Scientific Computing in Python

Interpolation : scipy.interpolate

Interpolation is a term used in numerical analysis to describe the process of creating new data points from a set of known data points. Spline functions and classes, one-dimensional and multi-dimensional (univariate and multivariate) interpolation classes, and more can be found in the SciPy library's scipy.interpolate subpackage. Interpolation is important not only in statistics but also in science, business, and for predicting values that fall between two existent data points.

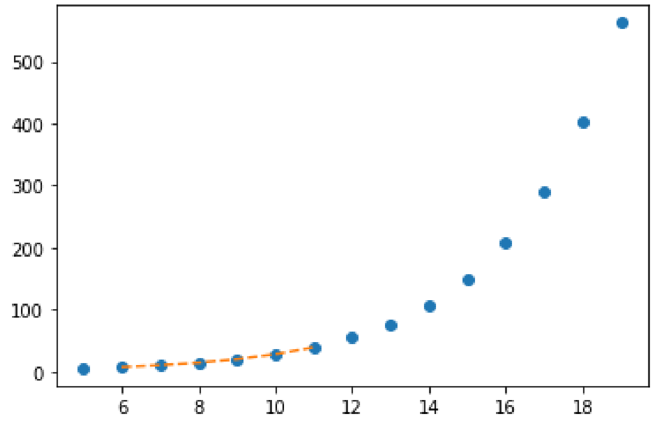
Example 1 : 1D Interpolation

The interp1d class in scipy.interpolate is a handy way to make a function out of fixed data points that may be evaluated anywhere within the domain defined by the input data using linear interpolation.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import interpolate
x = np.arange(5, 20)
y = np.exp(x/3.0)
f = interpolate.interp1d(x, y)
x1 = np.arange(6, 12)
y1 = f(x1)   # use interpolation function returned by `interp1d`
plt.plot(x, y, 'o', x1, y1, '--')
plt.show()
```

Output:

Scientific Computing in Python



Example 2 : Multivariate Interpolation

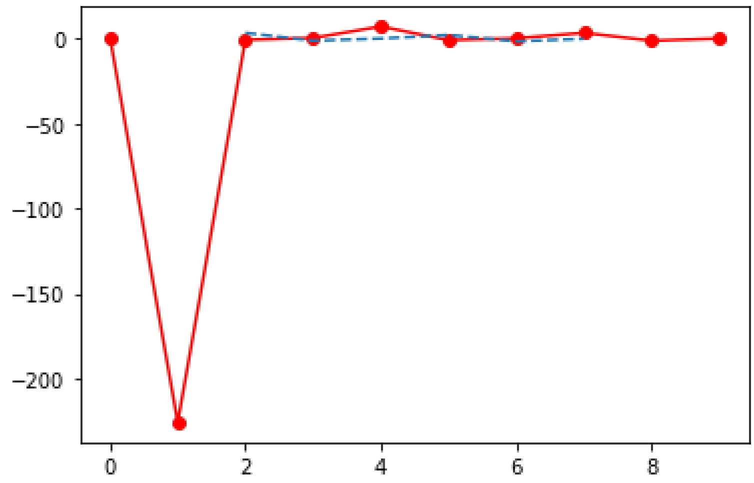
Multivariate interpolation (sometimes known as spatial interpolation) is used on functions with many variables. The interp2d function is demonstrated in the following example.

The interp2d(x, y, z) function interpolates across a 2-D grid by utilising the x, y, and z arrays to approximate some function f: "z = f(x, y)" and returns a function whose call method employs spline interpolation to discover the value of new points.

```
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.arange(0,10)
y = np.arange(10,25)
x1, y1 = np.meshgrid(x, y)
z = np.tan(x1+y1)
f = interpolate.interp2d(x, y, z, kind='cubic')
x2 = np.arange(2,8)
y2 = np.arange(15,20)
z2 = f(x2, y2)
plt.plot(x, z[0, :], 'ro-', x2, z2[0, :], '--')
plt.show()
```

Output:

Scientific Computing in Python



Example 3 : Spline Interpolation

Spline interpolation involves two steps: (1) computing a spline representation of the curve, and (2) evaluating the spline at the required points. There are two approaches to express a curve and generate (smoothing) spline coefficients in order to discover the spline representation: directly and parametrically. Using the function ***splrep***, the direct technique determines the spline representation of a curve in a 2-D plane.

The function ***splprep*** allows you to parametrically define curves in N-D space. Only one input argument is required for this function. The curve in N-D space is represented by a list of -arrays in this input. The number of curve points equals the length of each array, and each array contains one component of the N-D data point.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
x = np.arange(0, 2*np.pi+np.pi/4, 2*np.pi/8)
y = np.sin(x)
tck = interpolate.splrep(x, y, s=0)
xnew = np.arange(0, 2*np.pi, np.pi/50)
ynew = interpolate.splev(xnew, tck, der=0)
plt.figure()
plt.plot(x, y, 'x', xnew, ynew, xnew, np.sin(xnew), x, y, 'b')
plt.legend(['Linear', 'Cubic Spline', 'True'])
```

```
plt.axis([-0.05, 6.33, -1.05, 1.05])
plt.title('Cubic-spline interpolation')
plt.show()
```

Output: