Universidad de Sonora

División de Ciencias Exactas y Naturales

Física Computacional I

#3 Interpolación

Integrante:
Alejandro Guirado García

12 de febrero de 2016

1. Introducción

En está prácica interpolaremos puntos aleatorias con la función para interpolar de python. Primeramente se explicará en que consiste interpolar; En el subcampo matemático del análisis numérico, se denomina interpolación a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.

En ingeniería y algunas ciencias es frecuente disponer de un cierto número de puntos obtenidos por muestreo o a partir de un experimento y pretender construir una función que los ajuste.

Otro problema estrechamente ligado con el de la interpolación es la aproximación de una función complicada por una más simple. Si tenemos una función cuyo cálculo resulta costoso, podemos partir de un cierto número de sus valores e interpolar dichos datos construyendo una función más simple. En general, por supuesto, no obtendremos los mismos valores evaluando la función obtenida que si evaluamos la función original, si bien dependiendo de las características del problema y del método de interpolación usado la ganancia en eficiencia puede compensar el error cometido.

En todo caso, se trata de encontrar una función que:

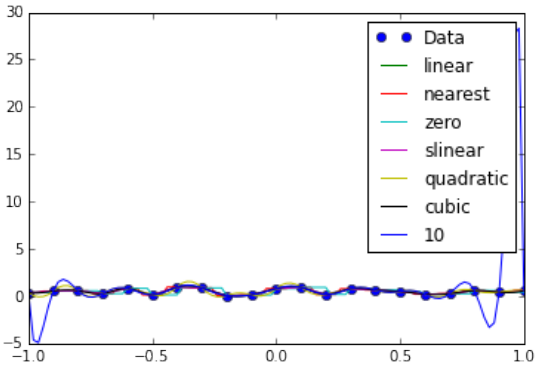
$$f(x_k) = y_k, k = 1, ..., n \tag{1}$$

Fue proporcionado un código base para modificar el cual es el siguiente:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# Original "data set" --- 21 random numbers between 0 and 1.
x0 = np.linspace(-1,1,21)
y0 = np.random.random(21)

plt.plot(x0, y0, 'o', label='Data')

# Array with points in between those of the data set for interpolation.
x = np.linspace(-1,1,101)

# Available options for interp1d
options = ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 10)

for o in options:
```

```
    f = interp1d(x0, y0, kind=o)    # interpolation function
    plt.plot(x, f(x), label=o)      # plot of interpolated data

plt.legend()
plt.show()
```

Este código nos proporcionó la siguiente interpolación:



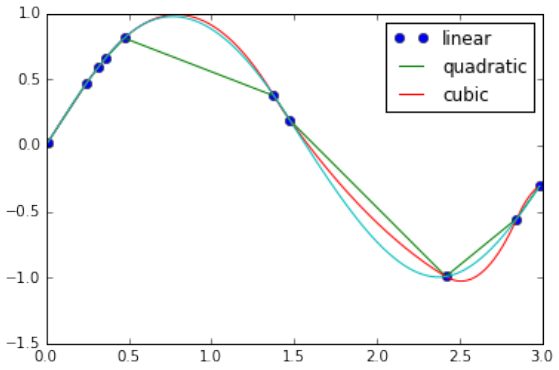
2. Interpolaciones

1. La primera interpolación es dados 10 puntos aleatorios entre x=0 y x=3, para f(x)=sin(2x).

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


x0 = np.random.random(10)
x = x0*3
y = np.sin(2*x)

plt.plot(x, y, 'o', label='Data')

xp = np.linspace(min(x), max(x), 51051)

  options = ('linear', 'quadratic', 'cubic')
for o in options:
    f = interp1d(x, y, kind=o)
```

```
    plt.plot(xp, f(xp), label=o)

plt.legend([ 'linear', 'quadratic', 'cubic'], loc='best')
plt.show()
```

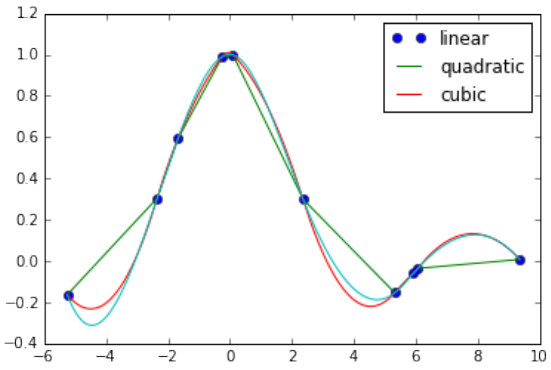
Este código nos proporcionó la siguiente interpolación:



2. La segunda interpolación es dados 20 puntos aleatorios entre x=-10 y x=10, para $f(x) = \frac{sin(x)}{x}$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


x0 = np.random.random(10)
x = (x0*20)-10
y = np.sin(x)/x

plt.plot(x, y, 'o', label='Data')

xp = np.linspace(min(x), max(x), 51051)

  options = ('linear', 'quadratic', 'cubic')
for o in options:
    f = interp1d(x, y, kind=o)
    plt.plot(xp, f(xp), label=o)

plt.legend([ 'linear', 'quadratic', 'cubic'], loc='best')
plt.show()
```

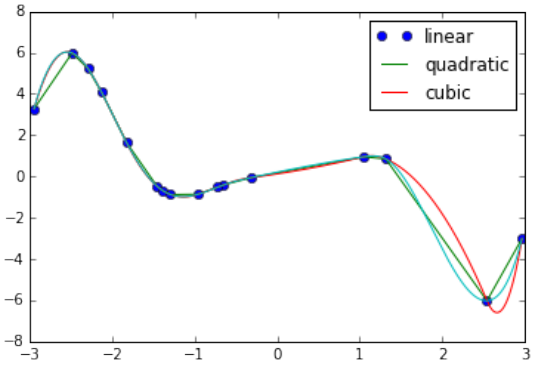
Este código nos proporcionó la siguiente interpolación:



3. La tercera interpolación es dados 20 puntos aleatorios entre x=-3 y x=3, para $f(x) = x^2 sin(2x)$.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


x0 = np.random.random(16)
x = (x0*6)-3
y = (x**2)*np.sin(2*x)


plt.plot(x, y, 'o', label='Data')


xp = np.linspace(min(x), max(x), 5100)


options = ('linear', 'quadratic', 'cubic')
for o in options:
    f = interp1d(x, y, kind=o)
    plt.plot(xp, f(xp), label=o)

plt.legend([ 'linear', 'quadratic', 'cubic'], loc='best')
plt.show()
```

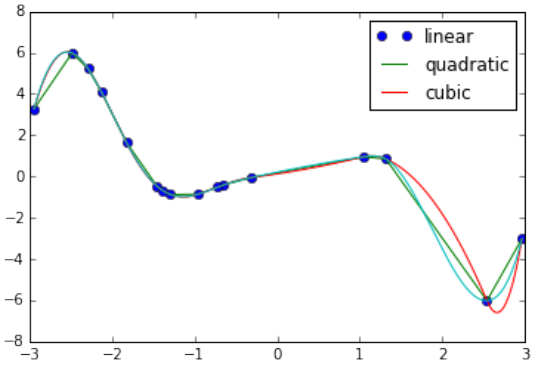
Este código nos proporcionó la siguiente interpolación:



4. La cuarta interpolación es dados 12 puntos aleatorios entre x=-2 y x=2, para $f(x) = x^s in(3x)$.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


x0 = np.random.random(12)
x = (x0*4)-2
y = (x**3)*np.sin(2*x)

plt.plot(x, y, 'o', label='Data')

xp = np.linspace(min(x), max(x), 5100)

options = ('linear', 'quadratic', 'cubic')
for o in options:
    f = interp1d(x, y, kind=o)
    plt.plot(xp, f(xp), label=o)

plt.legend([ 'linear', 'quadratic', 'cubic'], loc='best')
plt.show()
```

Este código nos proporcionó la siguiente interpolación:


linear
quadratic
cubic
8
6
4
2
0
−2
−4
−6
−8
−3
−2
−1
0
1
2
3