Interpolating Gridded Data

Gridded Data Representation

Use ‘meshgrid’ to create a 2-D axis aligned grid from two vectors, xv and yv.

xv = 0:3;

yv = 1:5;

[X,Y] = meshgrid(xv, yv)

X =

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

0 1 2 3

Y =

1 1 1 1

2 2 2 2

3 3 3 3

4 4 4 4

5 5 5 5

Use ``ndgrid” to create a 2-D space aligned grid from the same two vectors, xv and yv.

[X1,X2] = ndgrid(xv,yv)

X1 =

0 0 0 0 0

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

X2 =

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

1 2 3 4 5

**Remarks**

The ndgrid function is like meshgrid except that the order of the first two input arguments are switched. That is, the statement

[X1,X2,X3] = ndgrid(x1,x2,x3)

produces the same result as

[X2,X1,X3] = meshgrid(x2,x1,x3)

Because of this, ndgrid is better suited to multidimensional problems that aren't spatially based, while meshgrid is better suited to problems in two- or three-dimensional Cartesian space.

[X1,X2] = ndgrid([2 4 6 3 1])

X1 =

2 2 2 2 2

4 4 4 4 4

6 6 6 6 6

3 3 3 3 3

1 1 1 1 1

X2 =

2 4 6 3 1

2 4 6 3 1

2 4 6 3 1

2 4 6 3 1

2 4 6 3 1

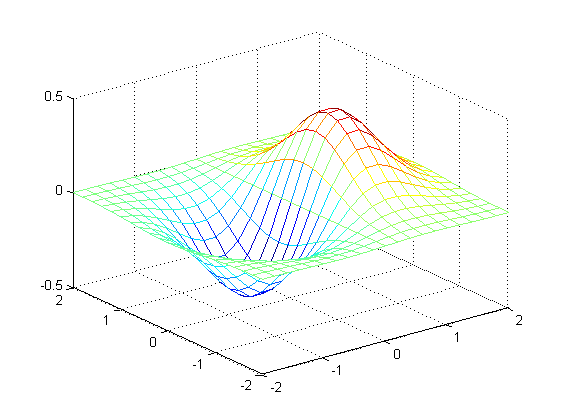
**Examples**

Evaluate the function over the range [-2,2]2.

[X1,X2] = ndgrid(-2:.2:2, -2:.2:2);

Z = X1 .\* exp(-X1.^2 - X2.^2);

mesh(X1,X2,Z)



**Full Grid**. A full grid is one in which the points are explicitly defined. The outputs of ndgrid and meshgrid define a full grid.

**Compact Grid**. The explicit definition of every point in a grid is expensive in terms of memory. The compact grid representation is a way to dispense with the memory overhead of a full grid. The compact grid representation stores just the grid vectors instead of the full grid.

Grid-based interpolation provides significant savings in computational overhead because the gridded structure allows MATLAB to locate a query point and its adjacent neighbors very quickly.

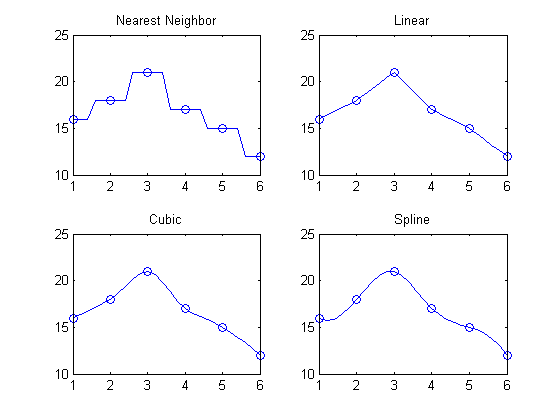
**Interpolation versus Fit**

The interpolation methods available in MATLAB create interpolating functions that pass though the sample data points. If you were to query the interpolation function at a sample location, you would get back the value at that sample data point. By contrast, curve and surface fitting algorithms do not necessarily pass through the sample data points.

**Interpolation Methods**

Grid-based interpolation offers several different methods for interpolation. When choosing an interpolation method, keep in mind that some require more memory or longer computation time than others. However, you may need to trade off these resources to achieve the desired smoothness in the result. The following table provides an overview of the benefits, trade-offs, and requirements for each method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Description | Continuity | Memory/Performance | Requirement |
| Nearest Neighbor | The interpolated value at a query point is the value at the nearest sample grid point. | Discontinous | Modest memory requirements  Fastest computation time | Requires 2 grid points in each dimension. |
| Next  Neighbor | The interpolated value at a query point is the value at the next sample grid point. | Discontinous | Same memory requirements and computation time as nearest neighbor. | * Available for 1D interpolation only. * Requires at least 2 grid points. |
| Previous Neighbor | The interpolated value at a query point is the value at the previous sample grid point. | Discontinous | Same memory requirements and computation time as nearest neighbor. | * Available for 1D interpolation only. * Requires at least 2 grid points. |
| Linear | The interpolated value at a query point is based on linear interpolation of the values at neighboring grid points in each respective dimension | C0 | * Requires more memory than nearest neighbor. * Requires more computation time than nearest neighbor. | Requires at least 2 grid points in each dimension. |
| Pchip | The interpolated value at a query point is based on a shape-preserving piece-wise cubic interpolation of the values at neighboring grid points | C1 | * Requires more memory than linear. * Requires more computation time than linear. | * Available for 1D interpolation only. * Requires at least 4 grid points. |
| Cubic | The interpolated value at a query point is based on cubic interpolation of the values at neighboring grid points in each respective dimension | C1 | Requires more memory than linear.   * Requires more computation time than linear. | * Grid must have uniform spacing, though the spacing in each dimension does not have to be the same. * Requires at least 4 points in each dimension. |
| Spline | The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension. | C2 | * Requires more memory than cubic. * Requires more computation time than cubic. | Requires 4 points in each dimension. |



**1-D Extrapolation With interp1**

This example shows how to use the 'extrap' option to interpolate beyond the domain of your sample points.

X=1:5;

Y=[12 16 31 10 6];

Xt=0:0.1:6;

Yt=interp1(X,Y,Xt,’pchip’,’extrap’);

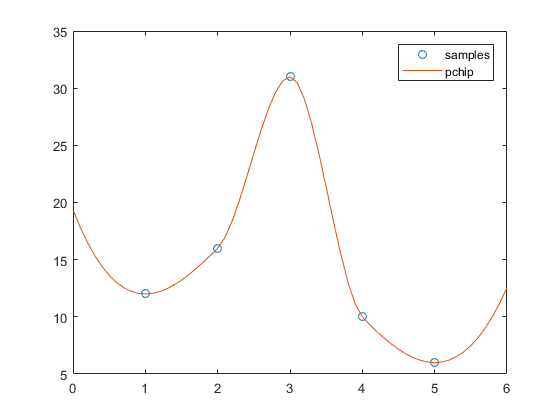
figure

plot(X,Y,’o’);

hold on

plot(Xt,Yt,’-‘);

hold off



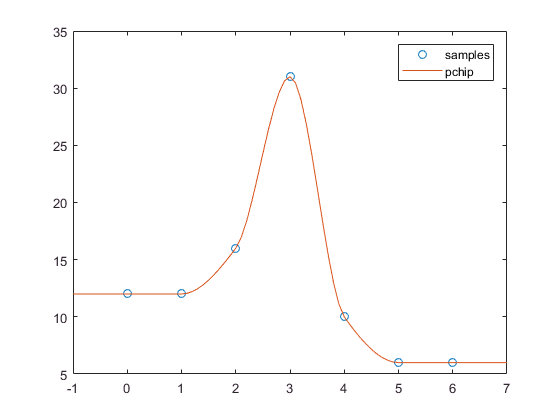
We can introduce additional points to gain more control over the behavior in the extrapolated regions. For example, you can constrain the curve to remain flat in the extrapolated region by extending the domain with repeated values.

X=0:6;

Y=[12 12 16 31 10 6 6];

Xt= -1:0.1:7;

Yt= Yt=interp1(X,Y,Xt,’pchip’);



The interp2 Function

The calling syntax for interp2 has the general form:

Vt = interp2(X,Y,V,Xt,Yt,method)

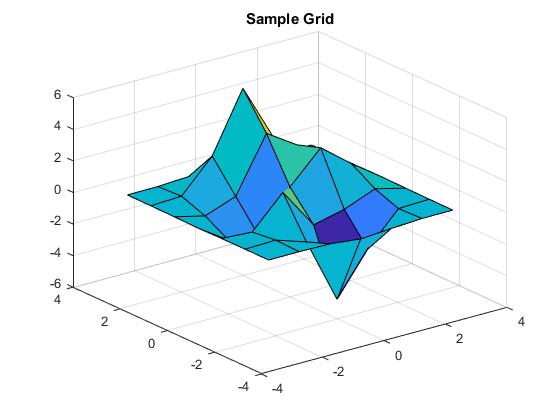
where X and Y are arrays of coordinates that define a grid in meshgrid format, and V is an array containing the values at the grid points. Xt and Yt are arrays containing the coordinates of the query points at which to interpolate. Method optionally specifies any of four interpolation methods: 'nearest', 'linear', 'cubic', or 'spline'.

The grid points that comprise X and Y must be monotonically increasing and should conform to the meshgrid format.

[X,Y]=meshgrid(-3:1:3);

V=peaks(X,Y);

surf(X,Y,V)

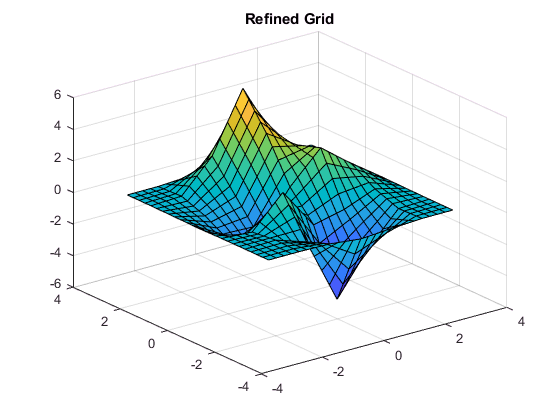


Generate a finer grid for interpolation

[Xt,Yt]=meshgrid(-3:.25:3);

Vt=interp2(X,Y,V,Xt,Yt,’linear’);

surf(Xt,Yt,Vt)



**The interp3 Function**

interp3 works in the same way as interp2 except that it takes two additional arguments: one for the third dimension in the sample grid and the other for the third dimension in the query points,

Vq = interp3(X,Y,Z,V,Xq,Yq,Zq,method).

As is the case with interp2, the grid points you supply to interp3 must be monotonically increasing and should conform to the meshgrid format.

%Define a function that generates values for X, Y, and Z input.

generatedvalues = @(X,Y,Z)(X.^2 + Y.^3 + Z.^4);

[X,Y,Z] = meshgrid((-5:.25:5)); %Create the sample data

V = generatedvalues(X,Y,Z);

Vt = interp3(X,Y,Z,V,2.35,1.76,0.23,'cubic'); %Interpolate at a point

Compare Vt to the exact value

exact= generatedvalues(2.35,1.76,0.23);

exact=10.9771

**Examples**

To generate a coarse approximation of flow and interpolate over a finer mesh:

[x,y,z,v] = flow(10);

[xi,yi,zi] = meshgrid(.1:.25:10, -3:.25:3, -3:.25:3);

vi = interp3(x,y,z,v,xi,yi,zi); % vi is 25-by-40-by-25

slice(xi,yi,zi,vi,[6 9.5],2,[-2 .2]), shading flat

