Actividad 3: Interpolación

Valenzuela Carrillo María Inés

29 de Enero del 2016

1. Introducción

Mediante la interpolación se pueden calcular valores intermedios entre los datos obtenidos experimentalmente y de esta manera ajustarlos a una o varias funciones que puedan representarlos. Existen diversos formas para interpolar valores medios, el método más común es la interpolación polinomial, la cual consiste en determinar el polinomio de orden n que ajusta a $n + 1$ datos. La interpolación de Lagrange, es una de las alternativas más atractivas que existe para interpolar, debido a la facilidad de programar[1].

Los métodos de interpolación pueden no ser complicados, sin embargo en ocasiones resultan muy extensos y en estos casos lo más conveniente es realizar un programa. Es por ello que esta actividad tiene como objetivo aprender a realizar interpolaciones mediante programas en Python.

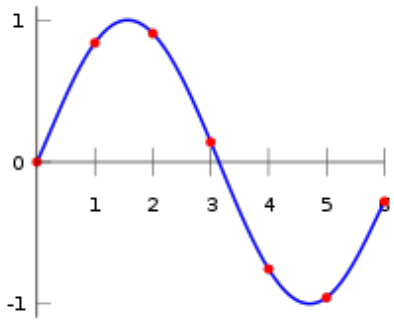


Figura 1: Interpolación polinomial [2]

2. Interpolaciones en Python

Existen distintas maneras de ajustar funciones a datos, en esta actividad se utilizará la interpolación lineal, cuadrática y cubica apoyados en la función scipy.interpolate de Python.

Se proporciona el siguiente código para realizarle modificaciones y adecuarlo a lo que se pide:

```
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()
```

2.1. Problema 1

Se pide darle a x 10 puntos aleatorios entre $x = 0$ y $x = 3$ para la función $f(x) = \sin(2x)$.

El código es:

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

# Original "data set" --- 21 random numbers between 0 and 1.
x00 = np.random.random(10)
x0=3*x00
y0 = np.sin(2*x0)

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

# Array with points in between those of the data set for interpolation.
x=np.linspace(min(x0), max(x0), 101)

# Available options for interp1d
options = ('slinear', 'quadratic', 'cubic')

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()
```

La gráfica resultante es:

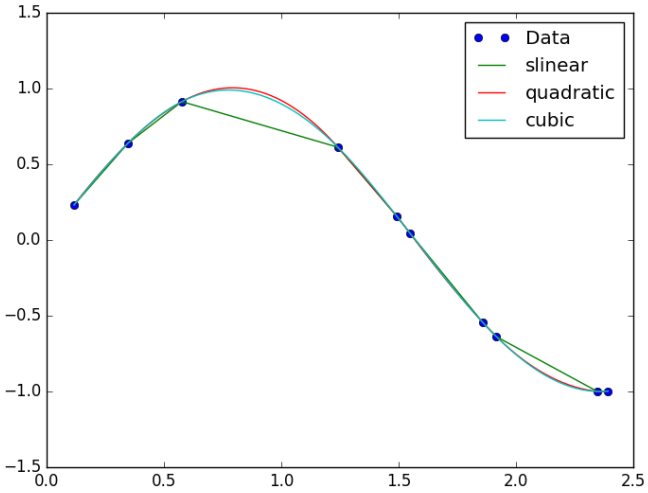
En esta gráfica de ve que la cuadrática y la cubica aproximan casi igual, mientras que la lineal solo en ciertos segmentos se parece a la curva real.

2.2. Problema 2

Se pide darle a x 20 puntos aleatorios entre $x = -10$ y $x = 10$ para la función $f(x) = \frac{\sin(2x)}{x}$.

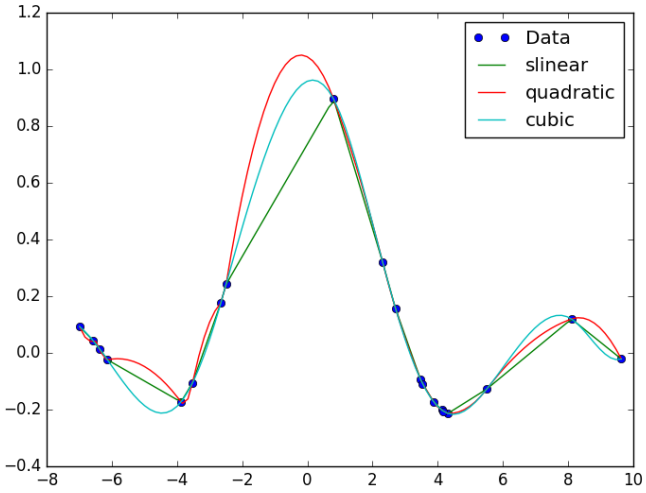
El código es:

```
import numpy as np
import matplotlib.pyplot as plt
```




```
from scipy.interpolate import interp1d

# Original "data set" --- 21 random numbers between 0 and 1.
x00 = np.random.random(20)
x0=20*x00-10
y0 = np.sin(x0)/x0

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

# Array with points in between those of the data set for interpolation.
x=np.linspace(min(x0), max(x0), 101)

# Available options for interp1d
options = ('slinear', 'quadratic', 'cubic')

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()
```

La gráfica resultante es:



Esta gráfica al contar con mayor cantidad de datos es más exacta y clara. La aproximación cuadrática y cubica no son tan parecidas como en el prolema 1.

2.3. Problema 3

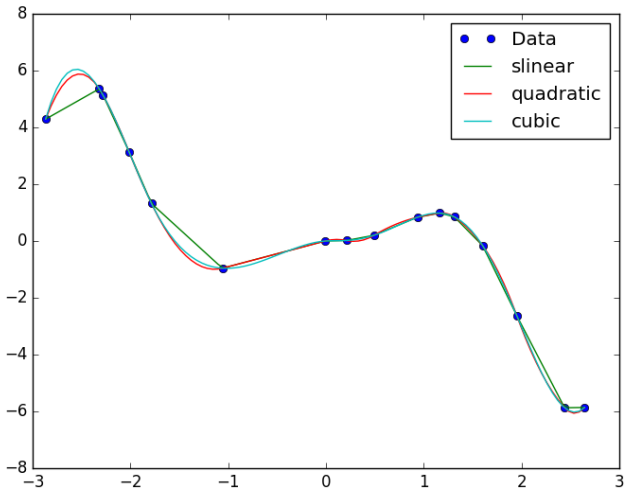
Se pide darle a x 16 puntos aleatorios entre $x = -3$ y $x = 3$ para la función $f(x) = x^2 \sin(2x)$.

El código es:

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

# Original "data set" --- 21 random numbers between 0 and 1.
x00 = np.random.random(16)
x0=6*x00-3
y0 = np.sin(2*x0)*(x0**2)
```

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

# Array with points in between those of the data set for interpolation.
x=np.linspace(min(x0), max(x0), 101)

# Available options for interp1d
options = ('slinear', 'quadratic', 'cubic')

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()
```

La gráfica resultante es:



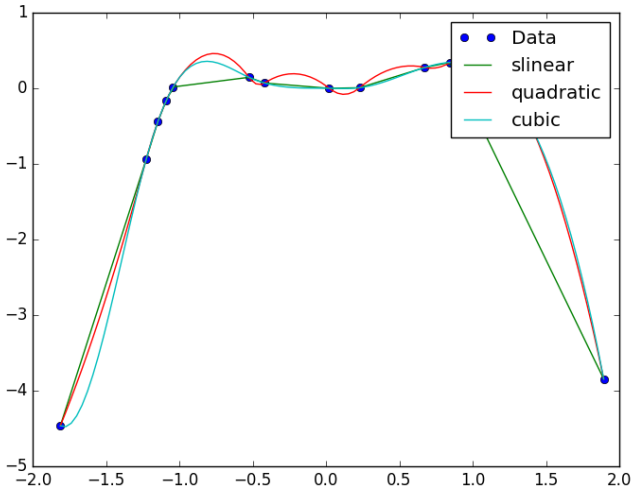
2.4. Problema 4

Se pide darle a x 12 puntos aleatorios entre $x = -2$ y $x = 2$ para la función $f(x) = x^3 \sin(3x)$.

El código es:

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

# Original "data set" --- 21 random numbers between 0 and 1.
x00 = np.random.random(12)
x0=4*x00-2
y0 = np.sin(3*x0)*(x0**3)

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

# Array with points in between those of the data set for interpolation.
x=np.linspace(min(x0), max(x0), 101)

# Available options for interp1d
options = ('slinear', 'quadratic', 'cubic')

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()
```

La gráfica resultante es:



3. Conclusión

En esta actividad de pudo ver que la programación es una herramienta muy útil al realizar interpolaciones. Mediante la función scipy.interpolate de Python se pudieron interpolar diferentes funciones con datos aleatorios de manera sencilla. También se logro observar que entre más puntos se tengan el resultado es mejor.

Lo anterior tiene gran aplicación práctica ya que para toda persona que trabaje con datos experimentales es más fácil manejar estos si conoce la manera en la que se comportan, es decir, ajustandolos a un modelo matemático.

Referencias

[1] Marco A. Cruz Chávez,
http://www.gridmorelos.uaem.mx/ mcruz//cursos/mn/interpolacion.pdf.

[2] Wikipedia, the free encyclopedia, *Interpolation.*