Funciones de interpolación con `python`

M. en C. Gustavo Contreras Mayén

27 de febrero de 2018

Contenido

1. Funciones de interpolación con `python`.
   1.1 La función `interp1d (x, y)`

Contenido

1. Funciones de interpolación con `python`.
   1.1 La función `interp1d (x, y)`

2. Fenómeno de Runge
   2.1 Uso de splines

`python` cuenta con 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.

La librería que debemos de utilizar es `scipy.interpolate`

La función `interp1d (x, y)`

Dentro de `scipy.interpolate` contamos con la función `interp1d` que requiere de dos argumentos - los valores de x e y que se utilizarán para la interpolación.

Se puede proporcionar un tipo de argumento opcional que especifica el tipo de procedimiento de interpolación.

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.

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.

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.

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.

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.

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 puntos. Por ejemplo:

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

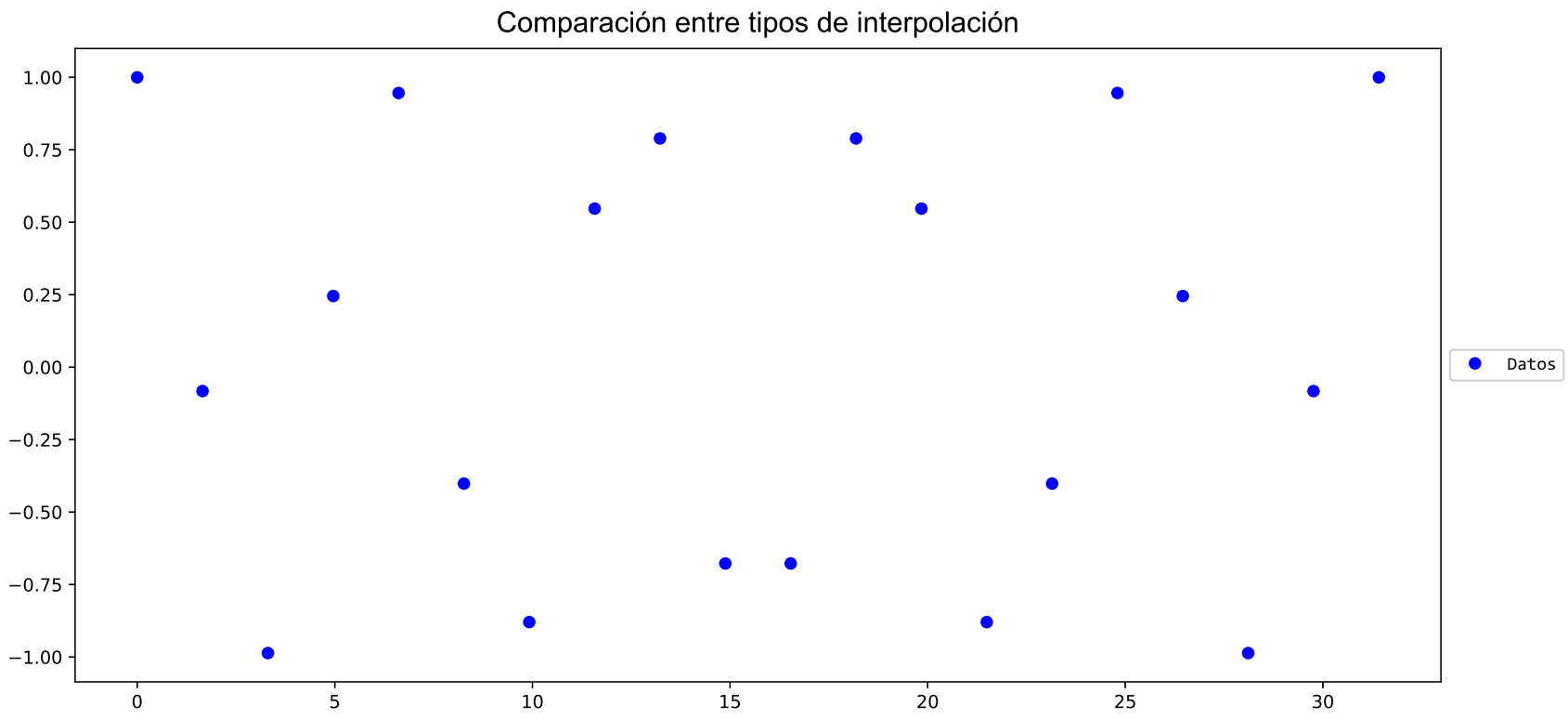
Utilizará un polinomio de orden 10 para interpolar entre puntos.

Ejemplo

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

```
1 x = np.linspace(0, 10 * np.pi, 20)
2 y = np.cos(x)
3
4 #para graficar
5
6 plt.plot(x,y, 'bo', label='Datos')
```

Resultado en la gráfica



Se interpolan los datos I

```
1 # Se interpolan los datos
2
3 fl = interp1d(x, y, kind='linear')
4
5 fq = interp1d(x, y, kind='quadratic')
6
7 # x.min and x.max se usan para asegurar
    que no
```

Se interpolan los datos II

```
8  # nos salimos del intervalo de
       interpolacion
9
10 xint = np.linspace(x.min(), x.max(), 1000)
11
12 yintl = fl(xint)
13
14 yintq = fq(xint)
15
16 #para graficar
```

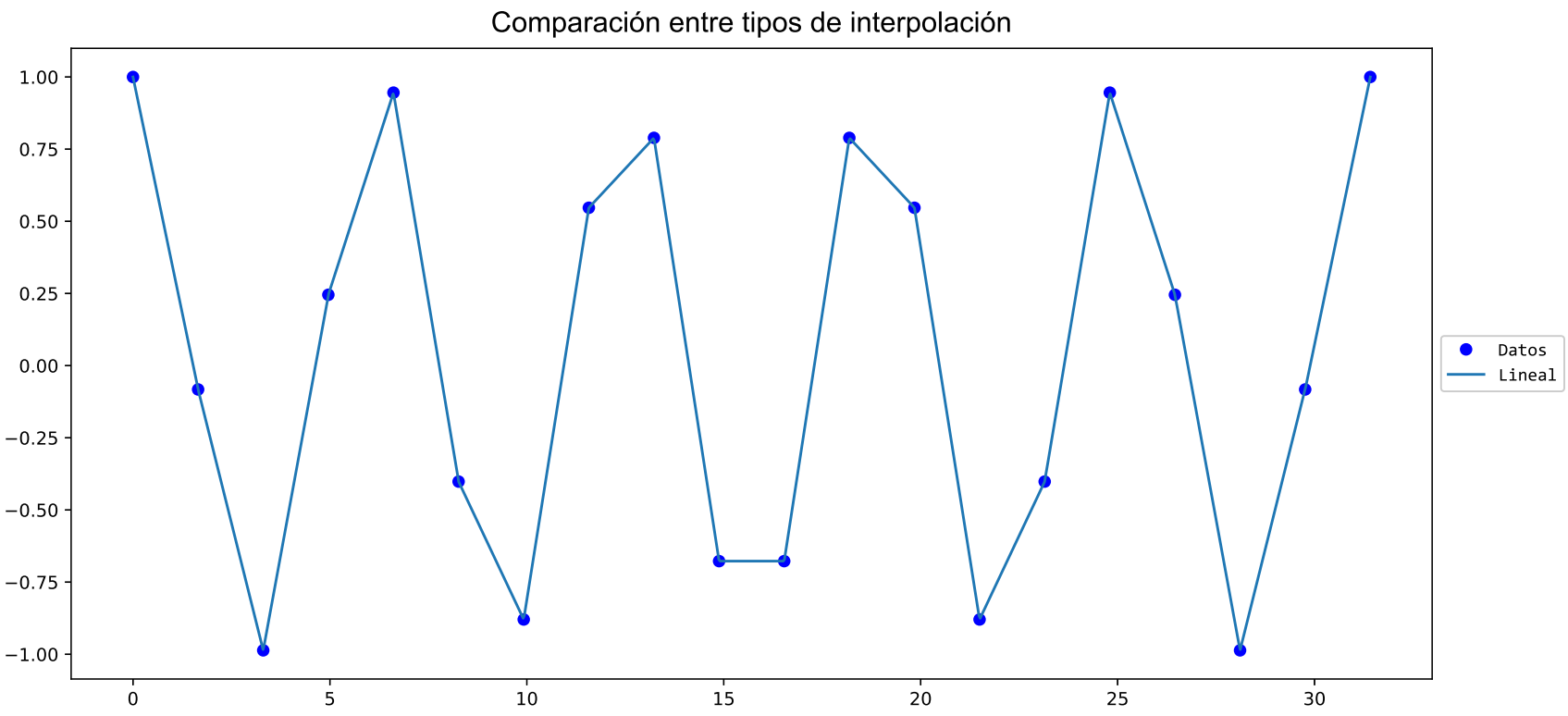
Se interpolan los datos III

```
17
18 plt.plot(xint, yintl, label='Lineal')
19 plt.plot(xint, yintq, label='Cubica')
```

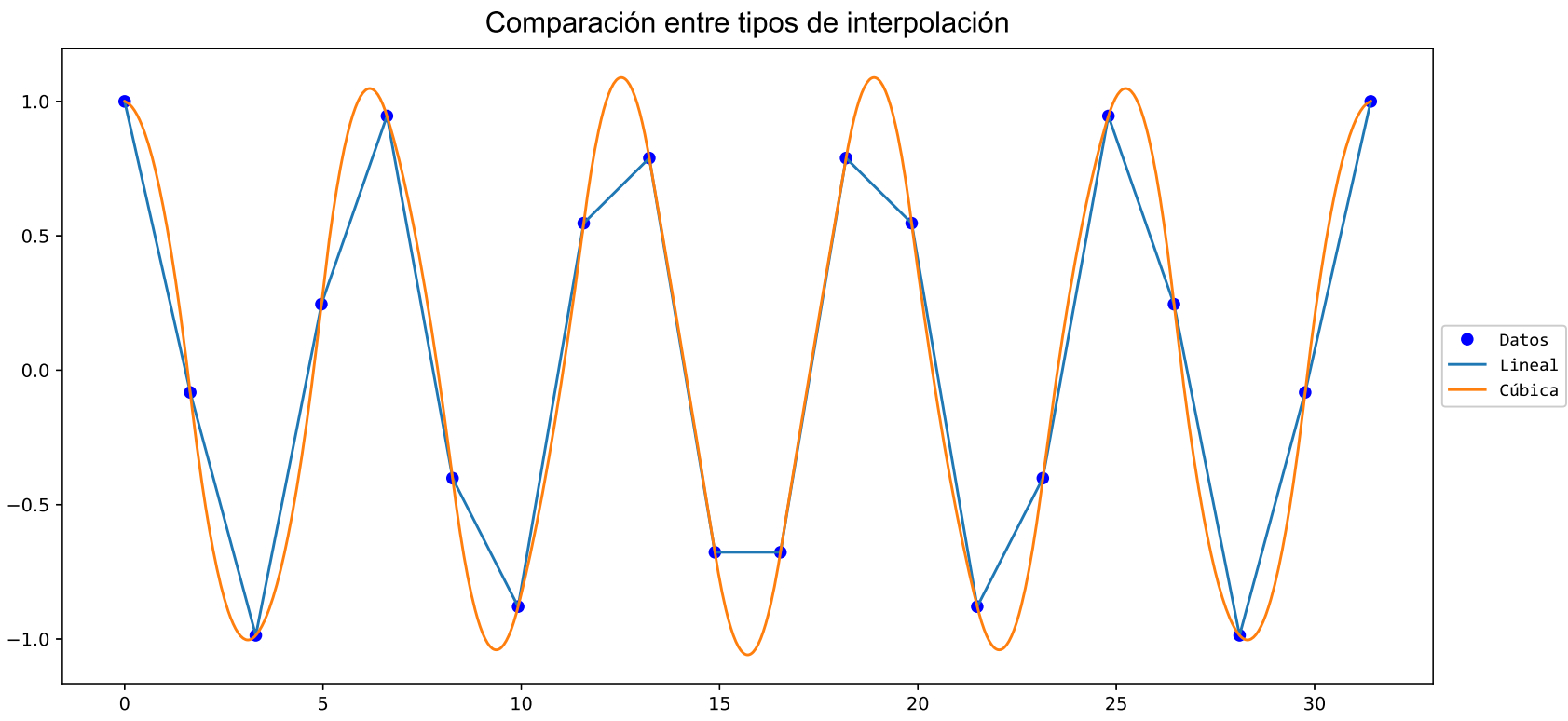
Se grafican los datos

```
1 #para graficar
2
3 plt.plot(xint, yintl, label='Lineal')
4
5 plt.plot(xint, yintq, label='Cubica')
```

Interpolación lineal



Interpolación cúbica



Veamos otro ejemplo.

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

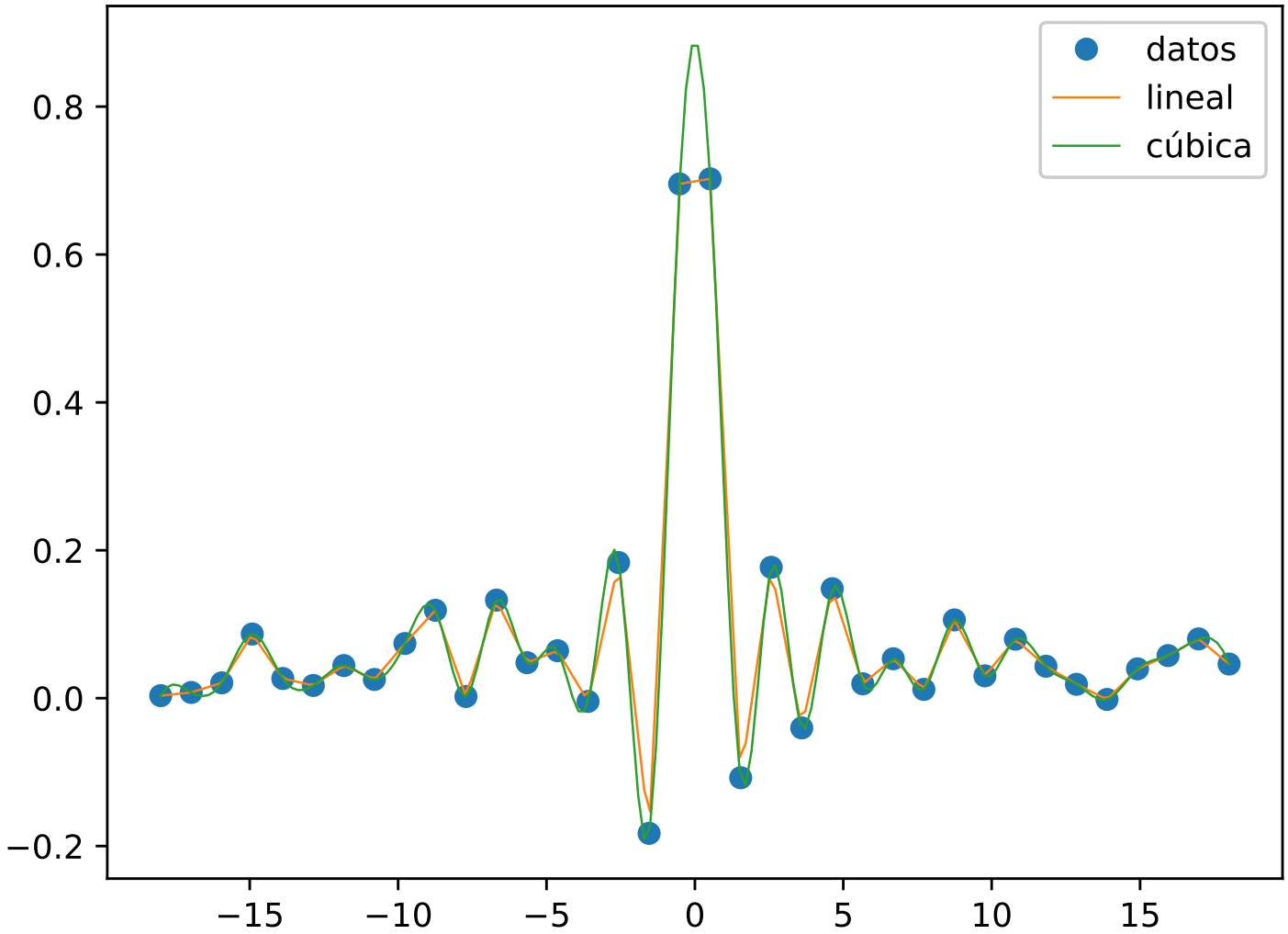
La función `sinc (x)`, también llamada “función de muestreo”, es una función que se encuentra comúnmente de las teorías de procesamiento de señales y de las transformadas de Fourier.

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

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

```
1  x = np.linspace(-18, 18, 36)
2  ruido = 0.1 * np.random.random(len(x))
3  senal = np.sinc(x) + ruido
4
5  interpretada = interpolate.interp1d(x,
       senal)
6  x2 = np.linspace(-18, 18, 180)
7  y = interpretada(x2)
8
9  cubica = interpolate.interp1d(x, senal,
       kind="cubic")
10 y2 = cubica(x2)
```

Interpolación con la señal de muestreo



Más sobre la interpolación.

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

1. Las funciones de interpolación son continuas.

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.

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.

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.

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.

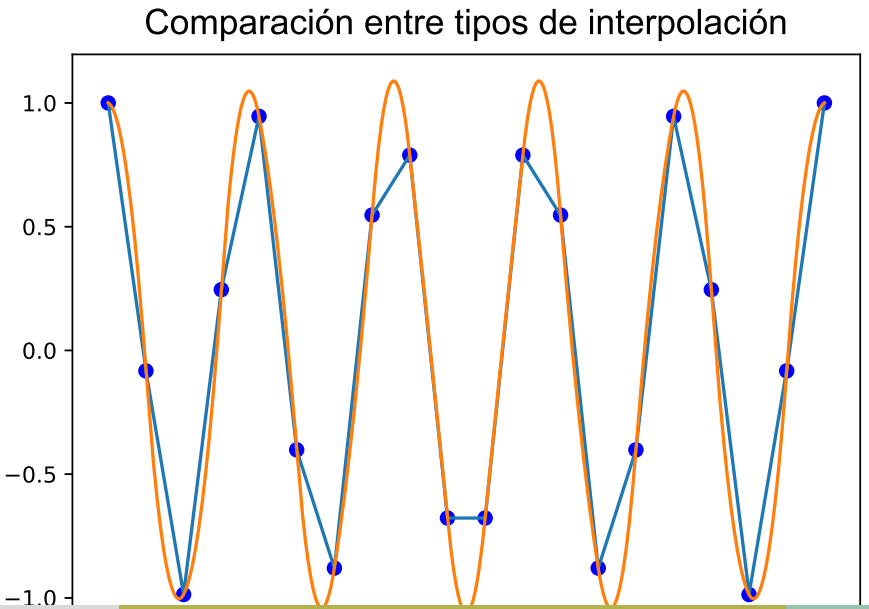
Más sobre la interpolación.

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.

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

El “spline cúbico” es el caballo de batalla en este terreno. Como se puede ver en la figura, proporciona una curva suave que parece ajustarse bien a los datos.



La suavidad se extiende más allá de lo que se ve en la gráfica: un “spline cúbico” tiene derivadas primera y segunda continuas.

Esta es una propiedad útil en la física, donde las derivadas primera y segunda son bastante comunes en los análisis teóricos (leyes de Newton, ecuaciones de Maxwell, ecuación de Schröedinger, etc.)

La suavidad se extiende más allá de lo que se ve en la gráfica: un “spline cúbico” tiene derivadas primera y segunda continuas.

Esta es una propiedad útil en la física, donde las derivadas primera y segunda son bastante comunes en los análisis teóricos (leyes de Newton, ecuaciones de Maxwell, ecuación de Schröedinger, etc.)

Una interpolación de spline cúbico es una buena opción en la mayoría de los casos.

Precaución con las funciones de interpolación

Precaución: la interpolación y la extrapolación no es lo mismo.

Una buena función de interpolación puede ser una aproximación muy mala fuera del conjunto de puntos de datos utilizados. Por esta razón, las funciones generadas por `interp1d (x, y)` ni siquiera devolverán un número cuando proporcione un valor de la variable independiente fuera del rango del conjunto de datos: se obtiene un `ValueError` en su lugar.

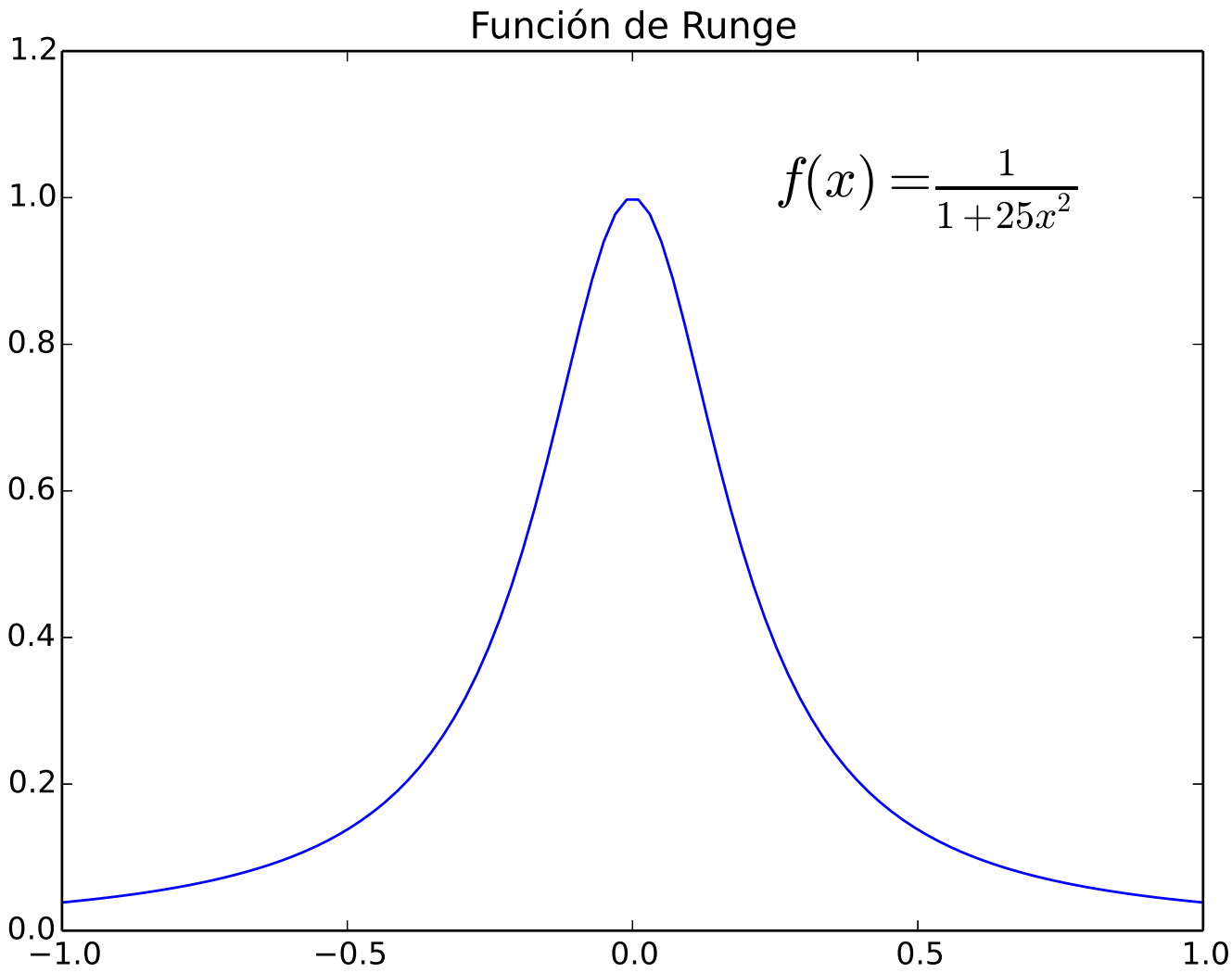
Fenónemo de Runge

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.

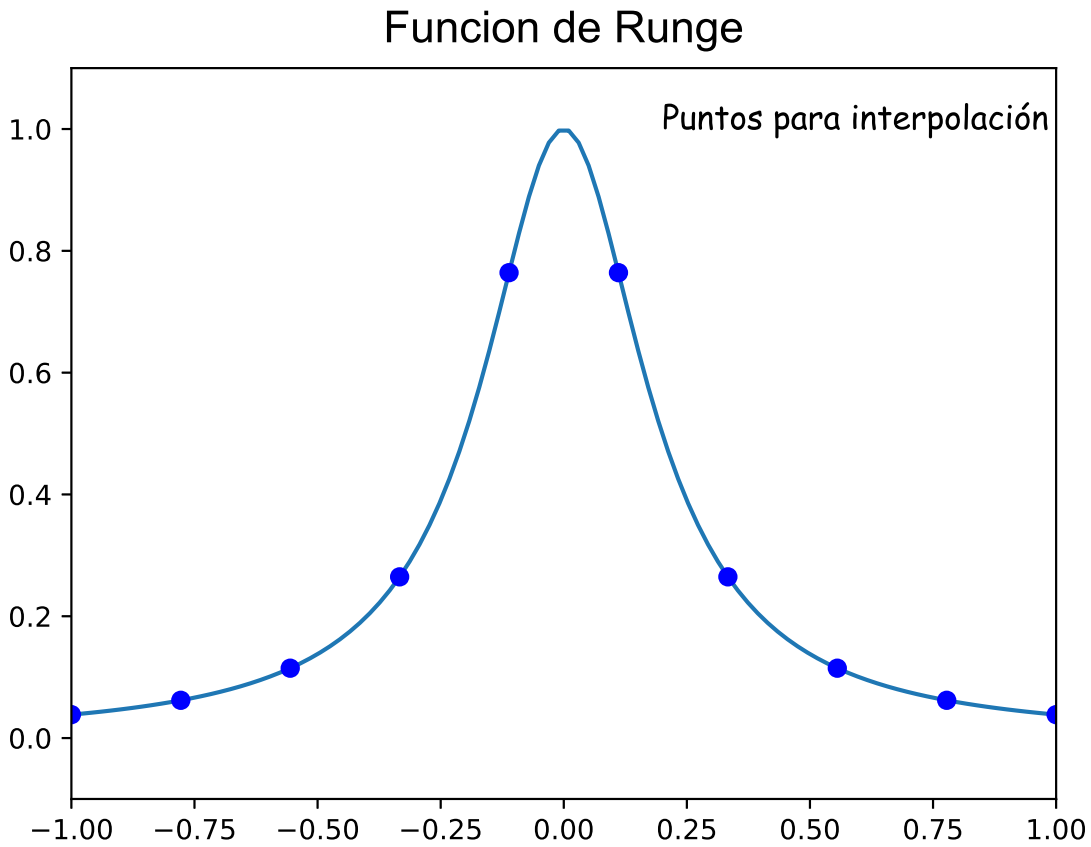
Veamos el siguiente ejemplo: sea la función:

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

Gráfica de la función $f(x)$

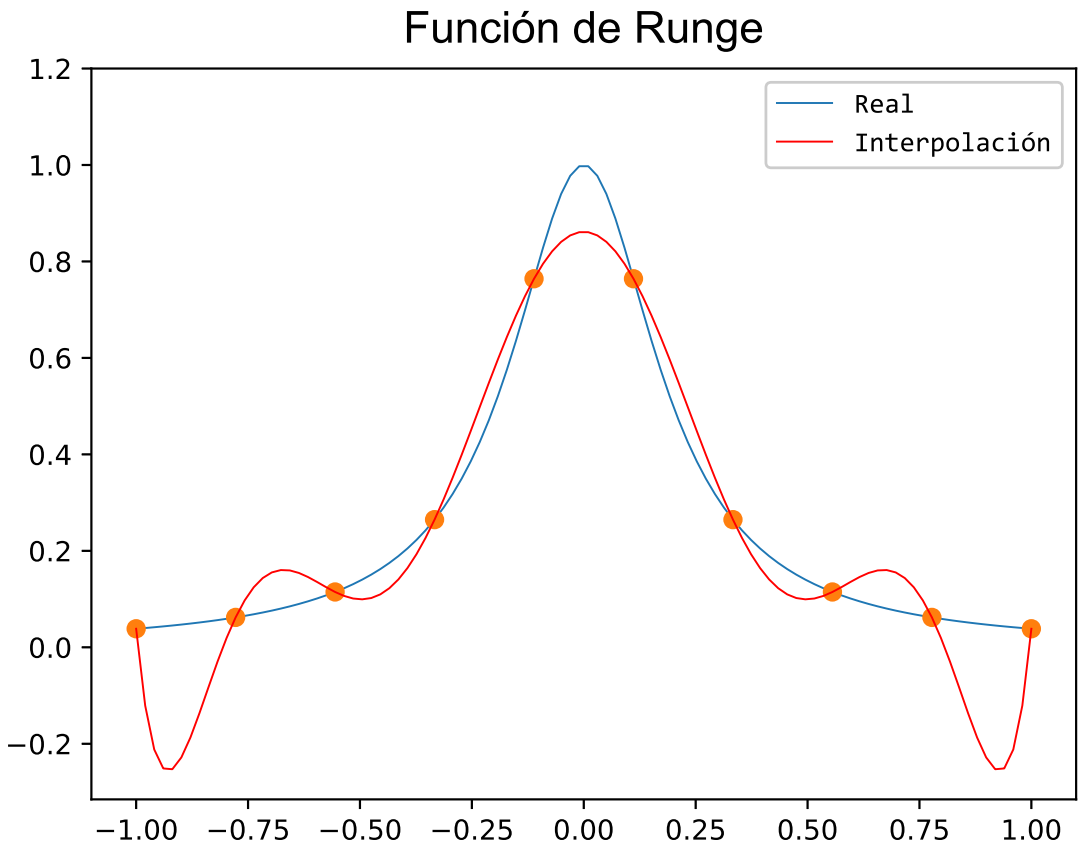


Elección de puntos para interpolar



Interpolación con Lagrange

Interpolando los puntos 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?

¿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.

Las tres B's de los splines

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

1. **Buenos:** Se pueden usar en la solución de una gran variedad de problemas.

Las tres B's de los splines

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

1. **Buenos:** Se pueden usar en la solución de una gran variedad de problemas.
2. **Baratos:** Ya que su cálculo es muy sencillo y económico.

Las tres B's de los splines

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

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

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 libería: `scipy.interpolate`, las cuales son:

1. **splrep**: Calcula el spline básico (B-spline) para una curva 1-D.

   Dados un conjunto de puntos $(x[i], y[i])$ 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.

Código I

```
1 import matplotlib.pyplot as plt
2 import scipy.interpolate as si
3 from numpy import *
4
5 x = linspace(-1, 1, 100)
6 y = 1./(1 + 25 * x**2)
7
8 def trazador_cub(n):
9     xi = linspace(-1, 1, n)
```

Código II

```
10      yi = 1./(1 + 25 * xi**2)
11      tck = si.splrep(xi, yi)
12      return tck
```

Código I

```
1 tck = trazador_cub(8)
2 ys8 = si.splev(x, tck)
3
4 tck = trazador_cub(12)
5 ys12 = si.splev(x, tck)
6
7 plt.plot(x, y)
8 plt.plot(x, ys8,'+g-', label='n=8')
9 plt.plot(x, ys12,'+r-',label ='n=12')
```

Código II

```
10 plt.legend(loc='best')
11 plt.title('Interpolacion con splines
       cubicos')
12 plt.ylim(-0.2, 1.2)
13 plt.show()
```

Resultados gráficos