Funciones de interpolación de `python`

Curso de Física Computacional - Guía de apoyo

M. en C. Gustavo Contreras Mayén.

1. Funciones de interpolación con `python`.

Encontramos en `python` una serie de funciones que permiten realizar la interpolación de un conjunto de datos, estimando la mejor aproximación, pero no debemos de confiarnos en dar por hecho que con ello, el error obtenido por la aproximación es el menor.

1.1. La función `scipy.interpolate`

La librería y el módulo que debemos de utilizar es `scipy.interpolate` La función `interp1d (x, y)` La sintaxis mínima de la función de interpolación es la siguiente:

```
interp1d(x, y, kind='linear', axis=-1)
```

Donde x, y son valores en arreglos que se utilizarán para aproximar la función $f : y = f(x)$. Considera lo siguiente

- $x : (N,)$ es un arreglo de valores reales.
- $y : (..., N, ...)$ es una arreglo de valores reales. El tamaño del arreglo y sobre el eje de interpolación debe ser igual al tamaño del arreglo x.
- `kind` : es una cadena (`str`) o un entero (`int`), es un argumento opcional.

- `axis` : es un valor entero (`int`), es una argumento opcional. Específica el eje de y en el cual se va a interpolar. El valor por defecto es el eje y.

Opciones para el tipo de interpolación.

Las opciones disponibles son:

1. `linear`: interpola a lo largo de una línea recta entre puntos de datos vecinos.
2. `nearest`: proyecta al punto de datos más cercano.
3. `zero`: proyecta al punto de datos anterior.
4. `slinear`: usa un "spline" lineal.
5. `cuadratic`: usa un "spline" cuadrático.
6. `cubic`: usa un "spline" cúbico.

El valor predeterminado de `interp1d` es una interpolación lineal. También se puede proporcionar un número entero, en cuyo caso la función utilizará un polinomio de ese orden para interpolar entre los puntos proporcionados.

Por ejemplo:

```
F = interp1d (x, y, kind = 10)
```

Utilizará un polinomio de orden 10 para interpolar entre puntos.

Ejemplo

Con el siguiente código generamos un conjunto de 20 datos distribuidos entre 0 y $10 * \pi$

Código 1: Puntos conocidos

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(0, 10 * np.pi, 20)
5 y = np.cos(x)
6
7 plt.plot(x,y, 'bo', label='Datos')
8 plt.show()
```

Obtenemos la siguiente gráfica

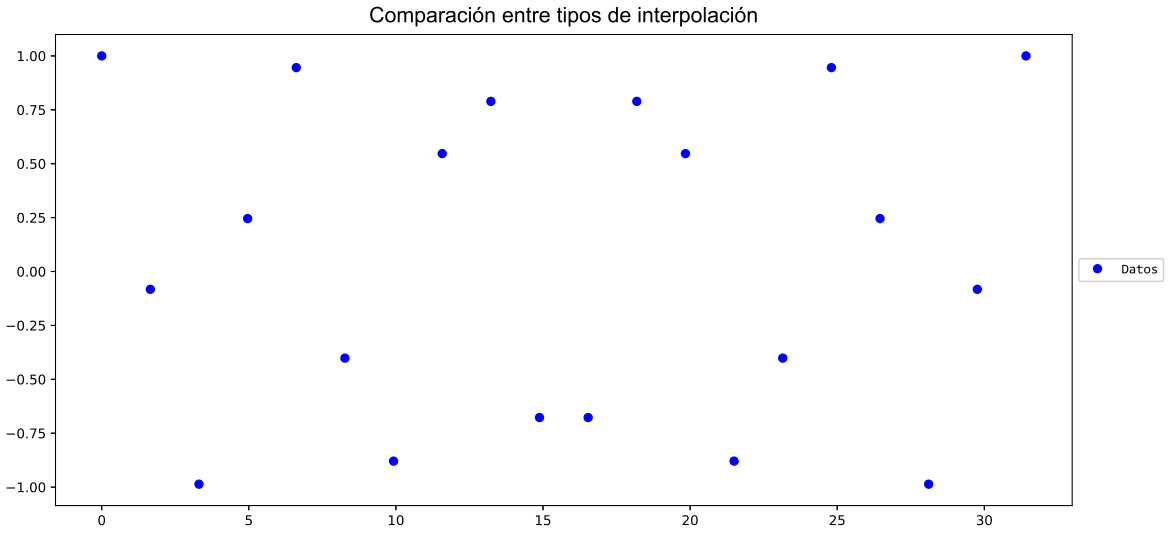

Figura 1: Datos que nos servirán para interpolar.

Ahora usaremos la función de `python` para interpolar los datos:

Código 2: Inteporlando con la función `interp1d`

```
fl = interp1d(x, y, kind='linear')

fq = interp1d(x, y, kind='quadratic')

# x.min y x.max se usan para asegurar que no
# nos salimos del intervalo de interpolacion

xint = np.linspace(x.min(), x.max(), 1000)

yintl = fl(xint)

yintq = fq(xint)

plt.plot(xint, yintl, label='Lineal')
plt.plot(xint, yintq, label='Cuadratica')
plt.show()
```

En la siguiente figura vemos el resultado de la interpolación lineal:

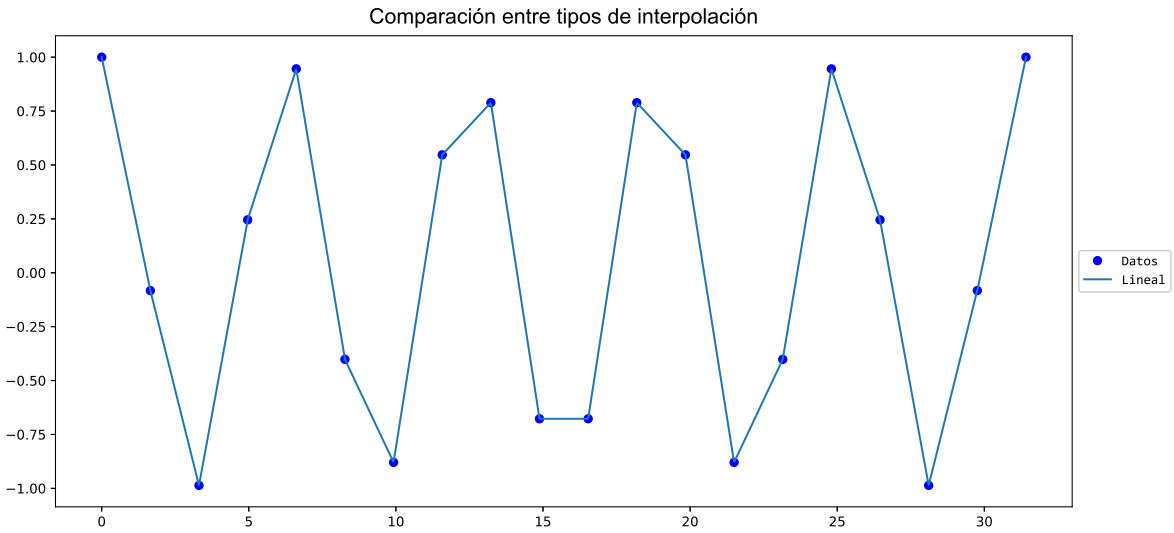

Figura 2: Resultado de la interpolación lineal

Para la interpolación cuadrática, vemos en la siguiente figura el resultado:

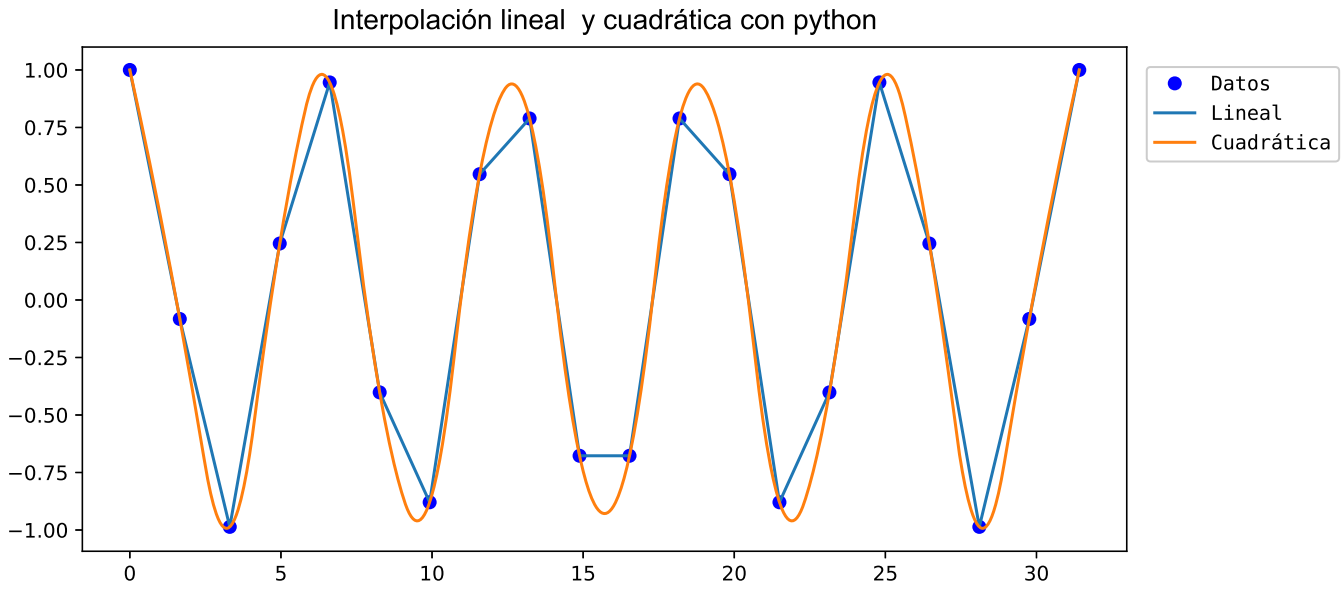

Figura 3: Resultado de la interpolación usando `kind=quadratic`

Veamos otro ejemplo.

Utilizaremos la función `sinc(x)` que está contenida dentro de la librería `numpy`.

La función `sinc (x)`, también llamada "función de muestreo", es una función que se encuentra comúnmente de las teorías de procesamiento de señales y de las transformadas de Fourier.

El nombre completo de la función es "seno cardinal", pero es comúnmente referido por su abreviatura, "sinc". Se define como

$$sinc(x) = \begin{cases} 1 & \text{para } x = 0 \\ \dfrac{\sin x}{x} & \text{para cualquier otro valor} \end{cases}$$

Entonces podemos presentar el siguiente código

Código 3: Inteporlando la función `sinx(x)`

```
x = np.linspace(-18, 18, 36)
ruido = 0.1 * np.random.random(len(x))
senal = np.sinc(x) + ruido

interpretada = interpolate.interp1d(x, senal)
x2 = np.linspace(-18, 18, 180)
y = interpretada(x2)

cubica = interpolate.interp1d(x, senal, kind="cubic")
y2 = cubica(x2)
```

El resultado de la intepolación lo podemos ver a continuación

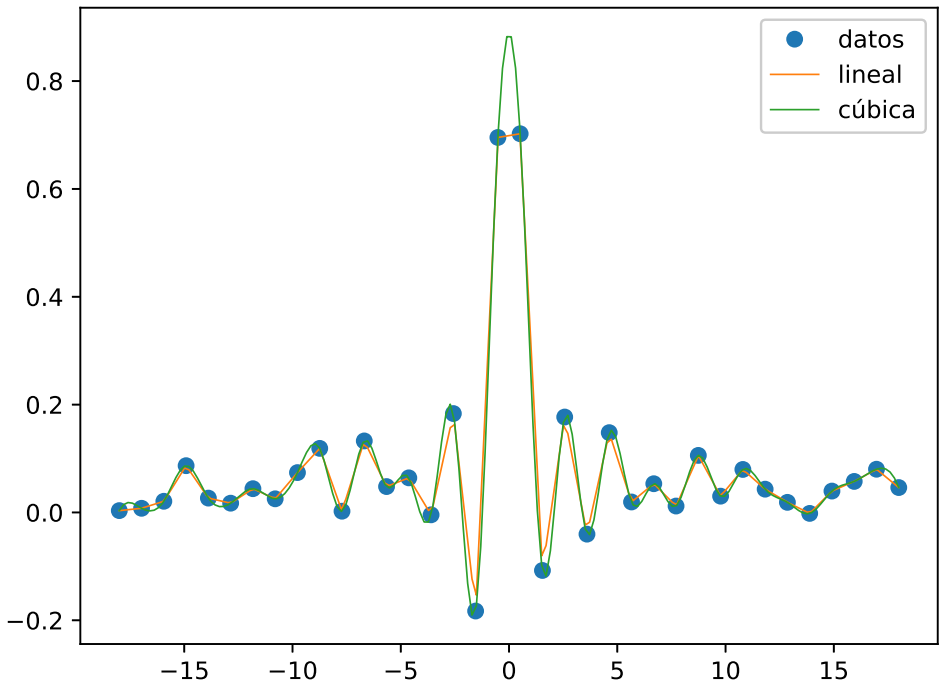

Figura 4: Interpolando puntos de la función `sinc(x)`

2. Más sobre la interpolación.

Es posible reconocer algunas características generales de la interpolación obtenida en las figuras:

1. Las funciones de interpolación son continuas.
2. Las funciones de interpolación pasan siempre por los puntos de datos.
3. Una función cuadrática puede dar un ajuste más malo que la interpolación lineal.
4. Aumentar el orden del polinomio no siempre conduce a un mejor ajuste.
5. Las funciones de interpolación pueden oscilar drásticamente entre los puntos de datos.
6. El ajuste empeora hacia los extremos del conjunto de datos.

Entonces, ¿qué debo hacer al interpolar mis propios datos?

3. Fenómeno de Runge

Hasta el momento hemos revisado un par de estrategias para calcular un polinomio que pase por un conjunto de datos (x_i, y_i), pero hay que considerar un efecto importante al respecto: no siempre el mejor polinomio será aquel el de mayor grado n.

Veamos el siguiente ejemplo: sea la función:

$$f(x) = \frac{1}{1 + 25x^2}$$

La gráfica de la función $f(x)$ es la siguiente

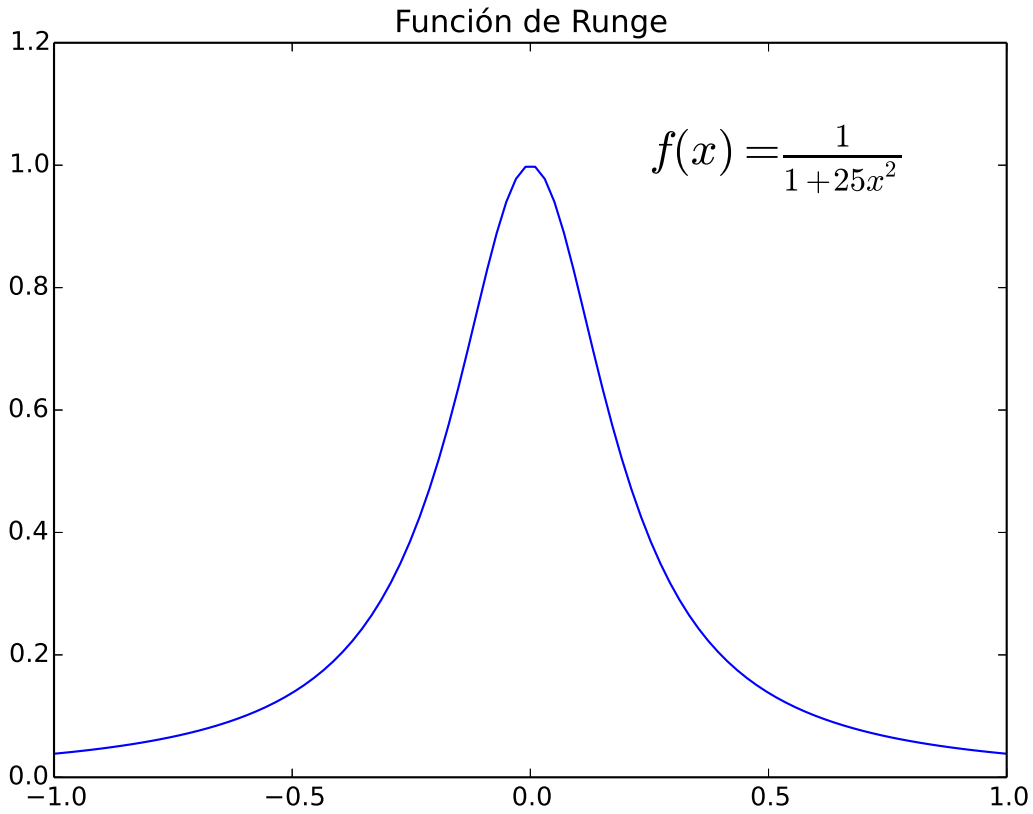

Figura 5: Función de Runge

Elección de puntos para interpolar.

Hagamos una elección aleatoria de puntos:

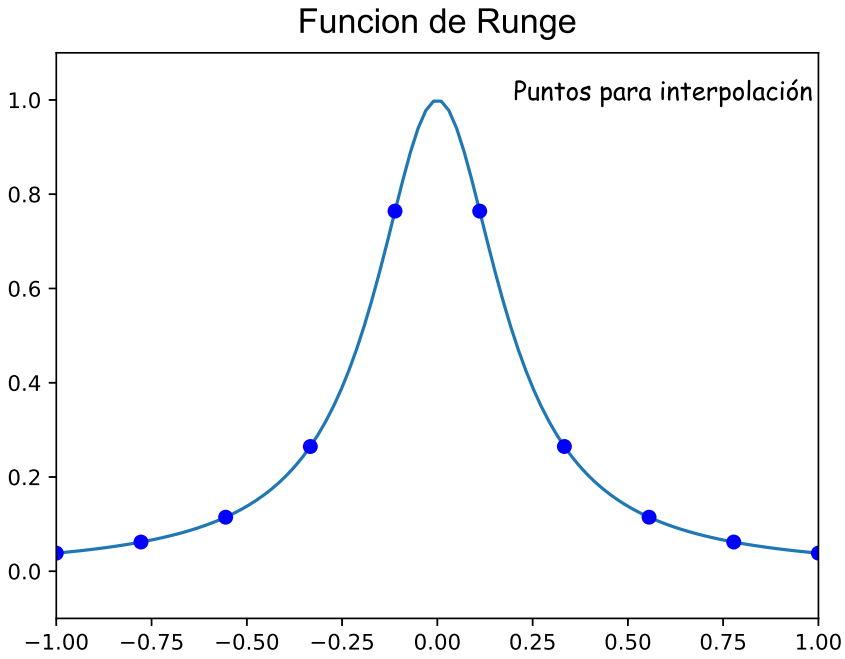

Figura 6: Elección de puntos.

Si hacemos una interpolación mediante el método de Lagrange, obtendremos lo siguiente:

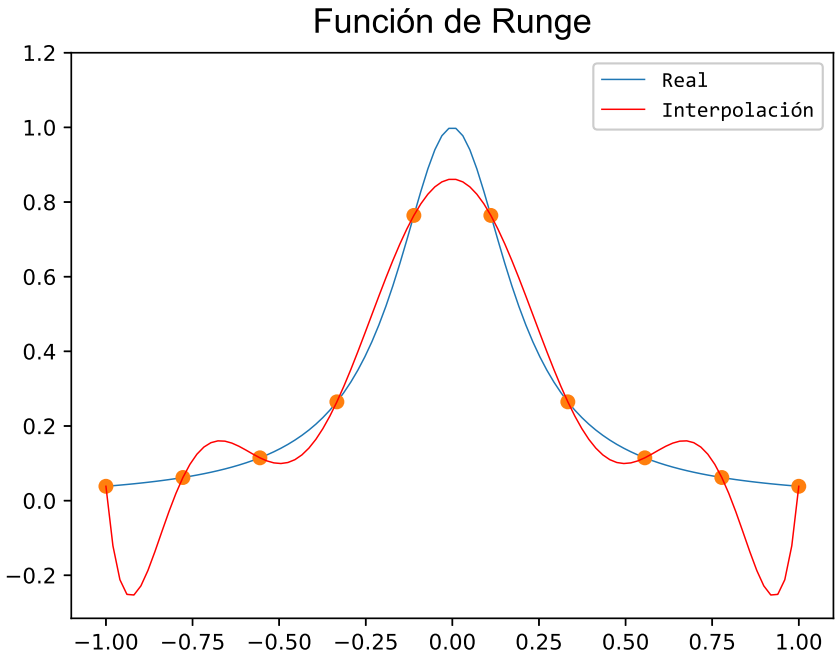

Figura 7: Gráfica de la interpolación con Lagrange.

¿Qué hacemos al respecto?

Como hemos visto en la gráfica anterior, la función que resulta del proceso de interpolación "oscila" a través de los puntos que deseamos interpolar; aquí el caso es que si aumentamos el grado del polinomio, los resultados serán aún más indeseables.

La pregunta obligada es: ¿qué podemos hacer para mejorar la interpolación?

3.1. ¿Qué es un spline?

En términos nada riguroso, se puede decir que un spline es una función definida por una familia de polinomios "sociables", donde el término sociable se usa para indicar que los polinomios que constituyen una función *spline*, están estrechamente vinculados.

El nombre de *spline*, viene del inglés ya que es un instrumento que utilizaban los ingenieros navales para dibujar curvas suaves, forzadas a pasar por un conjunto de puntos prefijados.

3.2. Las tres B's de los splines

El uso de las funciones splines tiene mucha aceptación y popularidad se deben a tres razones básicas, ya que son:

1. **Buenos:** Se pueden usar en la solución de una gran variedad de problemas.
2. **Bonitos:** La teoría matemática en que se basan es muy simple y a la vez elegante.
3. **Baratos:** Ya que su cálculo es muy sencillo y económico.

4. Manejando splines con python

Usaremos la librería scipy que contiene varias funciones con las que ahorramos tiempo para manejar splines y ajustar funciones sociables a un conjunto de datos.

No está de más que revises la teoría al respecto, en la mayoría de los libros de análisis numérico, podrás encontrar la construcción matemática y formal de los splines.

Funciones para los splines

Necesitaremos de dos pasos para el uso de splines con python, de la librería: scipy.interpolate, las cuales son:

1. splrep: Calcula el spline básico (B-spline) para una curva 1-D.
   Dado un conjunto de puntos $(x[i], y[i])$, la función determina una aproximación con un spline suave de grado k en el intervalo $xb \leq x \leq xe$.
2. splev: Evalúa un B-spline o sus derivadas.
   Dados los nodos y coeficientes de un B-spline, calcula el valor del polinomio suave y sus derivadas.

La sintaxis mínima de la función splrep es la siguiente:
`splrep(x, y, xb=None, xe=None, k=3)` donde:

- x, y : son arreglos de valores que definen la función $y = f(x)$.
- xb, xe : es el intervalo en donde se realizará el ajuste. En caso de que no se proporcione, se tomarán por defecto $x[0]$ y $x[-1]$, respectivamente.
- k : es un entero (`int`), es un argumento opcional. Corresponde al grado del spline. Se recomienda usar splines cúbicos.

La sintaxis mínima para la función splev es:
`splev(x, tck)` donde

- x : es un arreglo. Considera los nodos y los coeficientes del B-spline, calcula el valor del polinomio suavizado.
- tck : es un tupla de 3 elementos. Representa la secuencia de elementos que devuelve splrep que contiene los nodos, los coeficientes y el grado del spline.

Hagamos un ejemplo para revisar el uso de los splines, seguiremos con la función de Runge:

Código 4: Código para usar un spline

```
1  import matplotlib.pyplot as plt
2  import scipy.interpolate as si
3  import numpy as np
4
5  x = np.linspace(-1, 1, 100)
6  y = 1./(1 + 25 * x**2)
7
8  def trazadorCub(n):
9      xi = np.linspace(-1, 1, n)
10     yi = 1./(1 + 25 * xi**2)
11     tck = si.splrep(xi, yi)
12     return tck
13
14 tck = trazadorCub(8)
15 ys8 = si.splev(x, tck)
```

```
16
17 tck = trazadorCub(12)
18 ys12 = si.splev(x, tck)
19
20 plt.plot(x, y)
21 plt.plot(x, ys8, '+g-', label='n=8')
22 plt.plot(x, ys12, '+r-', label ='n=12')
23 plt.legend(loc='best')
24 plt.title('Interpolacion con splines cubicos')
25 plt.ylim(-0.2, 1.2)
26 plt.show()
```

La función `trazadorCub(n)`, genera el número de puntos "medidos o experimentales", en el primer caso, genera 8 puntos espaciados de manera equidistante, mientras que en el segundo caso, tenemos 12 puntos distribuidos sobre el eje$x$.

Al usar la función `splev`, se introducen 100 valores para generar el spline cúbico. La gráfica que se genera y en donde podemos ver el resultado al calcular el spline, es la siguiente:

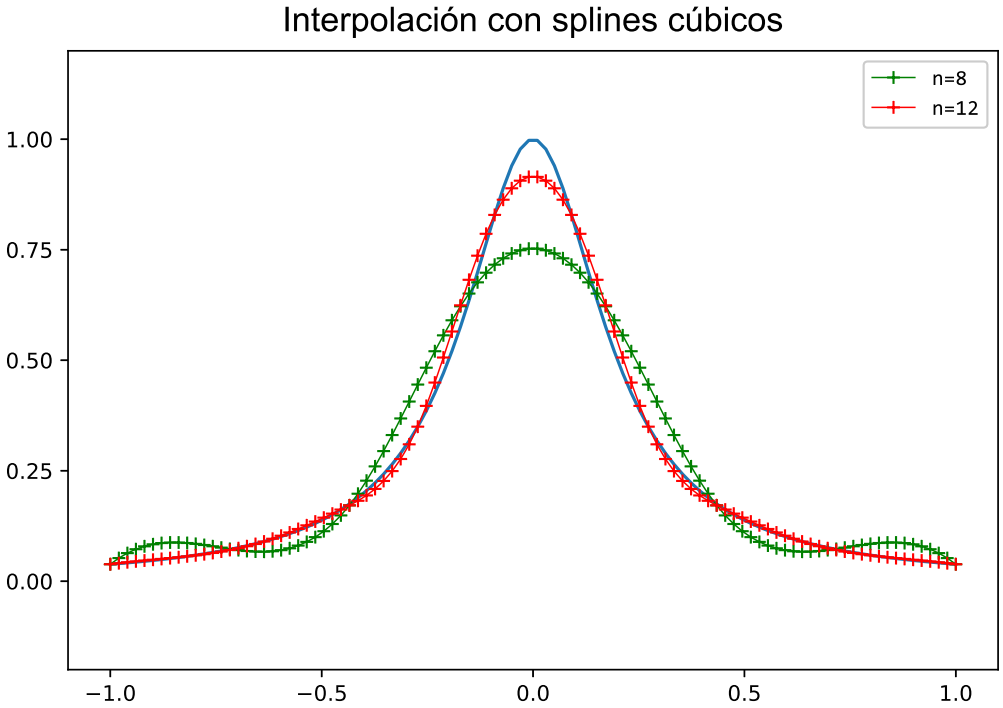

Figura 8: Uso de los splines en python.