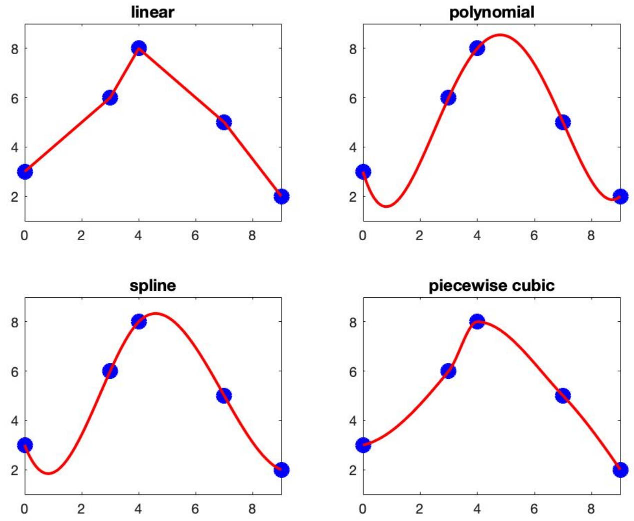
Ch. 3: Interpolation

We consider the problem of interpolating a function through a set of points:

$$(x_1, y_1),\ (x_2, y_2),\ \ldots, (x_n, y_n)$$

That is, we want to find a function $p(x)$ such that

$$p(x_i) = y_i$$



Polynomial Interpolation

First off let's consider polynomials. The *Lagrange form* of the interpolating polynomial:

$$p(x) = \sum_{k=1}^{n} \left(\frac{\prod_{i \neq k}(x - x_i)}{\prod_{i \neq k}(x_k - x_i)} \right) y_k$$

The Matlab command `polyinterp` is based on this form. The syntax is

```
polyinterp(x,y,u)
```

where `x` has the $x-$coordinates, `y` has the $y-$coordinates, and `u` is the vector of $x-$values at which to evaluate the polynomial.

Code to produce the graph on the previous slide:

```
x = [0 3 4 7 9];
y = [3 6 8 5 2];
xx = 0:.01:9;
plot(xx, polyinterp(x,y,xx))
```

Another approach to finding the interpolating polynomial is to solve a system for the coefficients. If

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

(This is the convention used in Matlab.) Then the system

$$p(x_i) = y_i, \quad i = 1, \ldots, n$$

can be written in matrix-vector form as

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

The matrix above is called a *Vandermonde matrix*. As long as the x_i's are distinct, this matrix is invertible (exercise), and hence the system can, in principle, be solved for the coefficients. However, if the number of points is large (i.e. around 10 or more) the matrix turns out to be badly conditioned (exercise). There are other problems with interpolating with high degree polynomials as well.

Piecewise Linear Interpolation

For x in

$$x_k \leq x \leq x_{k+1}$$

the linear function interpolating the points (x_k, y_k) and (x_{k+1}, y_{k+1}) is easily found using the point-slope form:

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

Alternatively we can write in terms of the *local variable*

$$s = x - x_k$$

and slope or *first divided difference*

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

Then

$$L(x) = y_k + s\delta_k$$

Piecewise Cubic Interpolation

Most of the interpolation schemes used in practice are piecewise cubic. As before we use the local variable and first divided difference:

$$h_k = x_{k+1} - x_k, \quad s = x - x_k, \quad \delta_k = \frac{y_{k+1} - y_k}{h_k}$$

We also call d_k the slope of the interpolant at x_k:

$$d_k = P'(x_k)$$

Then we can write

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

Note that P is piecewise cubic and satisfies

$$P(x_k) = y_k, \qquad P(x_{k+1}) = y_{k+1}$$

$$P'(x_k) = d_k, \qquad P'(x_{k+1}) = d_{k+1}$$

It “only” remains to determine the slopes d_k.

Shape Preserving Piecewise Cubic

The "piecewise cubic Hermite interpolating polynomial", or `pchip` is defined by determining the slopes

$$d_k = P'(x_k)$$

in the following way:

(1) If δ_k and δ_{k+1} have opposite signs (adjacent points are either both above or below the point (x_k, y_k)), we set

$$d_k = 0$$

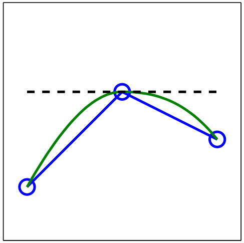
(2) If δ_k and δ_{k+1} have the same sign, d_k is the weighted harmonic mean of the discrete slopes. If the intervals on either side are equal:

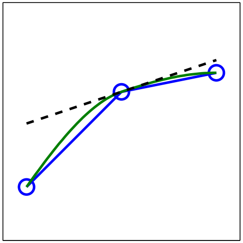
$$\frac{1}{d_k} = \frac{1}{2}\left(\frac{1}{\delta_{k-1}} + \frac{1}{\delta_k}\right)$$

If not:

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

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





Ch. 3: Interpolation

In Matlab:

```
pchip(x,y,u)
```

Notice that with pchip we define the slopes at each point. In this way the function $P(x)$ and its first derivative $P'(x)$ are continuous at each point (x_k, y_k). However, the second derivative $P''(x)$ is generally **not** going to be continuous.

Another way to define a cubic interpolant is to require that it's first and second derivatives are continuous at each point (x_k, y_k).

Cubic Splines

Cubic splines are defined to be piecewise cubic functions such that $P(x)$, $P'(x)$ and $P''(x)$ are continuous at each interpolated point (x_k, y_k) (referred to as a “knot”).

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

We require

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

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

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

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

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

This leads to the condition

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

This is a tridiagonal system for the slopes $d_k, k = 1, \ldots, n$.

For the endpoints additional conditions must be imposed. One popular choice is to use a single cubic through the three points at the left end (and right end).

Ch. 3: Interpolation

We can write the equations for the slopes as

$$Ad = r$$

where

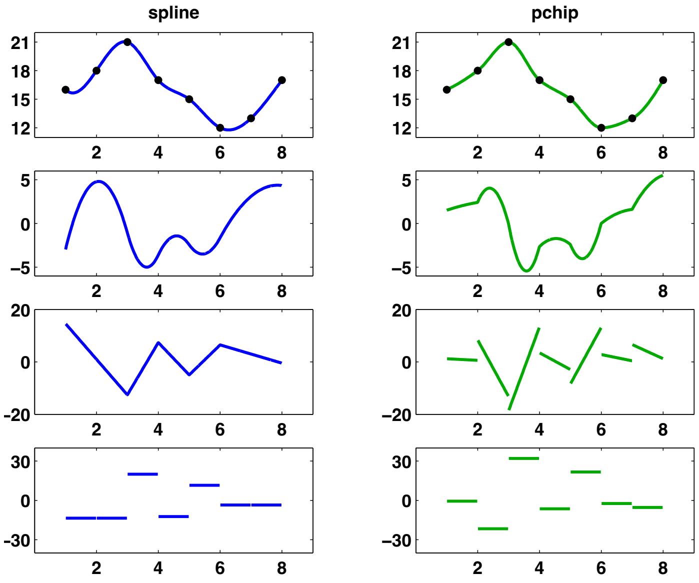
$$A = \begin{pmatrix} h_2 & h_2 + h_1 & & & & \\ h_2 & 2(h_1 + h_2) & h_1 & & & \\ & h_3 & 2(h_2 + h_3) & h_2 & & \\ & & \ddots & \ddots & & \\ & & & h_{n-1} & 2(h_{n-2} + h_{n-2}) & h_{n-2} \\ & & & & h_{n-1} + h_{n-2} & h_{n-2} \end{pmatrix}$$

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

If the points are all equally spaced, $h_k = h$,

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

Ch. 3: Interpolation


spline
pchip