Tema 2 - Operaciones matemáticas básicas

Técnicas de Interpolación II

M. en C. Gustavo Contreras Mayén

Facultad de Ciencias - UNAM

28 de febrero de 2018

Técnicas de interpolación

Continuamos revisando algunas técnicas de inteporlación de datos.

Interpolación de Newton

Aunque el método de interpolación de Lagrange es conceptualmente sencillo, no es en sí, un algoritmo eficiente.

Interpolación de Newton

Aunque el método de interpolación de Lagrange es conceptualmente sencillo, no es en sí, un algoritmo eficiente.

Un mejor método computacional se obtiene con el Método de Newton, donde el polinomio de interpolación se escribe de la forma:

$$P_n(x) = a_0 + (x - x_0)\, a_1 + (x - x_0)(x - x_1)\, a_2 + \ldots + \\ + (x - x_0)(x - x_1) \ldots (x - x_{n-1})\, a_n$$

Polinomio de Newton

Este polinomio nos permite contar con un procedimiento de evaluación más eficiente.

Por ejemplo, con cuatro pares de datos ($n = 3$), tenemos que el polinomio de interpolación es:

$$P_3(x) = a_0 + (x - x_0)\, a_1 + (x - x_0)(x - x_1)\, a_2 + \\ + (x - x_0)(x - x_1)(x - x_2)\, a_3$$

$$P_3(x) = a_0 + \{a_1 + (x - x_1)\,[a_2 + (x - x_2)\, a_3]\}$$

que puede ser evaluado hacia atrás con las siguientes relaciones de recurrencia:

Relaciones de recurrencia

$$
\begin{aligned}
P_0 &= a_3 \\
P_1 &= a_2 + (x - x_2)\, P_0(x) \\
P_2 &= a_1 + (x - x_1)\, P_1(x) \\
P_3 &= a_0 + (x - x_0)\, P_2(x)
\end{aligned}
$$

Relaciones de recurrencia

$$
\begin{aligned}
P_0 &= a_3 \\
P_1 &= a_2 + (x - x_2)\, P_0(x) \\
P_2 &= a_1 + (x - x_1)\, P_1(x) \\
P_3 &= a_0 + (x - x_0)\, P_2(x)
\end{aligned}
$$

Para un n arbitrario, tenemos:

Relaciones de recurrencia

$$
\begin{aligned}
P_0 &= a_3 \\
P_1 &= a_2 + (x - x_2)\, P_0(x) \\
P_2 &= a_1 + (x - x_1)\, P_1(x) \\
P_3 &= a_0 + (x - x_0)\, P_2(x)
\end{aligned}
$$

Para un n arbitrario, tenemos:

$$
\begin{aligned}
P_0(x) &= a_n \\
P_k &= a_{n-k} + (x - x_{n-k})\, P_{k-1}(x), \quad k = 1, 2, \ldots, n
\end{aligned}
$$

Definimos `xDatos` para las coordenadas x del conjunto de puntos y n al grado de polinomio, podemos usar el siguiente algoritmo para calcular el polinomio de Newton $P_n(x)$:

Código 1: Cálculo del polinomio de Newton

```
1 p = a[n]
2 for k in range(1, n + 1):
3     p = a[n - k] + (x - xDatos[n - k
    ])*p
```

Construcción del algoritmo

Los coeficientes de P_n se calculan forzando que el polinomio pase a través del conjunto de puntos

$$y_i = P_n(x_i), \quad i = 0, 1, \ldots, n$$

Construcción del algoritmo

De tal manera que tenemos un sistema de ecuaciones simultáneas:

$$
\begin{aligned}
y_0 &= a_0 \\
y_1 &= a_0 + (x_1 - x_0)\, a_1 \\
y_2 &= a_0 + (x_2 - x_0)\, a_1 + (x_2 - x_0)(x_2 - x_1)\, a_2 \\
&\vdots \\
y_n &= a_0 + (x_n - x_0)\, a_1 + \ldots + \\
&\quad + (x_n - x_0)(x_n - x_1) \ldots (x_n - x_{n-1})\, a_n
\end{aligned}
$$

Diferencias divididas

Se introducen las diferencias divididas, de la siguiente forma:

$$\nabla y_i = \frac{y_i - y_0}{x_i - x_0}, \qquad i = 1, 2, \ldots, n$$

Diferencias divididas

Se introducen las diferencias divididas, de la siguiente forma:

$$\nabla y_i = \frac{y_i - y_0}{x_i - x_0}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^2 y_i = \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \qquad i = 1, 2, \ldots, n$$

Diferencias divididas

Se introducen las diferencias divididas, de la siguiente forma:

$$\nabla y_i = \frac{y_i - y_0}{x_i - x_0}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^2 y_i = \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^3 y_i = \frac{\nabla^2 y_i - \nabla^2 y_2}{x_i - x_2}, \qquad i = 1, 2, \ldots, n$$

Diferencias divididas

Se introducen las diferencias divididas, de la siguiente forma:

$$\nabla y_i = \frac{y_i - y_0}{x_i - x_0}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^2 y_i = \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^3 y_i = \frac{\nabla^2 y_i - \nabla^2 y_2}{x_i - x_2}, \qquad i = 1, 2, \ldots, n$$

$$\vdots$$

Diferencias divididas

Se introducen las diferencias divididas, de la siguiente forma:

$$\nabla y_i = \frac{y_i - y_0}{x_i - x_0}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^2 y_i = \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \qquad i = 1, 2, \ldots, n$$

$$\nabla^3 y_i = \frac{\nabla^2 y_i - \nabla^2 y_2}{x_i - x_2}, \qquad i = 1, 2, \ldots, n$$

$$\vdots$$

$$\nabla^n y_i = \frac{\nabla^{n-1} y_n - \nabla^{n-1} y_{n-1}}{x_n - x_{n-1}}$$

La solución al sistema de ecuaciones es entonce:

$$
\begin{aligned}
a_0 &= y_0 \\
a_1 &= \nabla\, y_1 \\
a_2 &= \nabla^2\, y_2 \\
&\vdots \\
a_n &= \nabla^n\, y_n
\end{aligned}
$$

Coeficientes del polinomio

Si los coeficientes se calculan a mano, es conveniente escribirlos con el siguiente formato: (con $n = 4$)

x_0	y_0				
x_1	y_1	∇y_1			
x_2	y_2	∇y_2	$\nabla^2 y_2$		
x_3	y_3	∇y_3	$\nabla^2 y_3$	$\nabla^3 y_3$	
x_4	y_4	∇y_4	$\nabla^2 y_4$	$\nabla^3 y_4$	$\nabla^4 y_4$

Coeficientes del polinomio

Los términos en la diagonal principal

$$y_0, \nabla y_1, \nabla^2 y_2, \nabla^3 y_3, \nabla^4 y_4$$

son los coeficientes del polinomio.

Coeficientes del polinomio

Los términos en la diagonal principal

$$y_0, \nabla y_1, \nabla^2 y_2, \nabla^3 y_3, \nabla^4 y_4$$

son los coeficientes del polinomio.

Si los puntos de datos se enumeran en un orden diferente, las entradas de la tabla van a cambiar, pero el polinomio resultante será el mismo, recordemos que un polinomio de interpolación de grado n con $n + 1$ datos diferentes, es único.

Coeficientes del polinomio

Las operaciones en la computadora se pueden realizar con un arreglo unidimensional a, usando el siguiente algoritmo (tomando la notación $m = n + 1$ = número de puntos):

Código 2: Coeficiente del polinomio de Newton

```
1 a = yDatos.copy()
2 for k in range(1, m):
3     for i in range(k, m):
4         a[i] = (a[i] - a[k-1])/(
    xDatos[i] - xDatos[k-1])
```

Inicialmente el arreglo a contiene las coordenadas y del conjunto de datos, es decir, la segunda columna de la tabla.

Inicialmente el arreglo a contiene las coordenadas y del conjunto de datos, es decir, la segunda columna de la tabla.

Cada vez que pasa por el bucle externo, se genera la siguiente columna, por lo que se sobre-escriben los elementos de a, por tanto, al concluir el bucle, a contiene los elementos de la diagonal, que son los coeficientes del polinomio.

Módulos con `python`

Para facilitar el mantenimiento y la lectura los programas demasiado largos pueden dividirse en módulos, agrupando elementos relacionados.

Los módulos son entidades que permiten una organización y división lógica de nuestro código. Los archivos son su contrapartida física: cada archivo de `python` almacenado en disco equivale a un módulo.

Módulo con las funciones

Este módulo incluye dos funciones que se requieren para la interpolación de Newton: la función **`coeficientes`** y la función **`evaluaPoli`**.

La función `coeficientes`

Dados los conjuntos de puntos en los arreglos `xDatos` y `yDatos`, la función **`coeficientes`** devuelve el arreglo a con los coeficientes.

La función `evaluaPoli`

Una vez que ya conocemos los coeficientes, $P_n(x)$ puede evaluarse para cualquier valor de x con la función `evaluaPoli`.

Código 3: Funciones en el módulo `evalPoli`

```
1  def coeficientes(xDatos, yDatos):
2      m = len(xDatos)
3      a = yDatos.copy()
4      for k in range(1, m):
5          a[k:m] = (a[k:m] - a[k-1])/(
       xDatos[k:m] - xDatos[k-1])
6      return a
7
8
9  def evaluaPoli(a, xDatos, x):
10     n = len(xDatos) - 1
11     p = a[n]
12     for k in range(1,n+1):
```

```
13            p = a[n-k] + (x - xDatos[n-k
    ]) * p
14        return p
```

Ejemplo

Los datos que se muestran en la siguiente tabla

x	0.15	2.30	3.15	4.85	6.25	7.95
y	4.79867	4.49013	4.2243	3.47313	2.66674	1.51909

Se obtuvieron de la función

$$f(x) = 4.8 \cos\left(\frac{\pi\, x}{20}\right)$$

Ejemplo

Con ese conjunto de datos, interpola mediante el polinomio de Newton para los puntos

$$x = 0, 0.5, 1.0, 1.5, \ldots, 7.5, 8.0$$

y compara los resultados con el valor “exacto” de los valores $y_i = f(x_i)$

Solución

¿Qué necesitamos?

Abriendo un archivo en Spyder 3, llamamos al módulo `numpy` y también al módulo que contiene las funciones para resolver el polinomio de interpolación:

```
1 from numpy import *
2 from newtonPoli import *
```

Cargando los coeficientes

Hay que crear los arreglos `xDatos` y `yDatos`, el arreglo a se obtiene de la función **`coeficientes`** que está dentro del módulo **`NewtonPoli`**

Código 4: Datos iniciales

```
1 xDatos = array([0.15, 2.3, ..., 7.95
    ])
2 yDatos = array([4.79867, 4.49013,
    ..., 1.51909])
3
4 a = coeficientes(xDatos, yDatos)
```

Evaluación de los puntos I

La siguiente parte es proporcionar el rango de puntos y mandar llamar la función **`evalPoli`**:

Código 5: Evaluando los puntos

```
1 print ('{:^3} \t {:^7} \t {:^7} \t
    {:<11}'.format('x', 'yInterpol', '
    yExacta', 'Err relativo'))
2 print ('-'*55)
3
4 for x in np.arange(0.0, 8.1, 0.5):
5     y = evaluaPoli(a, xDatos, x)
```

Evaluación de los puntos II

```
6       yExacta = 4.8 * np.cos(np.pi*x/2
    0.0)
7       print ('{:1.1f} \t {:1.5f} \t {:
    1.5f} \t {:1.5E}'.format(x, y,
    yExacta, abs(yExacta - y)/yExacta*
    100))
```

Solución en la terminal

x	yInterp	yExacta	Err relativo
0.0	4.80003	4.80000	$5.22802E-04$
0.5	4.78518	4.78520	$5.16392E-04$
1.0	4.74088	4.74090	$5.70846E-04$
1.5	4.66736	4.66738	$3.19661E-04$
2.0	4.56507	4.56507	$9.67148E-05$
2.5	4.43462	4.43462	$1.57180E-05$
⋮			
7.0	2.17915	2.17915	$3.43797E-04$
7.5	1.83687	1.83688	$6.76648E-04$
8.0	1.48329	1.48328	$2.67576E-04$

Graficando la función exacta y los puntos

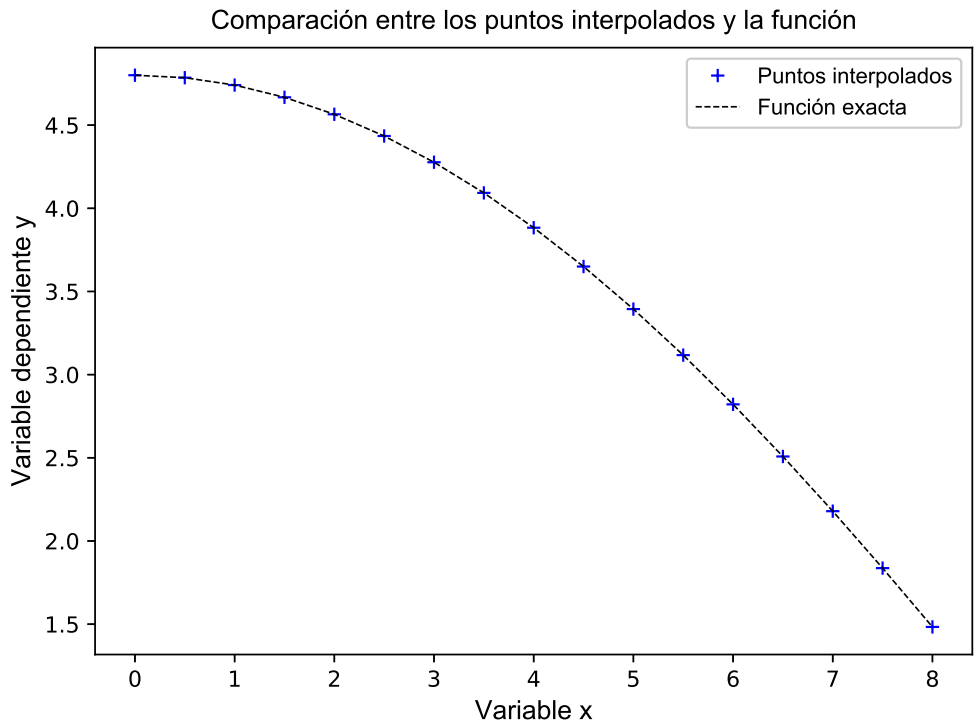


Figura 1: *Los puntos donde se evalúa con el polinomio de Newton, se superponen a la gráfica de la función "exacta".*

Interpolación con `python`

`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.

Interpolación con `python`

`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 la librería **`scipy.interpolate`** contamos con la función **`interp1d`** que requiere de dos argumentos - los valores de x e y que se utilizarán para la interpolación y un tercer argumento, que define el tipo de interpolación a realizar.

```
interp1d(x, y, kind="tipo interpolación")
```

La función `interp1d (x, y)`

Se puede proporcionar como tercer argumento opcional, un valor que especifica el tipo de procedimiento de interpolación.

En caso de no proporcionar algún valor, la función realiza una interpolación de tipo 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.

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.

Valor por defecto

El valor predeterminado del argumento **`kind`** 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.

Eligiendo el orden del polinomio

Por ejemplo:

Código 6: Ejemplo de orden de polinomio

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

Utilizará un polinomio de orden $n = 10$ para interpolar entre puntos (x, y).

Ejemplo

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

Código 7: Ejemplo para comparar el grado del polinomio de interpolación

```
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  #para graficar
8
9  plt.plot(x, y, 'bo', labe='Datos')
10 plt.show()
```

Los datos en una gráfica

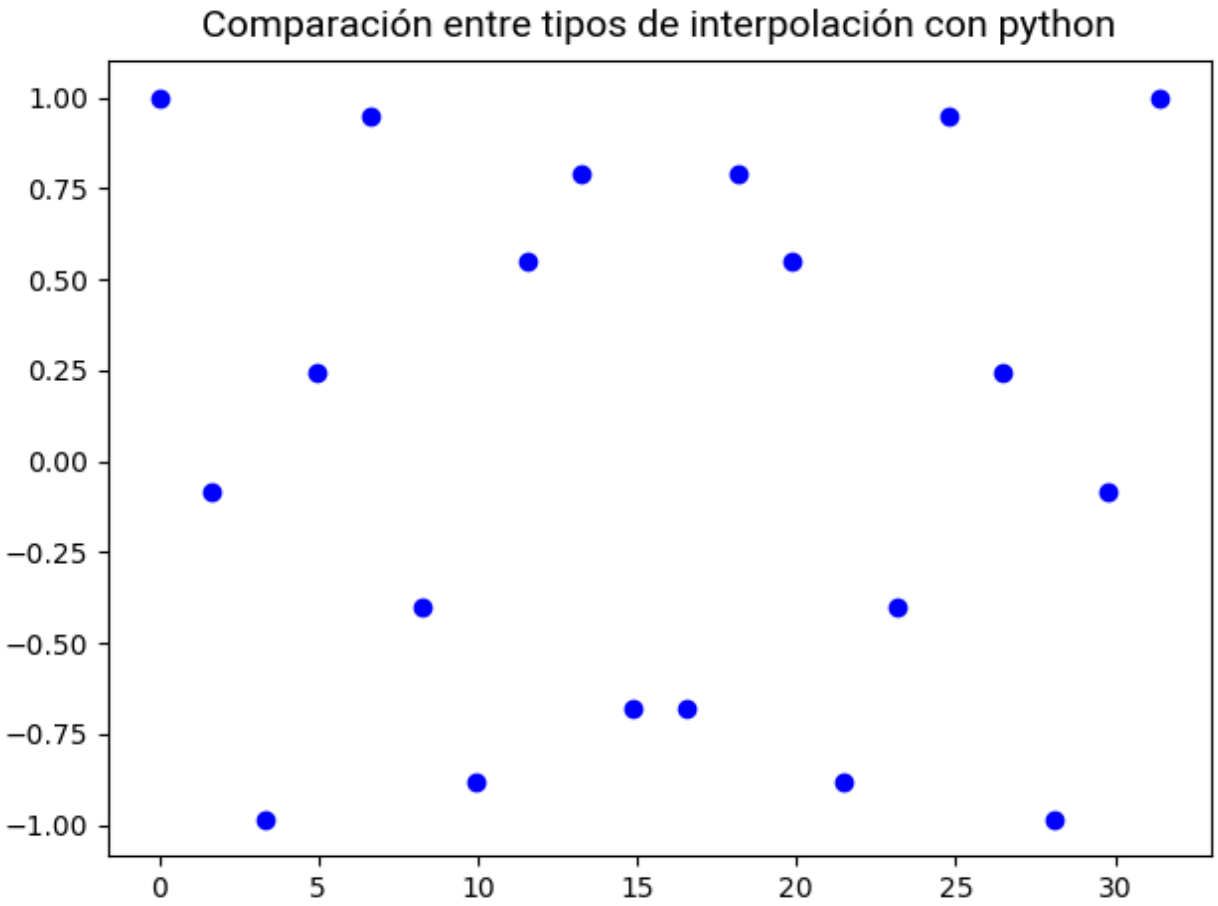


Figura 2: *Datos iniciales con los que vamos a interpolar.*

Se interpolan los datos I

Código 8: Cambiando el tipo de interpolación

```
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
8 # nos salimos del intervalo de
    interpolacion
9
```

Se interpolan los datos II

```
10 xint = np.linspace(x.min(), x.max(),
       1000)
11
12 yintl = fl(xint)
13
14 yintq = fq(xint)
15
16 #para graficar
17
18 plt.plot(xint, yintl, label='Lineal'
       )
19
20 plt.plot(xint, yintq, label='
       Cuadratica')
21
22 plt.legend(loc=1)
```

Se interpolan los datos III

```
23 plt.show()
```

Interpolación lineal

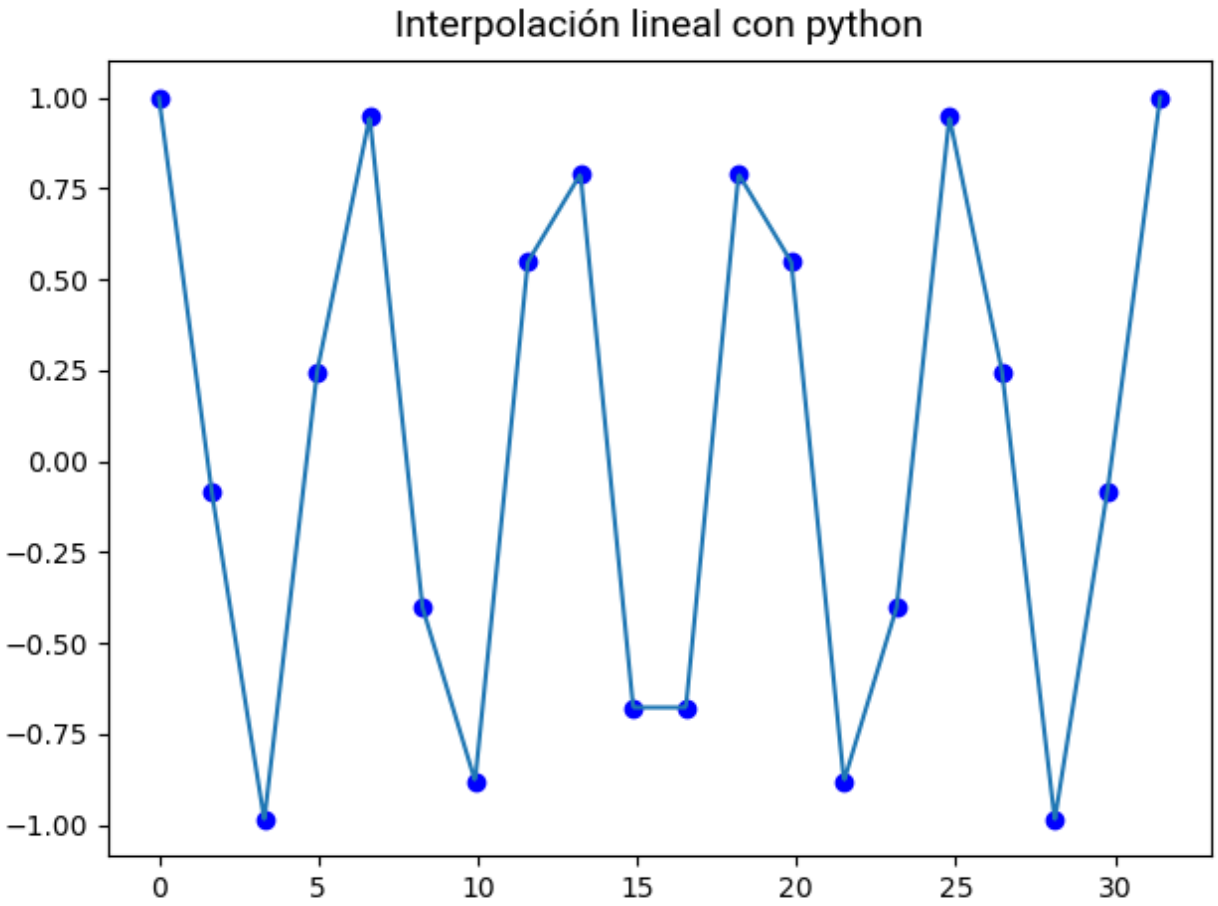


Figura 3: *La función que se obtiene con una interpolación lineal.*

Interpolación cuadŕatica

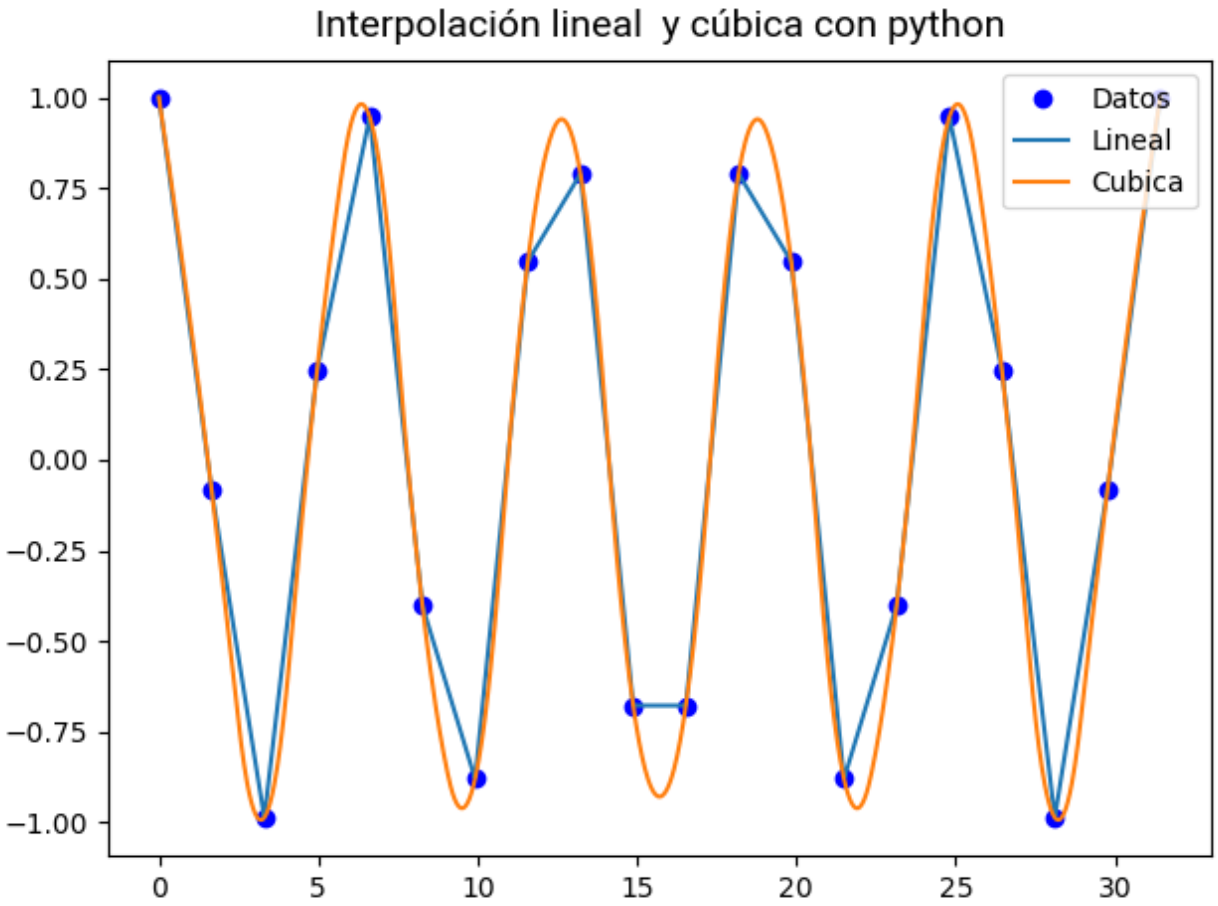


Figura 4: *La función que se obtiene con una interpolación cúbica.*

Veamos otro ejemplo: función `sinc(x)`

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.

Función seno cardinal

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

Se define como

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

Generamos algunos datos con `sinc(x)`

En el ejercicio vamos a generar un conjunto de datos aleatorio y se van a sumar a los valores de la función `sinc(x)`, de tal manera que representarían los datos experimentales.

Código con python I

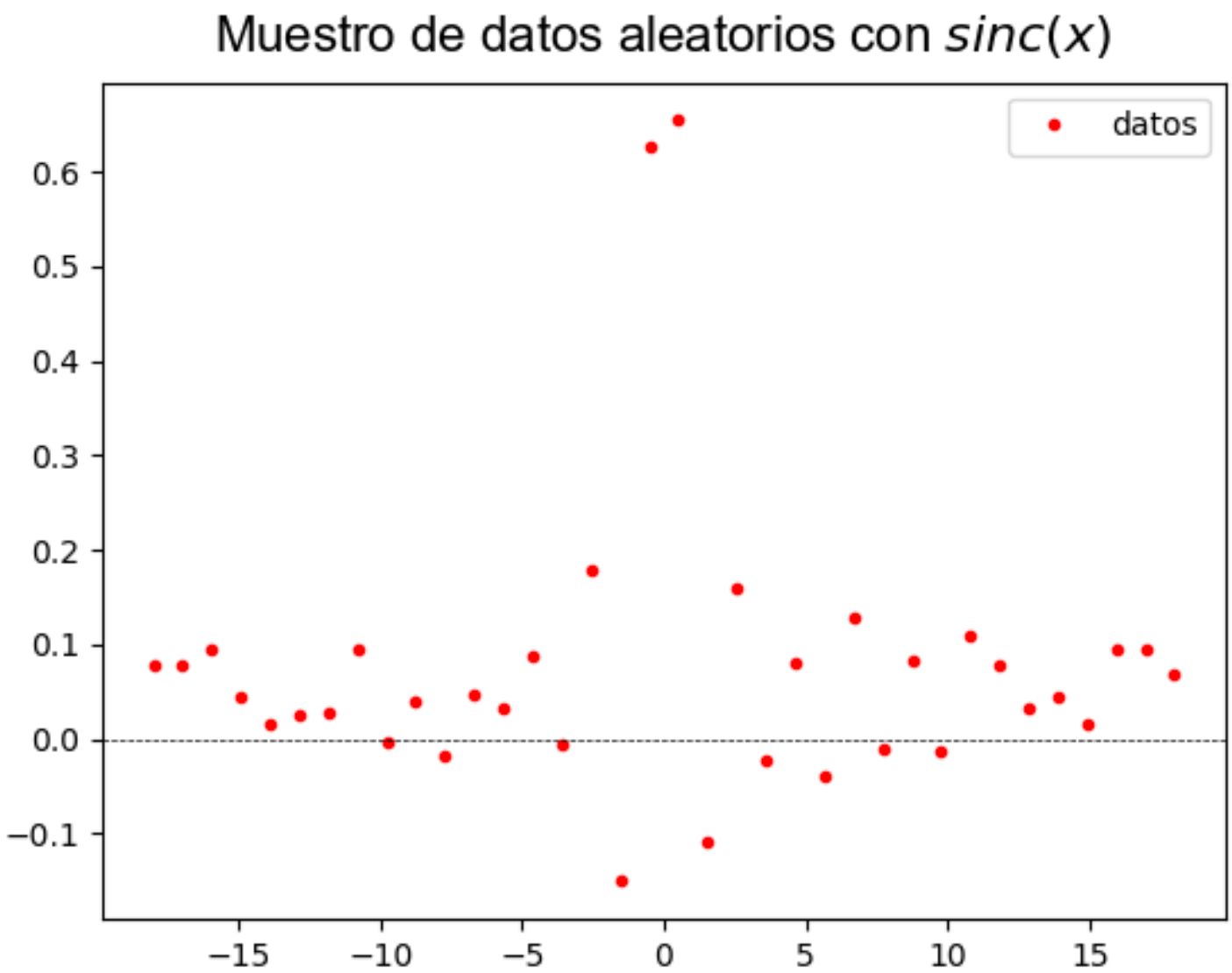
Código 9: Datos iniciales para la comparación

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  x = np.linspace(-18, 18, 36)
5
6  ruido = 0.1 * np.random.random(len(x
       ))
7  senal = np.sinc(x) + ruido
8
9  interpretada = interpolate.interp1d(
       x, senal)
10
11 x2 = np.linspace(-18, 18, 180)
```

Código con python II

```
12 y = interpretada(x2)
13
14 cubica = interpolate.interp1d(x,
       senal, kind="cubic")
15 y2 = cubica(x2)
```

Datos con la señal de muestreo



Función con interpolación lineal

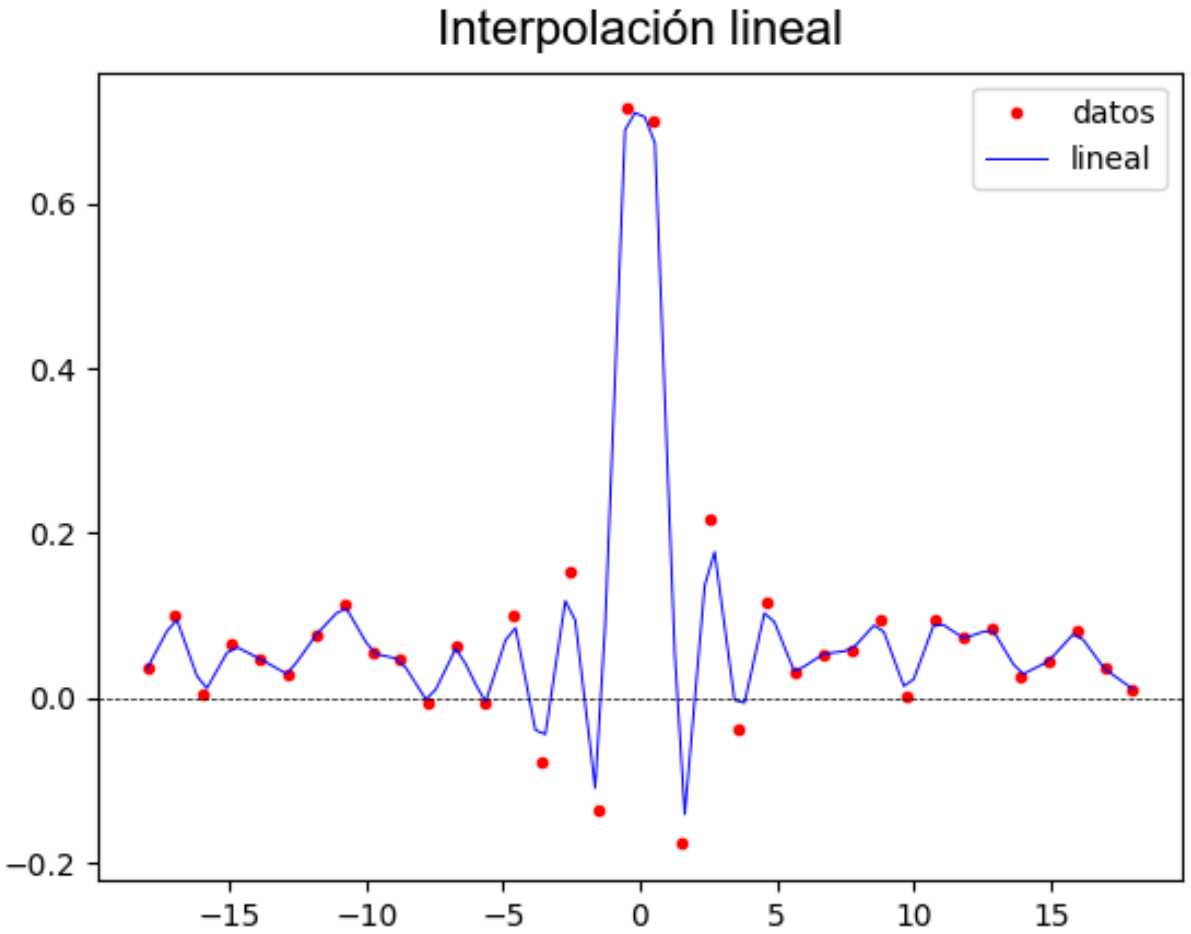


Figura 5: *La función no pasa por todos los puntos aleatorios.*

Funciones lineal y cúbica

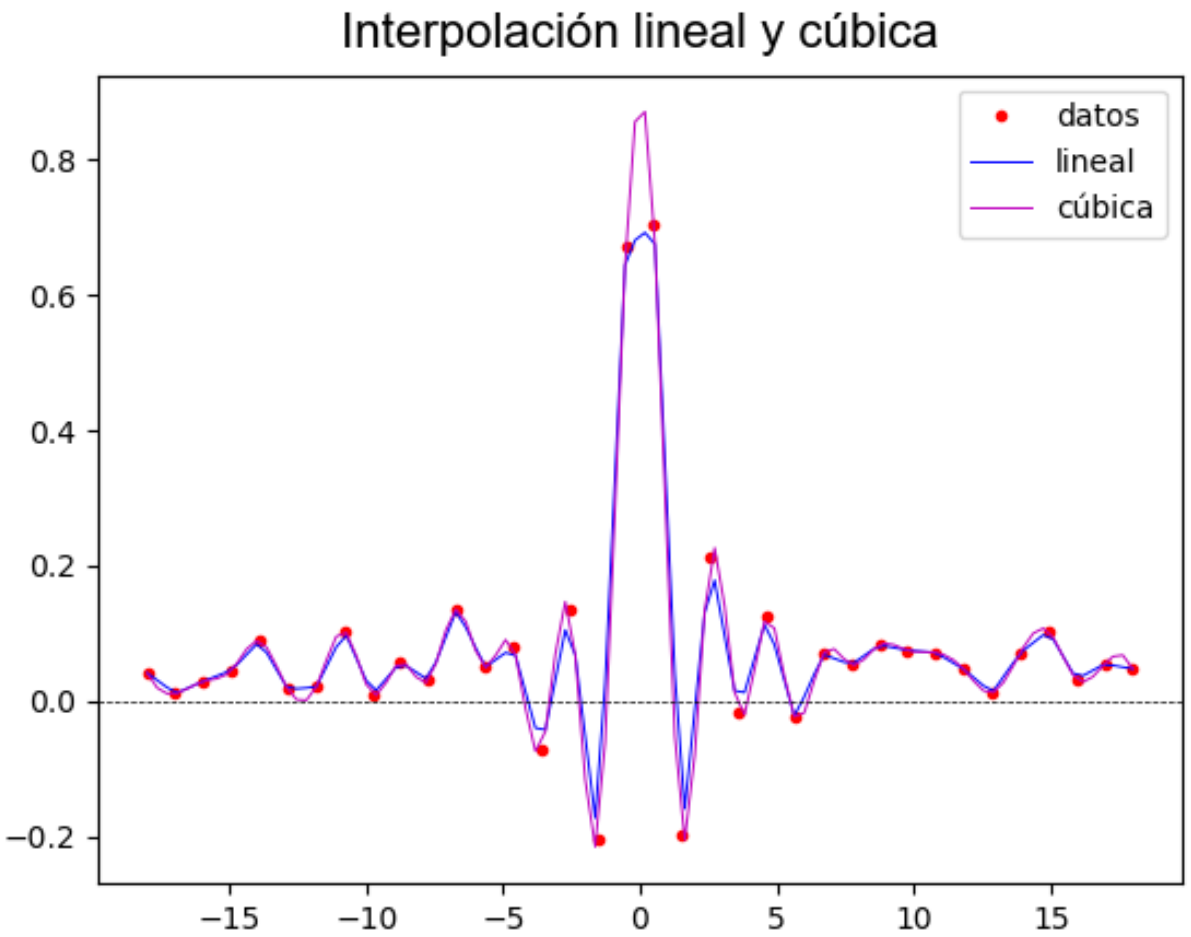


Figura 6: *Se mejora la aproximación con la interp. cúbica.*

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.

5. Las funciones de interpolación pueden oscilar drásticamente entre los puntos de datos.

Más sobre la interpolación.

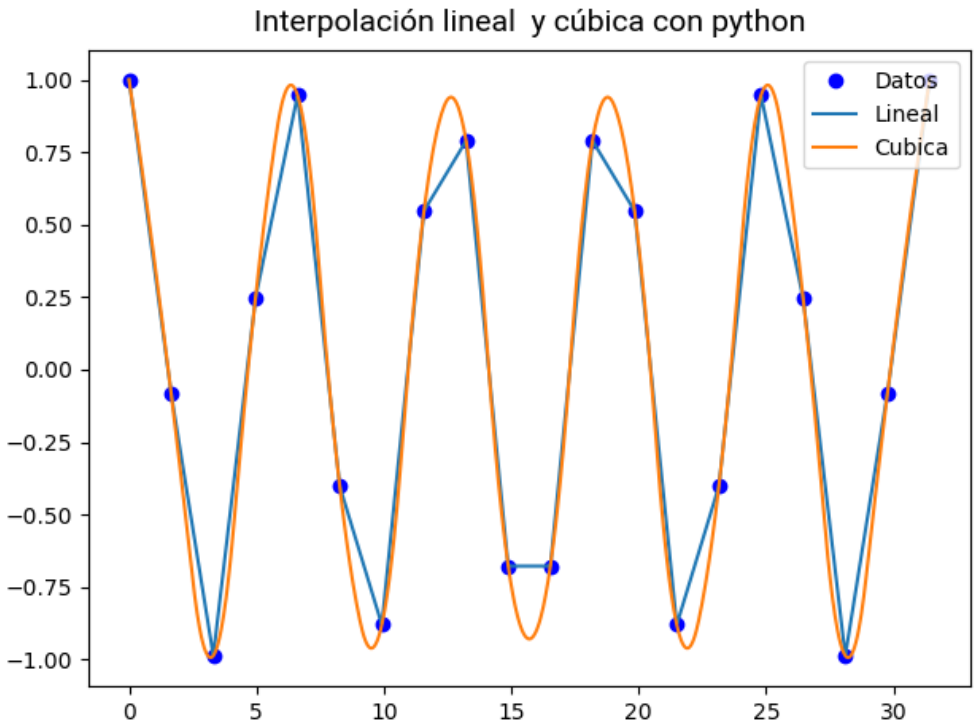
5. Las funciones de interpolación pueden oscilar drásticamente entre los puntos de datos.
6. El ajuste empeora hacia los extremos del conjunto de datos.

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

El “spline cúbico” es el caballo de batalla en este terreno.

Como se puede ver en la siguiente figura, proporciona una curva suave que parece ajustarse bien a los datos.

Curva con un ajuste suave



La suavidad de la curva

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 de la curva

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.

Precaución con las funciones de interpolación

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.

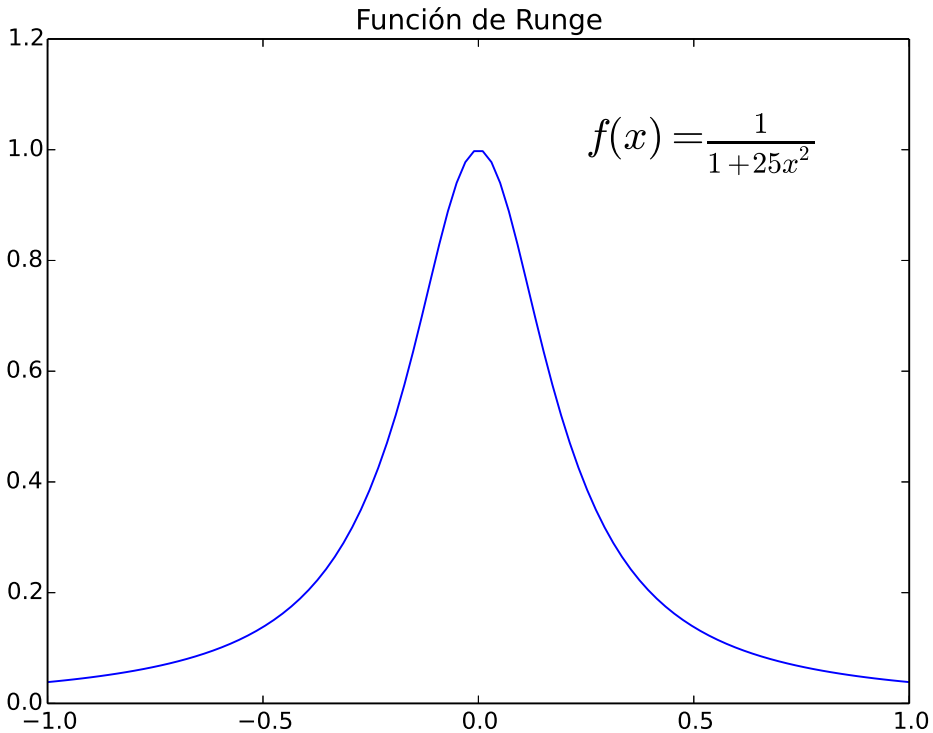
Fenónemo de Runge

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.

Gráfica de la función $f(x)$

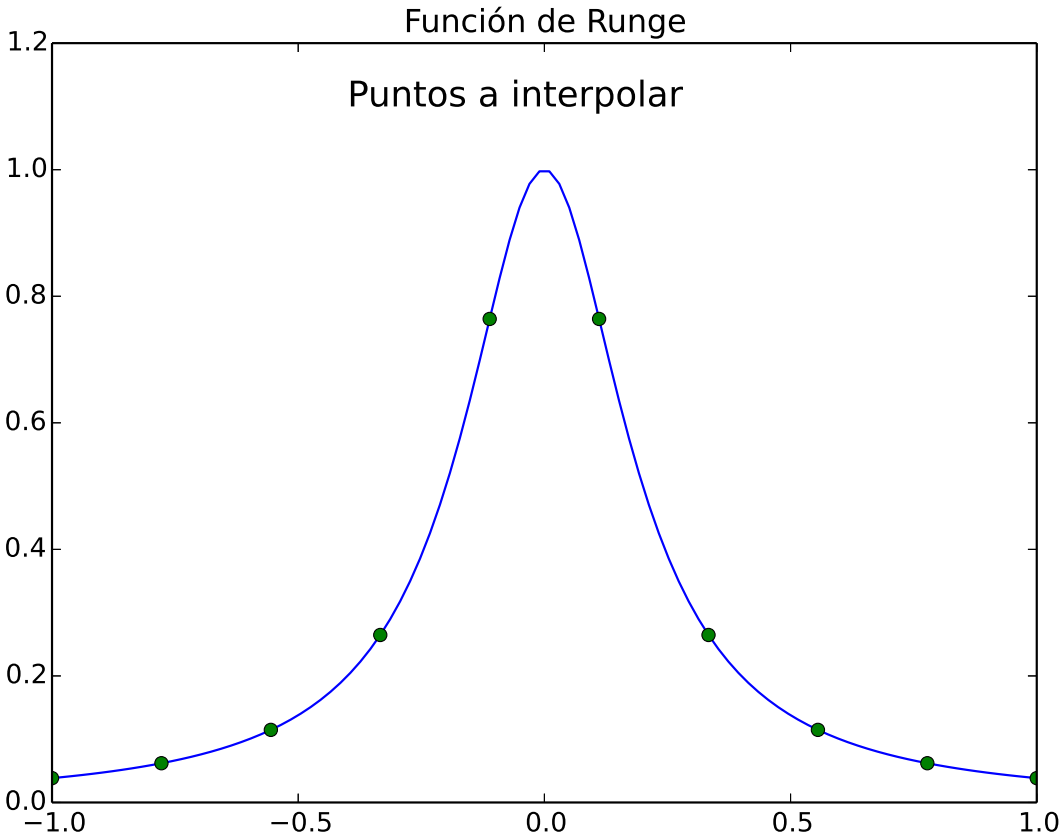
Veamos el siguiente ejemplo: sea la función:

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



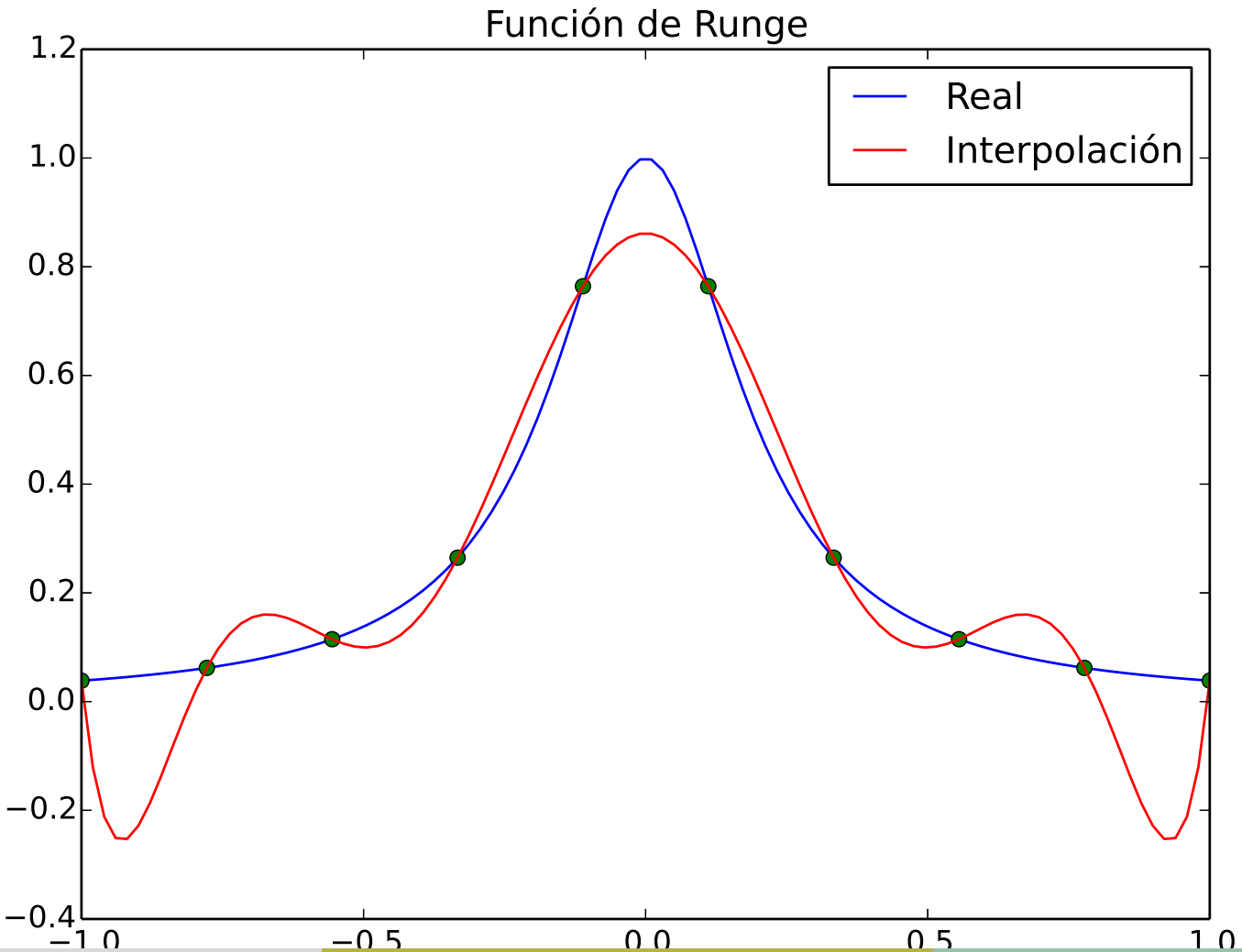
Elección de puntos para interpolar

Elegimos un conjunto aleatorio de puntos a los que vamos a 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.

¿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.

¿Qué es un spline?

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. **Bonitos:** La teoría matemática en que se basan es muy simple y a la vez elegante.

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. **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.

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.

Manejando splines con `python`

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 funciones para el uso de splines con `python`contenidos en la libería: **`scipy.interpolate`**, las cuales son:

La función `splrep`

`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 $x_b \leq x \leq x_e$.

En caso de que no se proporcione el intervalo, se toma como tal, los valores $x[0]$ y $x[-1]$

Argumentos de la función `splrep`

Tomamos la función de la librería **`scipy.interpolate`**, la función opera de la siguiente manera:

```
splrep(x, y, xb=None, xe=None, k=3 [, otros argumentos])
```

donde k es el grado de ajuste del spline. Se recomienda utilizar splines cúbicos.

Lo que devuelve `splrep`

La función devuelve lo siguiente:

`tck` : una tupla.

La tupla `(t,c,k)` contiene el vector de puntos (knots), los coeficientes del B-spline, y el grado del spline.

La función `splev`

`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.

Argumentos de la función

La función opera con los siguientes argumentos (revisa la documentación para obtener más información de los argumentos opcionales)

```
splev (x, tck)
```

Argumentos de la función

splev (x, tck)

donde:

x : Es un arreglo de puntos en donde se evalúa el spline suavizado o sus derivadas.

tck : Es una tupla de 3 elementos o un objeto B-Spline. Si es la tupla, debe de ser la secuencia que devuelve **splrep**, que contiene los nodos, los coeficientes y el grado del spline.

Lo que devuelve `splev`

`y` : un n-arreglo o lista de n-arreglos.

Es un arreglo de valores que representan la función spline evaluada en los puntos x. Si tck se proporcionó a partir de la función **`splrep`**, entonces es la lista de arreglos que representan la curva en el n-espacio dimensional.

La función de Runge y los splines

Usaremos las dos funciones mencionada para evaluar splines cúbicos como aproximación para la función Runge.

Se cambia el número de puntos n como intervalo para la aproximación.

Código I

Código 10: Código completo

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

Código II

```
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, label='Funcion Runge'
       )
21 plt.plot(x, ys8,'+g-', label='n=8')
22 plt.plot(x, ys12,'+r-',label ='n=12'
       )
```

Código III

```
23 plt.legend(loc='best')
24 plt.title('Interpolacion con splines
       cubicos')
25 plt.ylim(-0.2, 1.2)
26 plt.show()
```

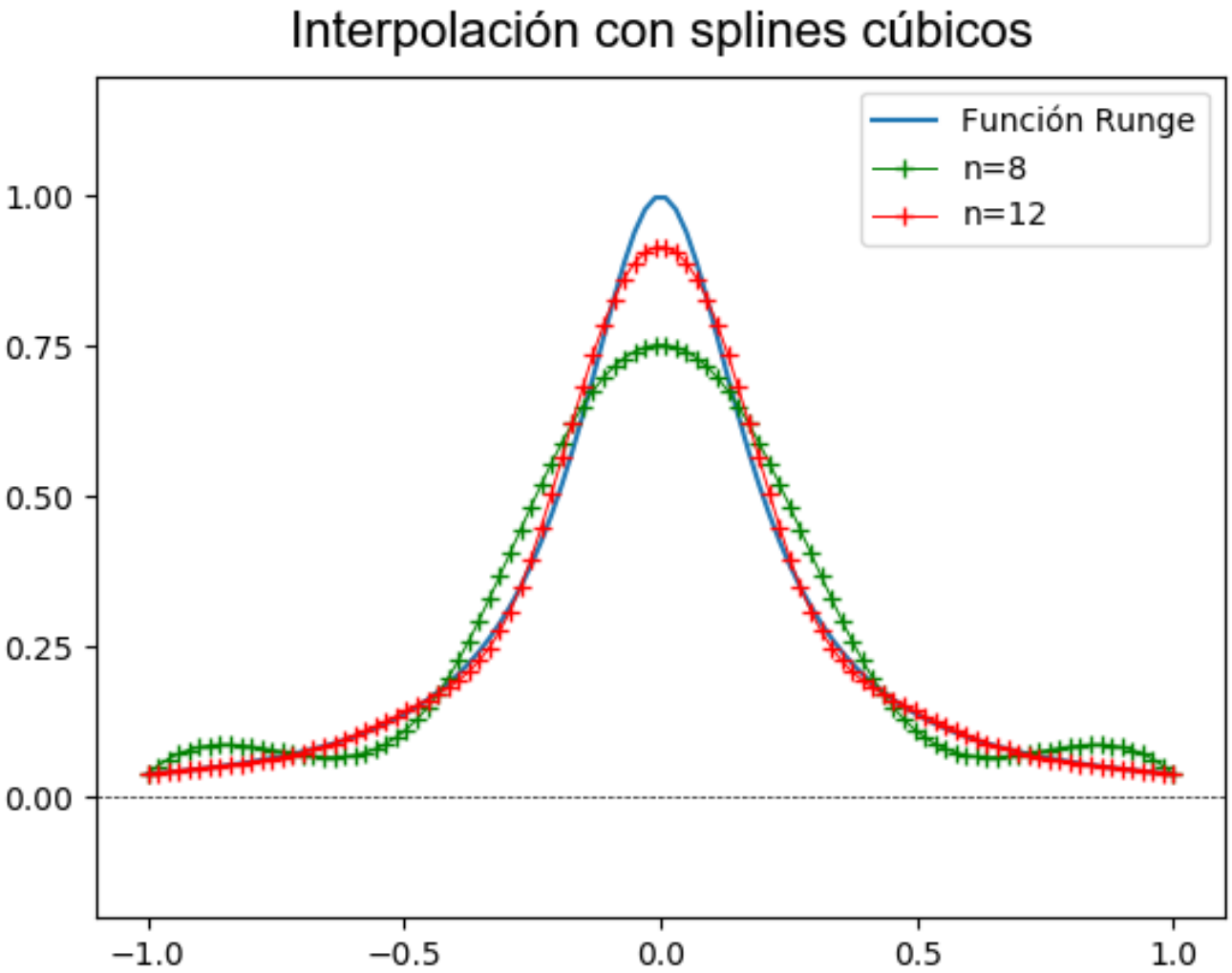


Figura 7: *Al aumentar el número de puntos en el arreglo para la función* ***splrep****, mejora la aproximación con el spline cúbico.*