Polynomials and Splines

1 The Interpolating Polynomial

We all know that two points determine a 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 linear function of \(x\) whose graph passes through the two points. There are many different formulas for the linear function, but they all lead to the same 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 at most \(n - 1\) 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 is actually 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.

\[
P(x_k) = y_k, \quad k = 1, \ldots, n
\]

Later, we will 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 Lagrange’s interpolation formula.

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

There are \(n\) terms in the sum and \(n - 1\) terms in each product, so this expression defines a polynomial of degree at most \(n - 1\). When \(P(x)\) is evaluated at \(x = x_k\), all the products except the \(k\)-th are zero. Furthermore, the \(k\)-th 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

\[
\text{disp([x; y])}
\]

displays
The Lagrangian polynomial interpreting this data is

\[ P(x) = \frac{(x-1)(x-2)(x-3)}{(-6)}(-5) + \frac{x(x-2)(x-3)}{(2)}(-6) + \]
\[ \frac{x(x-1)(x-3)}{(-2)}(-1) + \frac{x(x-1)(x-2)}{(6)}(16) \]

You can see that each term is a cubic, so the entire sum is a cubic. Moreover, when you 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, polynomials appear 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 monomial basis.

The coefficients of an interpolating polynomial using its monomial basis

\[ P(x) = c_1x^{n-1} + c_2x^{n-2} + \cdots + c_{n-1}x + c_n \]

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

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

The matrix \( V \) of this linear system is known as a Vandermonde matrix. In MATLAB its elements are

\[ v_{k,j} = x_k^{n-j} \]

These columns are reversed from their conventional order because MATLAB’s polynomial coefficient vectors have the highest power first.

The MATLAB function \texttt{vander} generates Vandermonde matrices. For our example data set

\[
V = \text{vander}(x)
\]

generates

\[
V = 
\begin{pmatrix}
  0 & 0 & 0 & 1 \\
  1 & 1 & 1 & 1 \\
  8 & 4 & 2 & 1 \\
  27 & 9 & 3 & 1 \\
\end{pmatrix}
\]
Then
\[ c = V\backslash y' \]
computes the coefficients
\[
\begin{align*}
c &= \\
&= 1.0000 \\
&= 0.0000 \\
&= -2.0000 \\
&= -5.0000
\end{align*}
\]
In fact, the example data was generated from the polynomial \( x^3 - 2x - 5 \).

One of the exercises will ask you to show that Vandermonde matrices are nonsingular if the points \( x_k \) are distinct. But, another one of the exercises will ask you to show that Vandermonde matrix can be very badly conditioned. Consequently, using the monomial basis 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 will describe several MATLAB functions that implement various interpolation algorithms. All of them have the calling sequence
\[
v = \text{interp}(x, y, u)
\]
The first input two 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 \( v(k) = \text{interp}(x, y, u(k)) \).

Our first such interpolation function, \texttt{polyinterp}, is based on Lagrange’s formula. The code uses MATLAB’s array operations to evaluate the polynomial at all the components of \( u \) simultaneously.

```matlab
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 \texttt{polyinterp}, create a vector of densely spaced evaluation points.
\[
u = -.25:.01:3.25;
\]
Then
\[
v = \text{polyinterp}(x, y, u);
\]
\[
\text{plot}(x, y, 'o', u, v, '-')
\]
The `polyinterp` function will also work correctly with symbolic variables. For example, create

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

Then evaluate and display the symbolic form of the interpolating polynomial

```matlab
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 this Lagrange form with

```matlab
P = simplify(P);
```

changes P to use the monomial basis

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

Here is another example, with a data set that will also be used with the other interpolating functions in this chapter.

```matlab
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

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
16 & 18 & 21 & 17 & 15 & 12 \\
\end{array}
\]

and

Already in this example, with only six well-behaved 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 oscillation. 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.

2 Piecewise linear interpolation

A simple picture of the data set from the last section can be produced by plotting the data twice, once with circles at the data points and once with straight lines connecting the points.

```matlab
x = 1:6;
y = [16 18 21 17 15 12];
plot(x,y,'o',x,y,'-');
```
To produce the lines connecting the points, the MATLAB graphics routines use piecewise linear interpolation. The formula for piecewise linear interpolation sets the stage for more sophisticated algorithms.

\[ L(x) = y_k + sd_k \]

Three quantities are involved. The interval index \( k \) must be determined so that

\[ x_k \leq x < x_{k+1} \]

The local variable, \( s \), is

\[ s = x - x_k \]

The first divided difference is

\[ d_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \]

With these quantities in hand

\[ L(x) = y_k + (x - x_k) \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \]

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 knots. 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, \( d_k \), on each subinterval and jumps at the knots.

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,sigma] = piecelin(x,y,u)
    %PIECELIN Piecewise linear interpolation.
    % v = piecelin(x,y,u) computes v(k) = L(u(k))
    % where L(x) is the piecewise linear interpolant.
    % First divided difference
    d = diff(y)./diff(x);
    % Find subinterval indices, x(k) <= u < u(k+1)
    n = length(x);
    k = ones(size(u));
    for j = 2:n-1
        k(u >= x(j)) = j;
    end
    % Evaluate interpolant
    s = u - x(k);
    v = y(k) + s.*d(k);

3 Piecewise cubic interpolation

Let $h_k$ denote the length of the $k$-th subinterval,

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

and $\delta_k$ denote the divided difference,

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

We will now let $d_k$ denote the slope of the interpolant at $x_k$. For the piecewise linear interpolant, $d_k = \delta_{k-1}$ or $\delta_k$, but is this not necessarily true for higher order interpolants.

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

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

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.

$$P(x_k) = y_k, \quad P(x_{k+1}) = y_{k+1}$$
$$P'(x_k) = d_k, \quad P'(x_{k+1}) = d_{k+1}$$
Functions that satisfy interpolation conditions on derivatives are known as Hermite or osculatory interpolants. ("Oscularii" 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 \texttt{pchip} and \texttt{spline}.

The acronym \texttt{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 \texttt{pchip} is a shape-preserving, "visually pleasing" interpolant that was introduced into MATLAB fairly recently. It is based on an old Fortran program by Fritsch and Carlson that is described in \textit{Numerical Methods and Software} by Kahaner, Moler and Nash. Here is how \texttt{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 $\delta_k$ and $\delta_{k-1}$ have opposite signs, or if either of them is zero, then $x_k$ is a discrete local minimum or maximum, so we set

$$d_k = 0$$

If $\delta_k$ and $\delta_{k-1}$ 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 break point, the reciprocal slope of the Hermite interpolant is the average of the reciprocal slopes of the piecewise linear interpolant on either side.

This is illustrated in the following figure.

The lower solid line is the piecewise linear interpolant. At the knot in the center, its reciprocal slope changes from 1 to 5. The reciprocal slope of the dashed line is 3, the average of 1 and 5. The curved line is the shape preserving interpolant, formed from two different cubics. The two cubics interpolate the center value and their derivatives both equal $1/3$ there. But there is a jump in the second derivative at the knot.

If $\delta_k$ and $\delta_{k-1}$ 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

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

This defines the \texttt{pchip} slopes at interior knots, but the slopes $d_1$ and $d_n$ at either end of data interval are determined by a slightly different, one-sided, analysis. The details are in \texttt{pchipx.m}.

Our other piecewise cubic interpolating function is a \textit{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 world of splines extends far beyond the basic one-dimensional, cubic, interpolatory function we are describing here. There are multi-dimensional, high order, variable knot, 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. De Boor is also the author of the `spline` function and the Spline Toolbox for MATLAB.

Here is how `spline` interpolates our sample data.

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 \( k \)-th subinterval, the second derivative is linear function of \( s = x - x_k \)

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

When \( x = x_k \), \( s = 0 \) 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. When \( x = x_{k+1} \), \( s = h_k \) 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 \(\delta_{k-1}, d_k\) 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 \(x = x_k\) leads to

\[
h_kd_{k-1} + 2(h_k + h_k)d_k + h_kd_{k-1}d_{k+1} = 3h_k\delta_{k-1} + 3h_k\delta_{k-1}
\]

When the knots are equally spaced, so that \(h_k\) does not depend upon \(k\), 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 \(\delta_k\). In the spline case, they are a kind of running average of the \(\delta_k\)'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 let use a single cubic on the first two subintervals, \(x_1 \leq x \leq x_3\). In effect, \(x_2\) is not a knot. When \(h_1 = h_2 = 1\), this leads

\[
d_1 + 2d_2 = \frac{5}{2}h_1 + \frac{1}{2}h_2
\]

The last two subintervals are treated similarly, so \(x_{n-1}\) is not a knot 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

\[
Ad = r
\]

Here the vector of unknown slopes is

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

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. The coefficient matrix \( A \) is tridiagonal.

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

The slopes are ultimately computed with MATLAB’s backslash operator.

\[
d = A\backslash r
\]

Since most of the elements of \( A \) are zero, it is appropriate to store \( A \) with MATLAB’s 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, here is 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 \texttt{pchiptx} and \texttt{splinetx} 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.

4 \texttt{pchiptx} and \texttt{splinetx}

The M-files \texttt{pchiptx} and \texttt{splinetx} are both based on piecewise cubic Hermite interpolation. On the $k$-th 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 a local monomial basis.

\[
P(x) = y_k + s d_k + s^2 c_k + s^3 b_k
\]

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 \texttt{pchiptx}. It calls upon 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, the code for \texttt{splinetx} is the same.

```matlab
function [v,d] = pchiptx(x,y,u)
% PCHIPTX Textbook piecewise cubic Hermite interpolation.
% v = pchiptx(x,y,u) finds the shape preserving piecewise cubic interpolant P(x), with P(x(j)) = y(j), and returns v(k) = P(u(k)).
% [v,d] = pchiptx(...) also returns d(k) = P'(x(k)).
% See PCHIP, SPLINETX.
% First derivatives
```
h = diff(x);
delta = diff(y)./h;
d = pchipslopes(h,delta);

% Piecewise polynomial coefficients

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

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

k = ones(size(u));
for j = 2:n-1
    k(u >= x(j)) = 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 knots and a one-sided formula at the end points.

function d = pchipslopes(h,delta);
% PCHIPSLOPES Derivative for hermite cubic interpolation.
% pchipslopes(h,delta) computes d(k) = P'(x(k)).

n = length(h)+1;
if n == 2
    % Linear interpolation.
    d = zeros(size(y));
    d(1:2) = delta(1);
    return
end

% Slopes at interior points
% delta = diff(y)./diff(x).
% d(k) = 0 if delta(k-1) and delta(k) have opposites signs
% or either is zero.
% d(k) = weighted harmonic mean of delta(k-1) and delta(k)
% if they have the same sign.

d = zeros(size(delta));
\[
\begin{align*}
k &= \text{find}(\text{sign}(\text{delta}(1:n-2)).*\text{sign}(\text{delta}(2:n-1)) > 0) + 1; \\
w1 &= 2*h(k)+h(k-1); \\
w2 &= h(k)+2*h(k-1); \\
d(k) &= (w1+w2)./(w1./\text{delta}(k-1) + w2./\text{delta}(k));
\end{align*}
\]

% Slopes at end points
\[
\begin{align*}
d(1) &= \text{pchipendpoint}(h(1),h(2),\text{delta}(1),\text{delta}(2)); \\
d(n) &= \text{pchipendpoint}(h(n-1),h(n-2),\text{delta}(n-1),\text{delta}(n-2));
\end{align*}
\]

function d = pchipendpoint(h1,h2,del1,del2)
% Non-centered, shape-preserving, three-point formula.
\[
d = ((2*h1+h2)*\text{del1} - h1*\text{del2})/(h1+h2);
\]
if sign(d) ~= sign(del1)
\[
d = 0;
\]
elseif (sign(del1) ~= sign(del2)) & (abs(d) > abs(3*del1))
\[
d = 3*\text{del1};
\]
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 Derivative for cubic spline interpolation.
% splineslopes(h,delta) computes d(k) = S'(x(k)).

% Set up sparse tridiagonal system and right hand side.
\[
\begin{align*}
\text{rows} &= (\text{size}(h,2) > \text{size}(h,1)); \\
h &= h(:); \\
delta &= \text{delta}(:); \\
n &= \text{length}(h)+1; \\
k &= 2:n-1; \\
h1 &= [h(k); \text{NaN}; 0]; \\
hc &= [h(2); 2*(h(k-1)+h(k)); h(n-2)]; \\
hr &= [0; \text{NaN}; h(k-1)]; \\
A &= \text{spdiags}([h1 hc hr], -1:1, n, n); \\
r &= [\text{NaN}; 3*(h(k).*\text{delta}(k-1)+h(k-1).*\text{delta}(k)); \text{NaN}];
\end{align*}
\]

% Not-a-knot end conditions
\[
\begin{align*}
A(1,2) &= h(2)+h(1); \\
A(n,n-1) &= h(n-1)+h(n-2); \\
r(1) &= ((h(1)+2*A(1,2))*h(2)*\text{delta}(1)+ \ldots \\
&\quad h(1)^2*\text{delta}(2))/A(1,2); \\
r(n) &= (h(n-1)^2*\text{delta}(n-2)+ \ldots \\
&\quad (2*A(n,n-1)+h(n-2)*\text{delta}(n-1))/A(n,n-1); \\
\end{align*}
\]
% Solve tridiagonal linear system

d = A\r;
if rows, d = d'; end

5 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, interp(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.

Here is the initial plot generated by our example data.
Interpolation

After initialization, the interpolation points can be varied with the mouse. If x has been specified, it remains fixed.

Exercises

1. 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.

<table>
<thead>
<tr>
<th>%</th>
<th>Date</th>
<th>Tom</th>
<th>Ben</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>27 2001</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>19 2001</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>03 2001</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>20 2001</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>01</td>
<td>09 2002</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>01</td>
<td>23 2002</td>
<td>14</td>
<td>8</td>
</tr>
</tbody>
</table>

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

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

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 this.

![Twin's weights](image)

2. Let’s make a plot of your hand. Start with

```matlab
figure
set(gcf, 'menubar', 'none')
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 may 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.)

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

```matlab
n = length(x);
s = (1:n)';
t = (1:.05:n)';
```

18
\[ u = \text{splinetx}(s,x,t); \]
\[ v = \text{splinetx}(s,y,t); \]
\[ \text{clf reset} \]
\[ \text{plot}(x,y,'.',u,v,'-'); \]

Do the same thing with \text{pchipxt}. Which do you prefer?

Here is the plot of my hand. Can you tell if it is done with \text{splinetx} or \text{pchipxt}?

![Hand Plot](image.png)

3. Answer the questions in the preamble for \text{vandal.m}.

```matlab
function d = vandal(n)
% VANDAL What does this program do?
% Exercise:
% Try VANDAL(n) for n <= 8.
% What theorem does VANDAL demonstrate?
% What goes wrong when n = 9?
% How can the difficulty be avoided?

x = sym(zeros(n,1));
for i = 1:n
    x(i) = sym(char('a'+i-1));
end
```

19
\begin{verbatim}
V = sym(ones(n,1));
for j = 2:n
    V(:,j) = x.*V(:,j-1);
end
pretty(V)
d = 1;
for k = 1:n-1
    for i = k+1:n
        V(i,:) = V(i,:) - V(k,:);
        p = V(i,k+1);
        d = d*p;
        V(i,:) = simplify(V(i,:)/p);
    end
end
\end{verbatim}

4. 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.

5. 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) Monomial basis.
\[ T_5(x) = 16x^5 - 20x^3 + 5x \]

(b) Relation to trigonometric functions.
\[ T_5(x) = \cos (5 \cos^{-1} x) \]

(c) Horner representation.
\[ T_5(x) = (((((16x + 0)x - 20)x + 0)x + 5)x + 0 \]

(d) Lagrange interpolating polynomial. Let
\[
\begin{align*}
x_1, x_6 &= \pm 1 \\
x_2, x_5 &= \pm \phi/2 \\
x_3, x_4 &= \pm (\phi - 1)/2
\end{align*}
\]
and

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

Then

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

(e) Factored representation. Let

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

Then

\[
T_5(x) = 16 \prod_{1}^{5} (x - z_k)
\]

(f) Three term recurrence.

\[
T_0(x) = 1
\]

\[
T_1(x) = x
\]

For \( n = 2, \ldots 5 \)

\[
T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)
\]

6. The M-file \texttt{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 \(-1 \leq x \leq 1\). Runge asked, as \( n \) increases, does \( P_n(x) \) converge to \( f(x) \). The answer is, for some \( x \), but not 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?
7. Answer the questions in the preamble for bsplines.m.

```matlab
function bsplines(s)
    BSPLINES Find "B-splines". "B-splines" are Basis Splines, a set of splines that provide a convenient representation for all splines. With the simple end conditions we have provided in "splinetx", it is not possible to find a complete set of B-splines, but we can get close.

    Your job is determine a value of the input parameter "s" so that most of the splines plotted are positive and do not produce red asterisks flagging negative values.

    What is the best value of s? For this s, what values of k produce red asterisks?
```

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

9. Modify splinetx and pchiptx so that, when called with only two input arguments, they produce PP, the piecewise polynomial structure produced by spline and pchip and used by ppval.

10. Monitor cond(A) for the matrix A involved in splinetx. What happens when two of the knots approach each other? Find a data set that makes cond(A) large.

11. Reproduce the summary figure, with four subplots showing the four interpolants discussed in this chapter.

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

13. (a) Consider

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

   Which, if any, of the four plots - linear, spline, pchip and polynomial - are superimposed? Why?

   (b) Same questions for

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

14. Why does interpgui(4) show only three graphs, not four, no matter where you move the points.
15. Use the Symbolic Toolbox to study the key properties of \( P(x) \), the function used in the section 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 knots, \( x_k \). You might start with something like

\[
\text{syms } x \ s \ h \ y1 \ y0 \ d1 \ d0 \\
P = (3*h*s^2-2*s^3)/h^3*y1 + (h^3-3*h*s^2+2*s^3)/h^3*y0 \\
\quad + s^2*(s-h)/h^2*d1 + s*(s-h)^2/h^2*d0
\]

and then look at expressions such as

\[
\text{subs}(P,s,0) \\
\text{subs}(\text{diff}(P,s,2),s,h)
\]

The symbolic functions \texttt{simplify} and \texttt{pretty} might come in handy.

16. Create modified versions of \texttt{pchip} and \texttt{spline} where the one-sided and not-a-knot end conditions are replaced by 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 periodic boundary conditions can be derived by imagining that there are extra knots \( x_0 = x_1 - h_{n-1} \) and \( x_{n+1} = x_n + h_1 \) where the function takes on values \( y_0 = y_{n-1} \) and \( y_{n+1} = y_2 \). With both \texttt{pchip} and \texttt{spline}, this leads to equations for the slopes \( d_1 \) and \( d_n \) that have the same form as the equations for the other \( d_k \)’s. The fictitious extra knots do not actually enter the calculation. The special case code for the end conditions can be eliminated and the resulting M-files are actually much simpler.

Demonstrate that your new functions work correctly on

\[
\begin{align*}
x & = [0 \ 2*\text{sort}(\text{rand}(1,6))*\pi \ 2*\pi]; \\
y & = \cos(x + 2*\text{rand}*\pi); \\
u & = 0:pi/50:2*\pi; \\
v & = \text{your\_function}(x,y,u); \\
\text{plot}(x,y,\prime'o',u,v,\prime'-')
\end{align*}
\]

17. (a) If you want to interpolate census data on the interval \( 1900 \leq t \leq 2000 \) 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
\[ t = 1900:10:2000 \]
\[ V = \text{vander}(t) \]

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 \( \mu \) and \( \sigma \) lead to a reasonably well conditioned Vandermonde matrix? One possibility is

\[ \mu = \text{mean}(t) \]
\[ \sigma = \text{std}(t) \]

but are there better values?