spline

Cubic spline data interpolation

Syntax

```
s = spline(x,y,xq)
pp = spline(x,y)
```

Description

`s = spline(x,y,xq)` returns a vector of interpolated values `s` corresponding to the query points in `xq`. The values of `s` are determined by cubic spline interpolation of `x` and `y`. example

`pp = spline(x,y)` returns a piecewise polynomial structure for use by `ppval` and the spline utility `unmkpp`. example

Examples

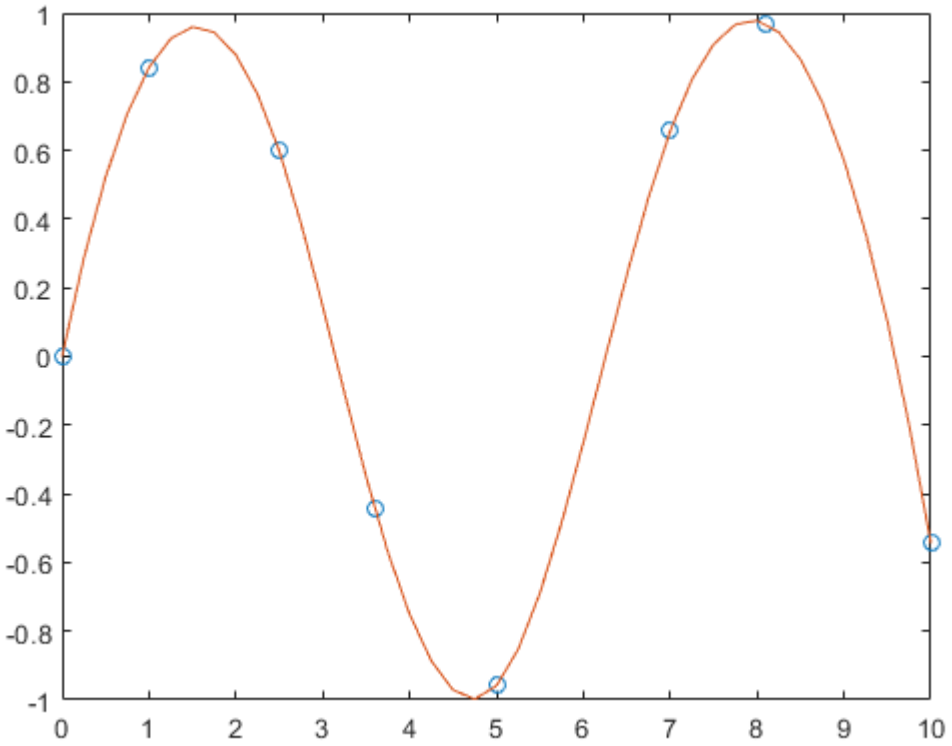
collapse all

Spline Interpolation of Sine Data

Use `spline` to interpolate a sine curve over unevenly-spaced sample points.

Try it in MATLAB

```
x = [0 1 2.5 3.6 5 7 8.1 10];
y = sin(x);
xx = 0:.25:10;
yy = spline(x,y,xx);
plot(x,y,'o',xx,yy)
```

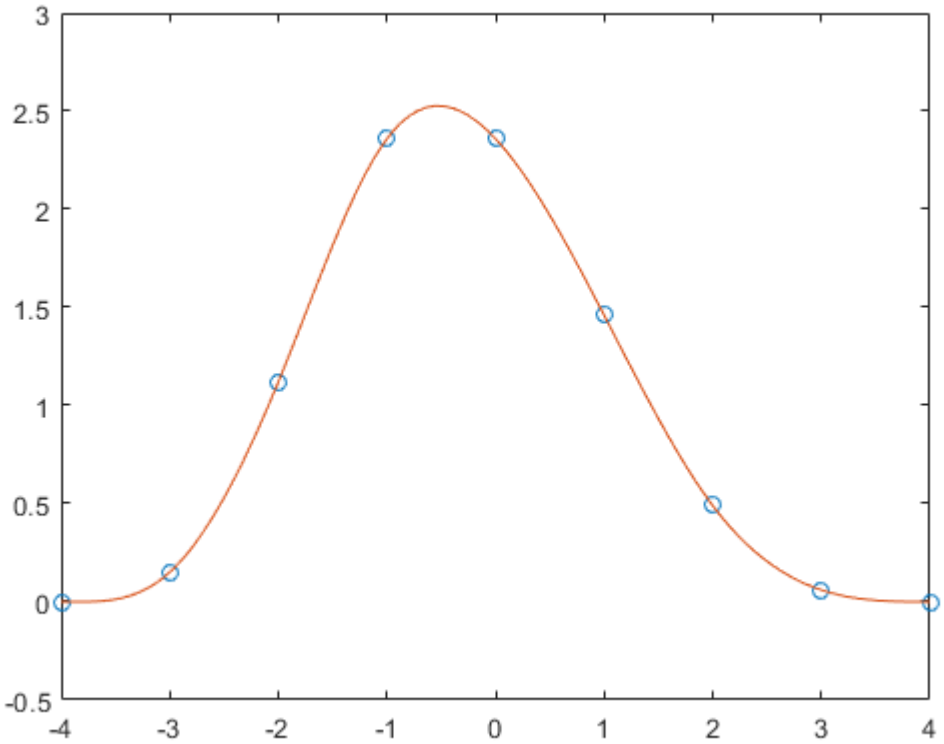


Spline Interpolation of Distribution with Specified Endpoint Slopes

Use clamped or complete spline interpolation when endpoint slopes are known. This example enforces zero slopes at the end points of the interpolation.

Try it in MATLAB

```
x = -4:4;
y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
cs = spline(x,[0 y 0]);
xx = linspace(-4,4,101);
plot(x,y,'o',xx,ppval(cs,xx),'-');
```



Extrapolation Using Cubic Spline

Extrapolate a data set to predict population growth.

Try it in MATLAB

Create two vectors to represent the census years from 1900 to 1990 (`t`) and the corresponding United States population in millions of people (`p`).

```
t = 1900:10:1990;
p = [ 75.995  91.972  105.711  123.203  131.669 ...
     150.697 179.323  203.212  226.505  249.633 ];
```

Extrapolate and predict the population in the year 2000 using a cubic spline.

```
spline(t,p,2000)
```

```
ans = 270.6060
```

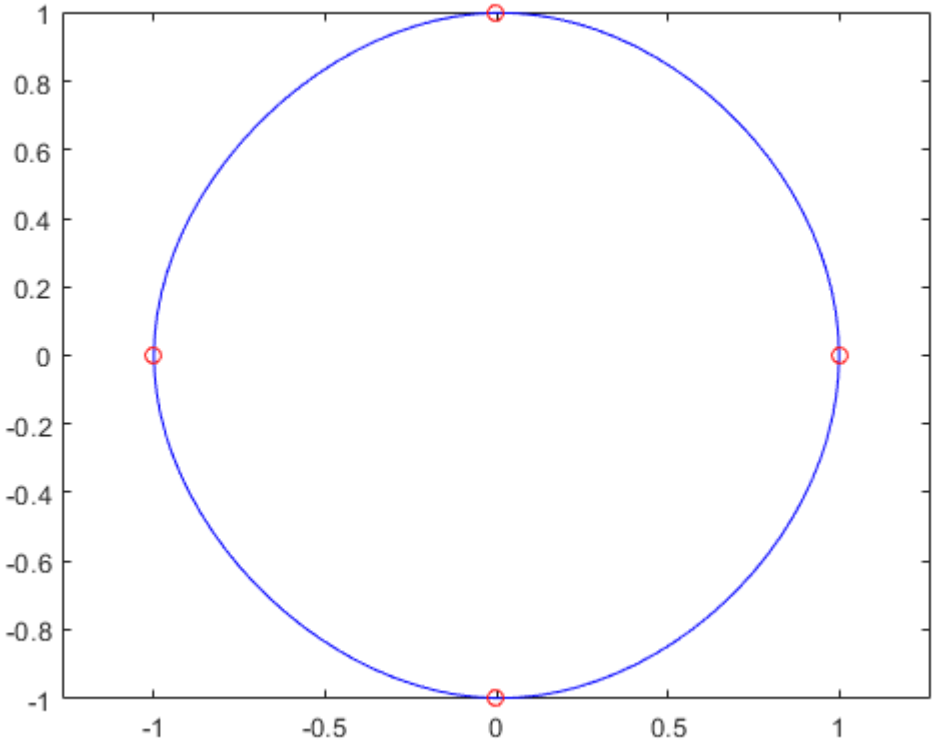
Spline Interpolation of Angular Data

Generate the plot of a circle, with the five data points `y(:,2),...,y(:,6)` marked with o's. The matrix `y` contains two more columns than does `x`. Therefore, `spline` uses `y(:,1)` and `y(:,end)` as the endslopes. The circle starts and ends at the point (1,0), so that point is plotted twice.

Try it in MATLAB

```
x = pi*[0:.5:2];
y = [0  1  0 -1  0  1  0;
```

```
     1  0  1  0 -1  0  1];
pp = spline(x,y);
yy = ppval(pp, linspace(0,2*pi,101));
plot(yy(1,:),yy(2,:),'-b',y(1,2:5),y(2,2:5),'or')
axis equal
```



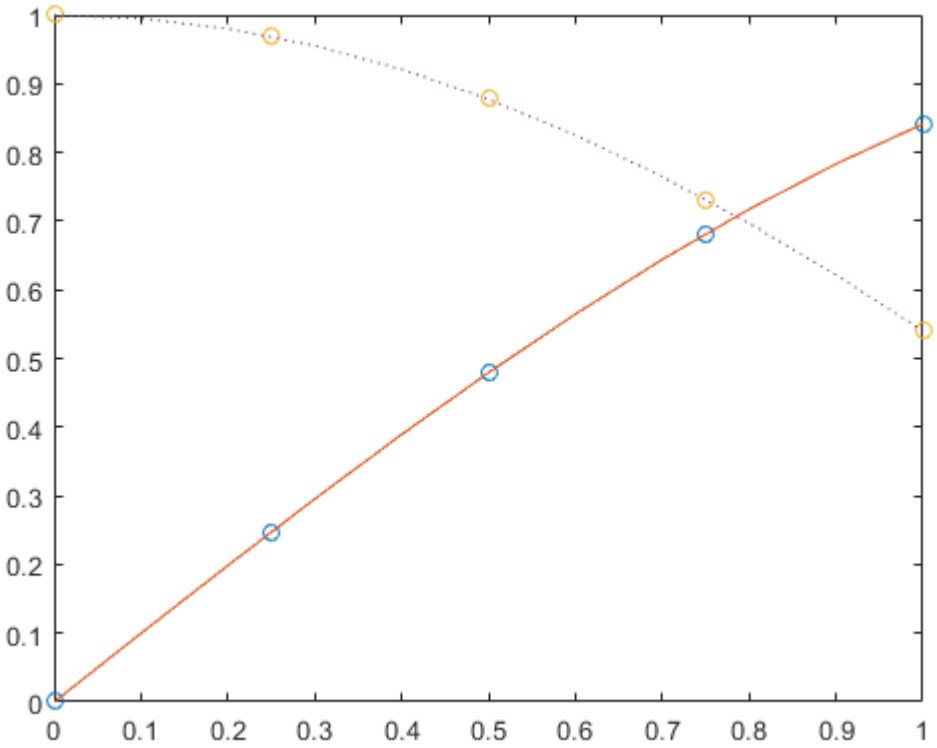
Spline Interpolation of Sine and Cosine Data

Use spline to sample a function over a finer mesh.

Try it in MATLAB

Generate sine and cosine curves for a few values between 0 and 1. Use spline interpolation to sample the functions over a finer mesh.

```
x = 0:.25:1;
Y = [sin(x); cos(x)];
xx = 0:.1:1;
YY = spline(x,Y,xx);
plot(x,Y(1,:),'o',xx,YY(1,:),'-')
hold on
plot(x,Y(2,:),'o',xx,YY(2,:),':')
hold off
```



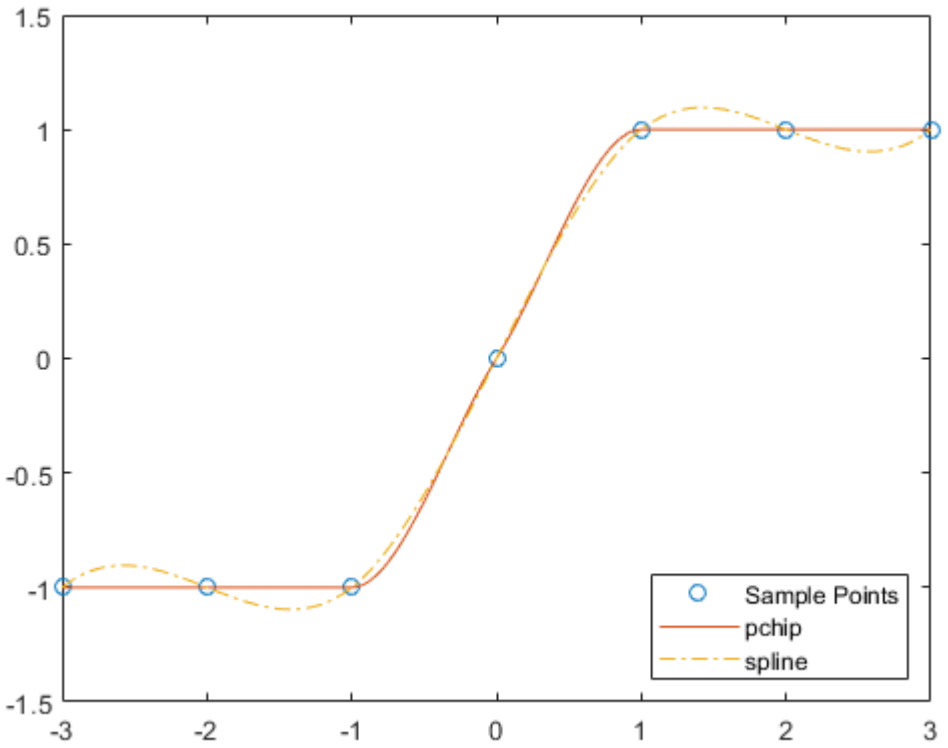
Data Interpolation Using `spline` and `pchip`

Compare the interpolation results produced by `spline` and `pchip` for two different functions.

Try it in MATLAB

Create vectors of `x` values, function values at those points `y`, and query points `xq`. Compute interpolations at the query points using both `spline` and `pchip`. Plot the interpolated function values at the query points for comparison.

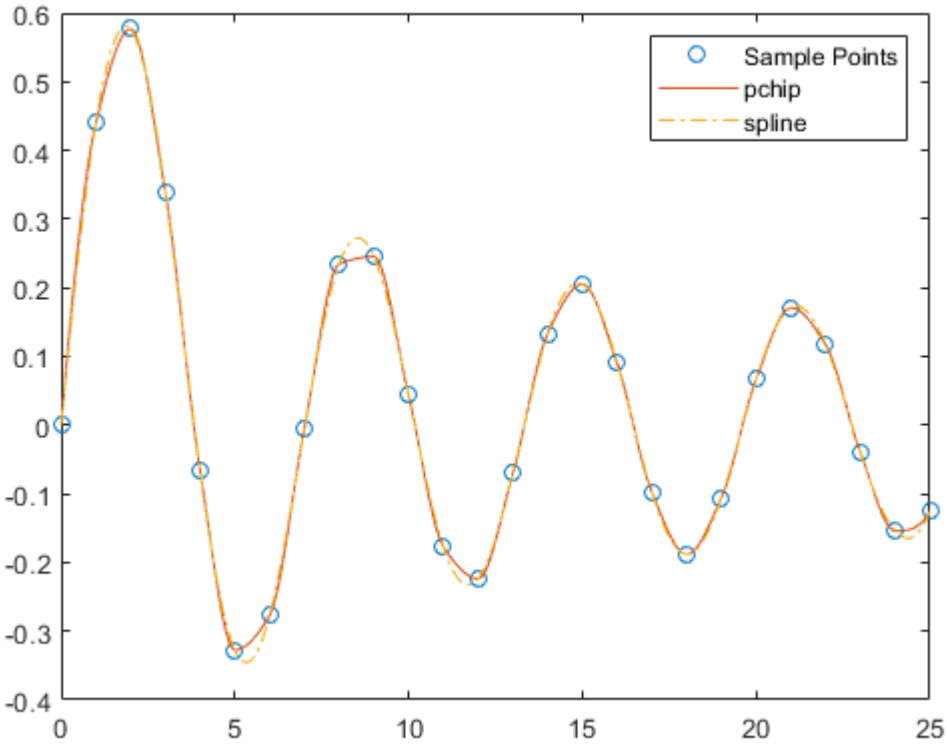
```
x = -3:3;
y = [-1 -1 -1 0 1 1 1];
xq1 = -3:.01:3;
p = pchip(x,y,xq1);
s = spline(x,y,xq1);
plot(x,y,'o',xq1,p,'-',xq1,s,'-.')
legend('Sample Points','pchip','spline','Location','SouthEast')
```



In this case, `pchip` is favorable since it does not oscillate as freely between the sample points.

Perform a second comparison using an oscillatory sample function.

```
x = 0:25;
y = besselj(1,x);
xq2 = 0:0.01:25;
p = pchip(x,y,xq2);
s = spline(x,y,xq2);
plot(x,y,'o',xq2,p,'-',xq2,s,'-.')
legend('Sample Points','pchip','spline')
```



When the underlying function is oscillatory, `spline` captures the movement between points better than `pchip`.

Input Arguments

collapse all

x — x-coordinates

vector

x-coordinates, specified as a vector. The vector `x` specifies the points at which the data `y` is given. The elements of `x` must be unique.

Data Types: `single` | `double`

y — Function values at x-coordinates

vector | matrix | array

Function values at x-coordinates, specified as a numeric vector, matrix, or array. `x` and `y` typically have the same length, but `y` also can have exactly two more elements than `x` to specify endslopes.

If `y` is a matrix or array, then the values in the last dimension, `y(:,...,:,j)`, are taken as the values to match with `x`. In that case, the last dimension of `y` must be the same length as `x` or have exactly two more elements.

The endslopes of the cubic spline follow these rules:

- If `x` and `y` are vectors of the same size, then the not-a-knot end conditions are used.
- If `x` or `y` is a scalar, then it is expanded to have the same length as the other and the not-a-knot end conditions are used.

- If `y` is a vector that contains two more values than `x` has entries, then `spline` uses the first and last values in `y` as the endslopes for the cubic spline. For example, if `y` is a vector, then:
  - `y(2:end-1)` gives the function values at each point in `x`
  - `y(1)` gives the slope at the beginning of the interval located at `min(x)`
  - `y(end)` gives the slope at the end of the interval located at `max(x)`
- Similarly, if `y` is a matrix or an `N`-dimensional array with `size(y,N)` equal to `length(x)+2`, then:
  - `y(:,...,:,j+1)` gives the function values at each point in `x` for `j = 1:length(x)`
  - `y(:,:,...:,1)` gives the slopes at the beginning of the intervals located at `min(x)`
  - `y(:,:,...:,end)` gives the slopes at the end of the intervals located at `max(x)`

Data Types: `single` | `double`

xq — Query points
vector

Query points, specified as a vector. The points specified in `xq` are the x-coordinates for the interpolated function values `s` that `spline` computes.

Data Types: `single` | `double`

Output Arguments

collapse all

s — Interpolated values at query points
vector | matrix | array

Interpolated values at query points, returned as a vector, matrix, or array.

The size of `s` is related to the sizes of `y` and `xq`:

- If `y` is a vector, then `s` has the same size as `xq`.
- If `y` is an array of size `Ny = size(y)`, then these conditions apply:
  - If `xq` is a scalar or vector, then `size(s)` returns `[Ny(1:end-1) length(xq)]`.
  - If `xq` is an array, then `size(s)` returns `[Ny(1:end-1) size(xq)]`.

pp — Piecewise polynomial
structure

Piecewise polynomial, returned as a structure. Use this structure with the `ppval` function to evaluate the piecewise polynomial at one or more query points. The structure has these fields.

Field	Description
`form`	`'pp'` for *piecewise polynomial*
`breaks`	Vector of length `L+1` with strictly increasing elements that represent the start and end of each of `L` intervals

Field	Description
`coefs`	L-by-k matrix with each row `coefs(i,:)` containing the local coefficients of an order k polynomial on the ith interval, `[breaks(i),breaks(i+1)]`
`pieces`	Number of pieces, L
`order`	Order of the polynomials
`dim`	Dimensionality of target

Since the polynomial coefficients in `coefs` are local coefficients for each interval, you must subtract the lower endpoint of the corresponding knot interval to use the coefficients in a conventional polynomial equation. In other words, for the coefficients `[a,b,c,d]` on the interval `[x1,x2]`, the corresponding polynomial is

$$f(x) = a(x - x_1)^3 + b(x - x_1)^2 + c(x - x_1) + d\,.$$

Tips

- You also can perform spline interpolation using the `interp1` function with the command `interp1(x,y,xq,'spline')`. While `spline` performs interpolation on rows of an input matrix, `interp1` performs interpolation on columns of an input matrix.

Algorithms

A tridiagonal linear system (possibly with several right-hand sides) is solved for the information needed to describe the coefficients of the various cubic polynomials that make up the interpolating spline. `spline` uses the functions `ppval`, `mkpp`, and `unmkpp`. These routines form a small suite of functions for working with piecewise polynomials. For access to more advanced features, see `interp1` or the Curve Fitting Toolbox™ spline functions.

References

[1] de Boor, Carl. A Practical Guide to Splines. Springer-Verlag, New York: 1978.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`interp1` | `mkpp` | `pchip` | `ppval` | `unmkpp`

Introduced before R2006a