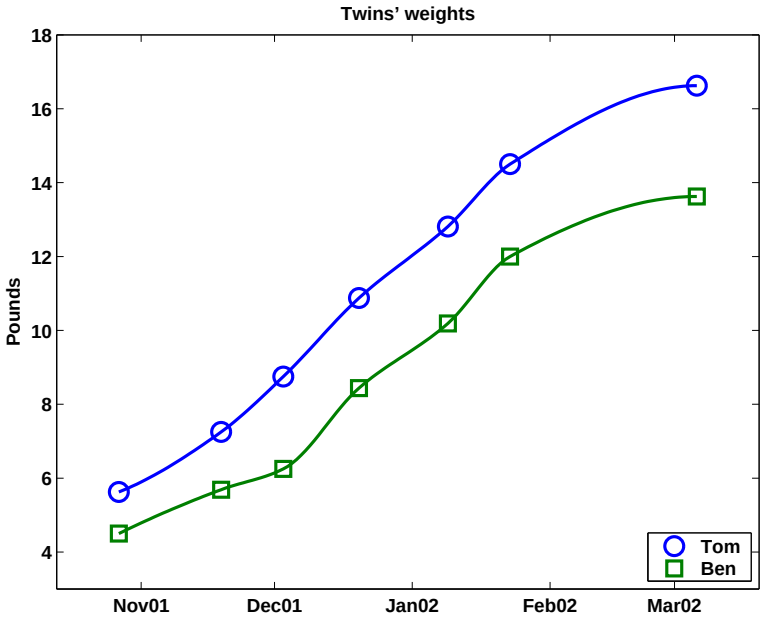

Figure 3.9. *Twins' weights*

```
figure('position',get(0,'screensize'))
axes('position',[0 0 1 1])
[x,y] = ginput;
```

Place your hand on the computer screen. Use the mouse to select a few dozen points outlining your hand. Terminate the `ginput` with a carriage return. You might find it easier to trace your hand on a piece of paper and then put the paper on the computer screen. You should be able to see the `ginput` cursor through the paper. (Save this data. We will refer to it in other exercises later in this book.)
Now think of `x` and `y` as two functions of an independent variable that goes from one to the number of points you collected. You can interpolate both functions on a finer grid and plot the result with

```
n = length(x);
s = (1:n)';
t = (1:.05:n)';
u = splinetx(s,x,t);
v = splinetx(s,y,t);
clf reset
plot(x,y,'.',u,v,'-');
```

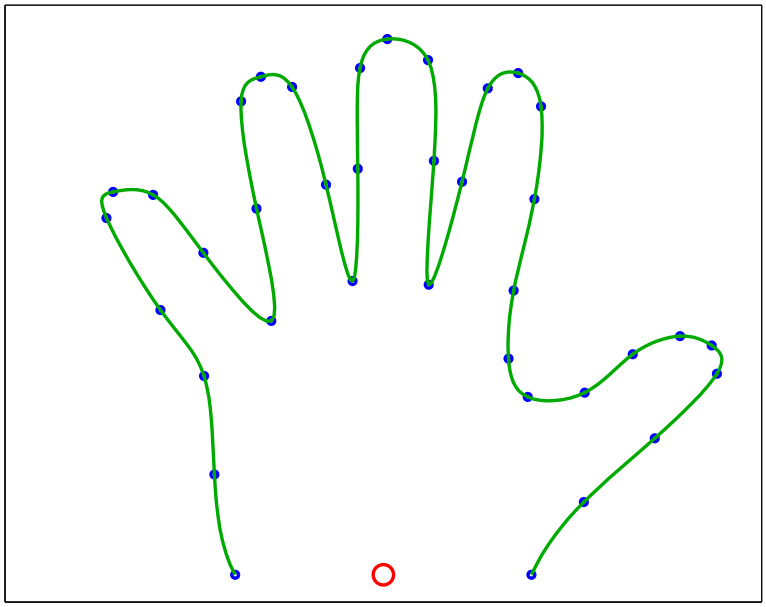

Figure 3.10. *A hand*

Do the same thing with `pchiptx`. Which do you prefer?
Figure 3.10 is the plot of my hand. Can you tell if it is done with `splinetx` or `pchiptx`?

3.5. The previous exercise uses the data index number as the independent variable for two-dimensional parametric interpolation. This exercise uses, instead, the angle θ from polar coordinates. In order to do this, the data must be centered so that it lies on a curve that is *starlike* with respect to the origin, that is every ray emanating from the origin meets the data only once. This means that you must be able to find values x_0 and y_0 so that the MATLAB statements

```
x = x - x0
y = y - y0
theta = atan2(y,x)
r = sqrt(x.^2 + y.^2)
plot(theta,r)
```

produce a set of points that can be interpolated with a single-valued function, $r = r(\theta)$. For the data obtained by sampling the outline of your hand, the point (x_0, y_0) is located near the base of your palm. See the small circle in figure 3.10. Furthermore, in order to use `splinetx` and `pchiptx`, it is also necessary to order the data so that `theta` is monotonically increasing.
Choose a subsampling increment, `delta`, and let

```
t = (theta(1):delta:theta(end))';
```

```
p = pchiptx(theta,r,t);
s = splinetx(theta,r,t);
```

Examine two plots,

```
plot(theta,r,'o',t,[p s],'-')
```

and

```
plot(x,y,'o',p.*cos(t),p.*sin(t),'-',s.*cos(t),s.*sin(t),'-')
```

Compare this approach with the one used in the previous exercise. Which do you prefer? Why?

3.6. This exercise requires the Symbolic Toolbox.
(a) What does `vandal(n)` compute and how does it compute it?
(b) Under what conditions on `x` is the matrix `vander(x)` nonsingular?

3.7. Prove that the interpolating polynomial is unique. That is, if $P(x)$ and $Q(x)$ are two polynomials with degree less than n that agree at n distinct points, then they agree at all points.

3.8. Give convincing arguments that each of the following descriptions defines the *same* polynomial, the Chebyshev polynomial of degree five, $T_5(x)$. Your arguments can involve analytic proofs, symbolic computation, numeric computation, or all three. Two of the representations involve the golden ratio,

$$\phi = \frac{1+\sqrt{5}}{2}$$

(a) Power form basis

$$T_5(x) = 16x^5 - 20x^3 + 5x$$

(b) Relation to trigonometric functions

$$T_5(x) = \cos\left(5\cos^{-1} x\right)$$

(c) Horner representation

$$T_5(x) = ((((16x+0)x-20)x+0)x+5)x+0$$

(d) Lagrange form

$$\begin{aligned} x_1, x_6 &= \pm 1 \\ x_2, x_5 &= \pm\phi/2 \\ x_3, x_4 &= \pm(\phi-1)/2 \end{aligned}$$

$$y_k = (-1)^k, \ k = 1, \ldots, 6$$

$$T_5(x) = \sum_k \left(\prod_{j \neq k} \frac{x - x_j}{x_k - x_j} \right) y_k$$

(e) Factored representation

$$\begin{aligned} z_1, z_5 &= \pm\sqrt{(2+\phi)/4} \\ z_2, z_4 &= \pm\sqrt{(3-\phi)/4} \\ z_3 &= 0 \end{aligned}$$

$$T_5(x) = 16\prod_1^5 (x - z_k)$$

(f) Three term recurrence

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x), \text{ for } n = 2, \ldots, 5 \end{aligned}$$

3.9. The M-file `rungeinterp.m` provides an experiment with a famous polynomial interpolation problem due to Carl Runge. Let

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

and let $P_n(x)$ denote the polynomial of degree $n-1$ that interpolates $f(x)$ at n equally spaced points on the interval $-1 \le x \le 1$. Runge asked whether as n increases, does $P_n(x)$ converge to $f(x)$? The answer is yes for some x, but no for others.
(a) For what x does $P_n(x) \to f(x)$ as $n \to \infty$?
(b) Change the distribution of the interpolation points so that they are not equally spaced. How does this affect convergence? Can you find a distribution so that $P_n(x) \to f(x)$ for all x in the interval?

3.10. We skipped from piecewise linear to piecewise cubic interpolation. How far can you get with the development of piecewise quadratic interpolation?

3.11. Modify `splinetx` and `pchiptx` so that, if called with two output arguments, they produce both the value of the interpolant and its first derivative. That is,

```
[v,vprime] = pchiptx(x,y,u)
```

and

```
[v,vprime] = splinetx(x,y,u)
```

compute $P(u)$ and $P'(u)$.

3.12. Modify `splinetx` and `pchiptx` so that, if called with only two input arguments, they produce `PP`, the piecewise polynomial structure produced by the standard MATLAB functions `spline` and `pchip` and used by `ppval`.

3.13. (a) Create two functions `perpchip` and `perspline` by modifying `pchiptx` and `splinetx` to replace the one-sided and not-a-knot end conditions with *periodic* boundary conditions. This requires that the given data has

$$y_n = y_1$$

and that the resulting interpolant is periodic. In other words, for all x,

$$P(x + \Delta) = P(x)$$

where

$$\Delta = x_n - x_1$$

The algorithms for both pchip and spline involve calculations with y_k, h_k, and δ_k to produce slopes d_k. With the periodicity assumption, all of these quantities become periodic functions, with period $n - 1$, of the subscript k. In other words, for all k,

$$\begin{aligned} y_k &= y_{k+n-1} \\ h_k &= h_{k+n-1} \\ \delta_k &= \delta_{k+n-1} \\ d_k &= d_{k+n-1} \end{aligned}$$

This makes it possible to use the same calculations at the endpoints that are used at the interior points in the nonperiodic situation. The special case code for the end conditions can be eliminated and the resulting M-files are actually much simpler.
For example, the slopes d_k for pchip with equally spaced points, are given by

$$d_k = 0, \text{if } \operatorname{sign}(\delta_{k-1}) \neq \operatorname{sign}(\delta_k),$$

$$\frac{1}{d_k} = \frac{1}{2}\left(\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k}\right), \text{if } \operatorname{sign}(\delta_{k-1}) = \operatorname{sign}(\delta_k),$$

With periodicity these formulas can also be used at the end points where $k = 1$ and $k = n$ because

$$\delta_0 = \delta_{n-1}, \text{and } \delta_n = \delta_1.$$

For spline, the slopes satisfy a system of simultaneous linear equations, for $k = 2, \ldots, n - 1$.

$$h_k d_{k-1} + 2(h_{k-1} + h_k)d_k + h_{k-1}d_{k+1} = 3(h_k\delta_{k-1} + h_{k-1}\delta_k)$$

With periodicty, at $k = 1$ this becomes

$$h_1 d_{n-1} + 2(h_{n-1} + h_1)d_1 + h_{n-1}d_2 = 3(h_1\delta_{n-1} + h_{n-1}\delta_1)$$

and at $k = n$,

$$h_n d_{n-1} + 2(h_{n-1} + h_1)d_n + h_{n-1}d_2 = 3(h_1\delta_{n-1} + h_{n-1}\delta_1)$$

The resulting matrix has two nonzero elements outside the tridiagonal structure. The additional nonzero elements in the first and last rows are $A_{1,n-1} = h_1$ and $A_{n,2} = h_{n-1}$
(b) Once you have `perchip` and `perspline`, you can use the NMC mfile `interp2dgui` to investigate closed curve interpolation in two dimensions. You should find that the periodic boundary conditions do a better job of reproducing symmetries of closed curves in the plane.

3.14. (a) Modify `splinetx` so that it forms the full tridiagonal matrix

```
A = diag(a,-1) + diag(b,0) + diag(c,1)
```

and uses backslash to compute the slopes.
(b) Monitor `condest(A)` as the spline knots are varied with `interpgui`. What happens if two of the knots approach each other? Find a data set that makes `condest(A)` large.

3.15. Modify `pchiptx` so that it uses a weighted average of the slopes instead of the weighted harmonic mean.

3.16. (a) Consider

```
x = -1:1/3:1
interpgui(1-x.^2)
```

Which, if any, of the four interpolants `linear`, `spline`, `pchip`, and `polynomial` are the same? Why?
(b) Same questions for

```
interpgui(1-x.^4)
```

3.17. Why does `interpgui(4)` show only three graphs, not four, no matter where you move the points?

3.18. Use the Symbolic Toolbox to study the key properties of $P(x)$, the function used section 3.3 on piecewise cubic Hermite interpolation. You should verify the equations given for $P''(x)$ and for the values of $P(x)$, $P'(x)$, and $P''(x)$ at the breakpoints x_k. You might start with something like

```
syms x s h y1 y0 d1 d0
P = (3*h*s^2-2*s^3)/h^3*y1 + (h^3-3*h*s^2+2*s^3)/h^3*y0 ...
     + s^2*(s-h)/h^2*d1 + s*(s-h)^2/h^2*d0
```

and then look at expressions such as

```
subs(P,s,0)
subs(diff(P,s,2),s,h)
```

The symbolic functions `simplify` and `pretty` might come in handy.
Demonstrate that your new functions work correctly on

```
x = [0 2*sort(rand(1,6))*pi 2*pi];
y = cos(x + 2*rand*pi);
u = 0:pi/50:2*pi;
v = your_function(x,y,u);
plot(x,y,'o',u,v,'-')
```

3.19. (a) If you want to interpolate census data on the interval $1900 \le t \le 2000$ with a polynomial,

$$P(t) = c_1 t^{10} + c_2 t^9 + \ldots + c_{10} t + c_{11}$$

you might be tempted to use the Vandermonde matrix generated by

```
t = 1900:10:2000
V = vander(t)
```

Why is this a really bad idea?
(b) Investigate *centering* and *scaling* the independent variable. If you are using MATLAB 6, plot some data, pull down the Tools menu on the figure window, select Basic Fitting, and find the check box about centering and scaling. What does this check box do?
(c) Replace the variable t by

$$s = \frac{t - \mu}{\sigma}$$

This leads to a modified polynomial $\tilde{P}(s)$. How are its coefficients related to those of $P(t)$? What happens to the Vandermonde matrix? What values of μ and σ lead to a reasonably well conditioned Vandermonde matrix? One possibility is

```
mu = mean(t)
sigma = std(t)
```

but are there better values?

Bibliography

[1] G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Prentice Hall, Englewood Cliffs, 1977.

[2] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Prentice Hall, Englewood Cliffs, 1989.

[3] The MathWorks, Inc., *Numerical Computing with MATLAB*, `http://www.mathworks.com/moler`

[4] C. de Boor, *A Practical Guide to Splines*, Springer Verlag, New York, 1978.