Data	Operation/Function	Interpolation Method
2D waves	**Interp2D**()	Bilinear
2D waves (points on the surface of a sphere)	**SphericalInterpolate**	Voronoi
3D waves	**Interp3D**(), **Interp3DPath**, ImageTransform **extractSurface**	Trilinear
3D scatter data	**Interpolate3D**	Barycentric

All the interpolation methods in this table consist of two common steps. The first step involves the identification of data points that are nearest to the interpolation location and the second step is the computation of the interpolated value using the neighboring values and their relative proximity. You can find the specific details in the documentation of the individual operation or function.

The Interpolate2 Operation

The **Interpolate2** operation performs linear, cubic spline and smoothing cubic spline interpolation on 1D waveform and XY data. The cubic spline interpolation is based on a routine in "Numerical Recipes in C". The smoothing spline is based on "Smoothing by Spline Functions", Christian H. Reinsch, *Numerische Mathematic* 10, 177-183 (1967).

Prior to Igor7, Interpolate2 was implemented as part of the Interpolate XOP. It is now built-in.

The Interpolate XOP also implemented an older operation named Interpolate which used slightly different syntax. If you are using the Interpolate operation, we recommend that you convert to using Interpolate2.

The main use for linear interpolation is to convert an XY pair of waves into a single wave containing Y values at evenly spaced X values so that you can use Igor operations, like **FFT**, which require evenly spaced data.

Cubic spline interpolation is most useful for putting a pleasingly smooth curve through arbitrary XY data. The resulting curve may contain features that have nothing to do with the original data so you should be wary of using the cubic spline for analytic rather than esthetic purposes.

Both linear and cubic spline interpolation are constrained to put the output curve through all of the input points and thus work best with a small number of input points. The smoothing spline does not have this constraint and thus works well with large, noisy data sets.

The Interpolate2 operation has a feature called "pre-averaging" which can be used when you have a large number of input points. Pre-averaging was added to Interpolate2 as a way to put a cubic spline through a large, noisy data set before it supported the smoothing spline. We now recommend that you use the smoothing spline instead of pre-averaging.

The Smooth Curve Through Noise example experiment illustrates spline interpolation. Choose File→Example Experiments→Feature Demos→Smooth Curve Through Noise.

Spline Interpolation Example

Before going into a complete discussion of Interpolate2, let's look at a simple example first.

First, make some sample XY data using the following commands:

```
Make/N=10 xData, yData                          // Make source data
xData = p; yData = 3 + 4*xData + gnoise(2)      // Create sample data
Display yData vs xData                          // Make a graph
Modify mode=2, lsize=3                          // Display source data as dots
```

Now, choose Analysis→Interpolate to invoke the Interpolate dialog.

Set Interpolation Type to Cubic Spline.

Choose yData from the Y Data pop-up menu.

Choose xData from the X Data pop-up menu.

Choose _auto_ from the Y Destination pop-up menu.

Choose _none_ from the X Destination pop-up menu.

Enter 200 in the Destination Points box.

Choose Off from the Pre-averaging pop-up menu.

Choose Evenly Spaced from the Dest X Coords pop-up menu.

Click Natural in the End Points section.

Notice that the dialog has generated the following command:

```
Interpolate2/T=2/N=200/E=2/Y=yData_CS xData, yData
```

This says that Interpolate2 will use yData as the Y source wave, xData as the X source wave and that it will create yData_CS as the destination wave. _CS means "cubic spline".

Click the Do It button to do the interpolation.

Now add the yData_CS destination wave to the graph using the command:

```
AppendToGraph yData_CS; Modify rgb(yData_CS)=(0,0,65535)
```

Now let's try this with a larger number of input data points. Execute:

```
Redimension/N=500 xData, yData
xData = p/50; yData = 10*sin(xData) + gnoise(1.0)     // Create sample data
Modify lsize(yData)=1                                 // Smaller dots
```

Now choose Analysis→Interpolate to invoke the Interpolate dialog. All the settings should be as you left them from the preceding exercise. Click the Do It button.

Notice that the resulting cubic spline attempts to go through all of the input data points. This is usually not what we want.

Now, choose Analysis→Interpolate again and make the following changes:

Choose Smoothing Spline from the Interpolation Type pop-up menu. Notice the command generated now references a wave named yData_SS. This will be the output wave.

Enter 1.0 for the smoothing factor. This is usually a good value to start from.

In the Standard Deviation section, click the Constant radio button and enter 1.0 as the standard deviation value. 1.0 is correct because we know that our data has noise with a standard deviation of 1.0, as a result of the "gnoise(1.0)" term above.

Click the Do It button to do the interpolation. Append the yData_SS destination wave to the graph using the command:

```
AppendToGraph yData_SS; Modify rgb(yData_SS)=(0,0,0)
```

Notice that the smoothing spline adds a pleasing curve through the large, noisy data set. If necessary, enlarge the graph window so you can see this.

You can tweak the smoothing spline using either the smoothing factor parameter or the standard deviation parameter. If you are unsure of the standard deviation of the noise, leave the smoothing factor set to 1.0 and try different standard deviation values. It is usually not too hard to find a reasonable value.