Universidad Autónoma de Querétaro

Facultad de Ingeniería

Interpolación mediante trazadores

Análisis numérico

Autor:
David Gómez Torres

21 Octubre del 2021

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

1.1. Interpolación mediante trazadores (splines)

Un procedimiento alternativo a la interpolación de polinomios consiste en colocar polinomios de grado inferior en subconjuntos de los puntos asociados con datos. Tales polinomios conectores se denominan **trazadores o splines**.

La Figura (1.1) ilustra una situación donde un trazador se comporta mejor que un polinomio de grado superior. Éste es el caso donde una función en general es suave, pero presenta un cambio abrupto en algún lugar de la región de interés. El polinomio de grado superior tiende a formar una curva de oscilaciones bruscas en la vecindad de un cambio súbito. En contraste, el trazador también une los puntos; pero como está limitado a cambios de tercer grado, las oscilaciones son mínimas.

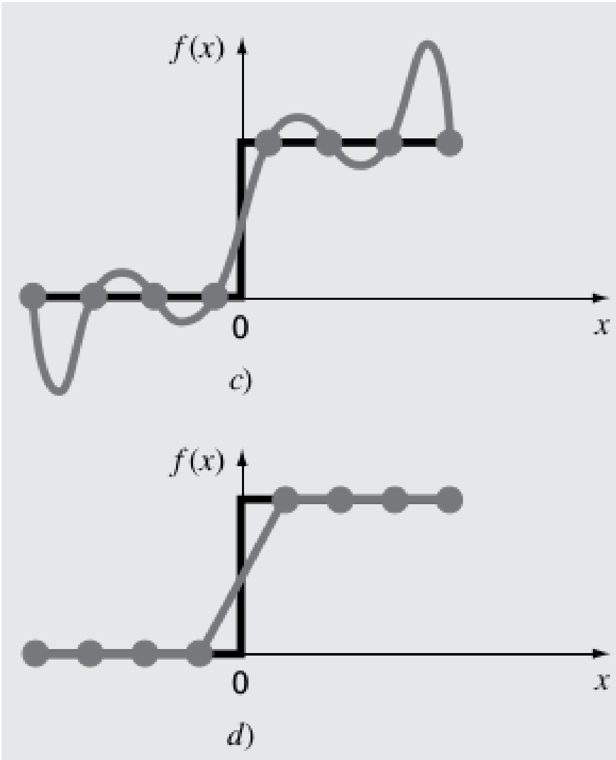

Figura 1.1: Diferencia entre un polinomio y un trazador

El concepto de **trazador** se originó en la técnica de dibujo que usa una cinta delgada y flexible (llamada **spline**, en inglés), para dibujar curvas suaves a través de un conjunto de puntos. El proceso se representa en la figura 18.15 para una serie de cinco alfileres (*puntos asociados con datos*). En esta técnica, el dibujante coloca un papel sobre una mesa de madera y coloca alfileres o clavos en el papel (y la mesa) en la ubicación de los puntos asociados con datos. Una curva cúbica suave resulta al entrelazar la cinta entre los alfileres. De aquí que se haya adoptado el nombre de "**trazador cúbico**" (en inglés: "*cubic spline*") para los polinomios de este tipo.

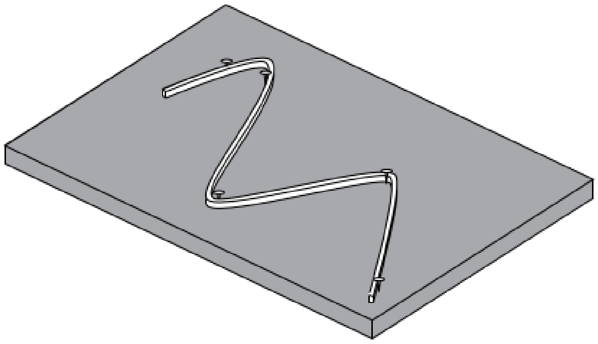

Figura 1.2: Técnica de dibujo para dibujar curvas

1.1.1. Trazadores lineales

Los trazadores de primer grado para un grupo de puntos asociados con datos ordenados pueden definirse como un conjunto de funciones lineales,

$$\begin{aligned}
f(x) &= f(x_0) + m_0(x - x_0) & x_0 \leq x \leq x_1 \\
f(x) &= f(x_1) + m_1(x - x_1) & x_1 \leq x \leq x_2 \\
&. \\
&. \\
f(x) &= f(x_{n-1}) + m_{n-1}(x - x_{n-1}) & x_{n-1} \leq x \leq x_n
\end{aligned} \tag{1.1}$$

donde m_i es la pendiente de la línea recta que une los puntos:

$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{1.2}$$

Estas ecuaciones se pueden usar para evaluar la función en cualquier punto entre x_0 y x_n localizando primero el intervalo dentro del cual está el punto. Después se usa la ecuación adecuada para determinar el valor de la función dentro del intervalo.

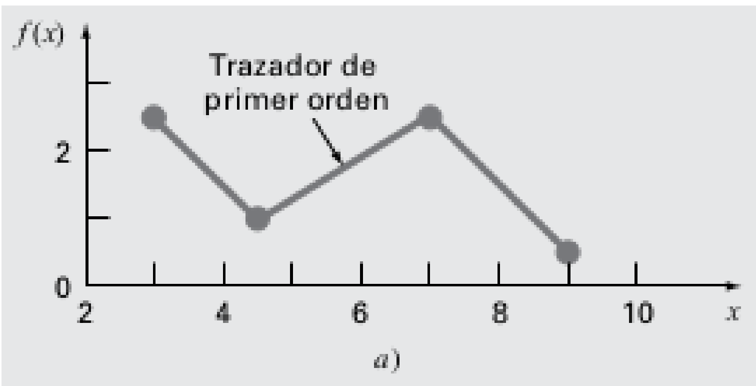

Figura 1.3: Trazador lineal

En los puntos asociados con datos donde se encuentran dos trazadores (llamado **nodo**), la pendiente cambia de forma abrupta. formalmente, la primer derivada de la función es discontinua en esos puntos. Esta deficiencia se resuelve usando trazadores polinomiales de grado superior, que aseguren suavidad en los nodos al igualar las derivadas en esos puntos.

1.1.2. Trazadores cuadráticos

El objetivo de los trazadores cuadráticos es obtener un polinomio de segundo grado para cada intervalo entre los puntos asociados con datos, que se puede representar como

$$f_i(x) = a_i x^2 + b_i x + c_i \tag{1.3}$$

Se requieren $3n$ ecuaciones o condiciones para evaluar las incógnitas.

1. Los valores de la función de polinomios adyacentes deben ser iguales en los nodos interiores.

$$a_{i-1}x_{i-1}^2 + b_{i-1}x_{i-1} + c_{i-1} = f(x_{i-1})$$

para $i = 2$ a n.

2. La primera y la última función deben pasar a través de los puntos extremos.

$$a_1 x_0^2 + b_1 x_0 + c_1 = f(x_0)$$

en total tenemos $2n–2 + 2 = 2n$ condiciones.

3. Las primeras derivadas en los nodos interiores deben ser iguales.

$$2a_{i-1}x_{i-1} + b_{i-1} = 2a_i x_{i-1} + b_i$$

para $i = 2$ a n.

4. Suponga que en el primer punto la segunda derivada es cero.

$$a_1 = 0$$

La interpretación visual de esta condición es que los dos primeros puntos se unirán con una línea recta.

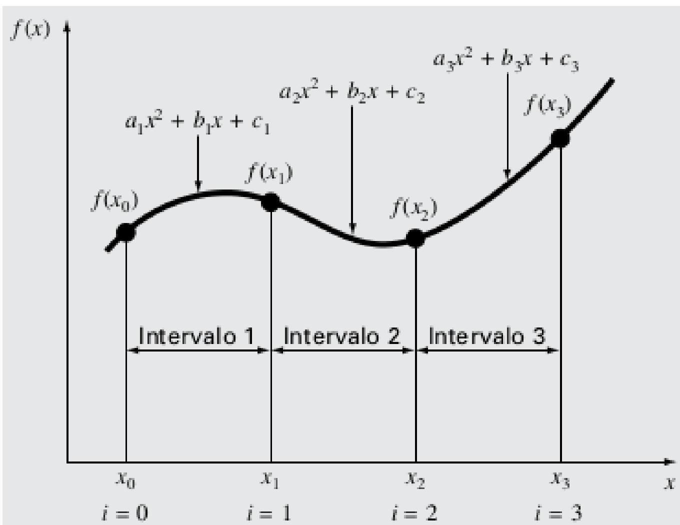

Figura 1.4: Intervalos de trazadores cuadráticos

1.1.3. Trazadores cúbicos

El objetivo en los trazadores cúbicos es obtener un polinomio de tercer grado para cada intervalo entre los nodos:

$$a_i x^3 + b_i x^2 + c_i x + d_i \tag{1.4}$$

Como con los trazadores cuadráticos, se requieren 4n condiciones para evaluar las incógnitas. Éstas son:

1. Los valores de la función deben ser iguales en los nodos interiores ($2n$–2 condiciones).
2. La primera y última función deben pasar a través de los puntos extremos (2 condiciones).
3. Las primeras derivadas en los nodos interiores deben ser iguales (n–1 condiciones).
4. Las segundas derivadas en los nodos interiores deben ser iguales (n–1 condiciones).
5. Las segundas derivadas en los nodos extremos son cero (2 condiciones).

   La deducción matemática para el trazador cúbico está dada por

$$\begin{aligned} f(x) &= \frac{f_i''(x_{i-1})}{6(x_i - x_{i-1})}(x_i - x)^3 + \frac{f_i''(x_i)}{6(x_i - x_{i-1})}(x - x_{i-1})^3 \\ &\quad + \left[\frac{f(x_{i-1})}{x_i - x_{i-1}} - \frac{f''(x_{i-1})(x_i)(x_i - x_{i-1})}{6}\right](x_i - x) \\ &= \left[\frac{f(x)}{x_i - x_{i-1}} - \frac{f''(x_i)(x_i - x_{i-1})}{6}\right](x - x_{i-1}) \end{aligned} \tag{1.5}$$

1.1.4. Algoritmo para trazadores cúbicos

Algorithm 1 Función Spline

Entrada: array x, y; entero n, xu,yu,dy,d2y;
Salida: array yint2;
  f_1 = 2 * (x_2 - x_0)
  g_1 = (x_2 - x_1)
  r_1 = 6/(x_2-x_1 * (y_2-y_1))
  for i=2, n-2 **do**
    e_i = (x_i - x_{i-1})
    f_i = 2 * (x_{i-1} - x_{i-1})
    g_i = (x_{i-1} - x_i)
    r_i = 6/(x_{i-1} - x_i) * (y_{i-1} - y_i)
    r_i = r_i+6/(x_i - x_{i-1}) * (y_{n-1} - y_i)
  end for
  e_{n-1} = (x_{n-1} - x_{n-2})
  f_{n-1} = 2 * (x_n - x_{n-2})
  r_{n-1} = 6/(x_n - x_{n-1}) * (y_n - y_{n-1})
  r_{n-1} = r_{n-1} + 6/(x_{n-1} - x_{n-2}) * (y_{n-2} - y_{n-1})

CAPÍTULO 2

METODOLOGÍA

2.1. Trazador cuadrático

Problema 18.13

Dados los datos

x	1.6	2	2.5	3.2	4	4.5
f(x)	2	8	14	15	8	2

Desarrolle trazadores cuadráticos para los cinco primeros puntos asociados con datos del problema, y pronostique f(3.4) y f(2.2).

1. Gráfica del ajuste por trazador a los datos

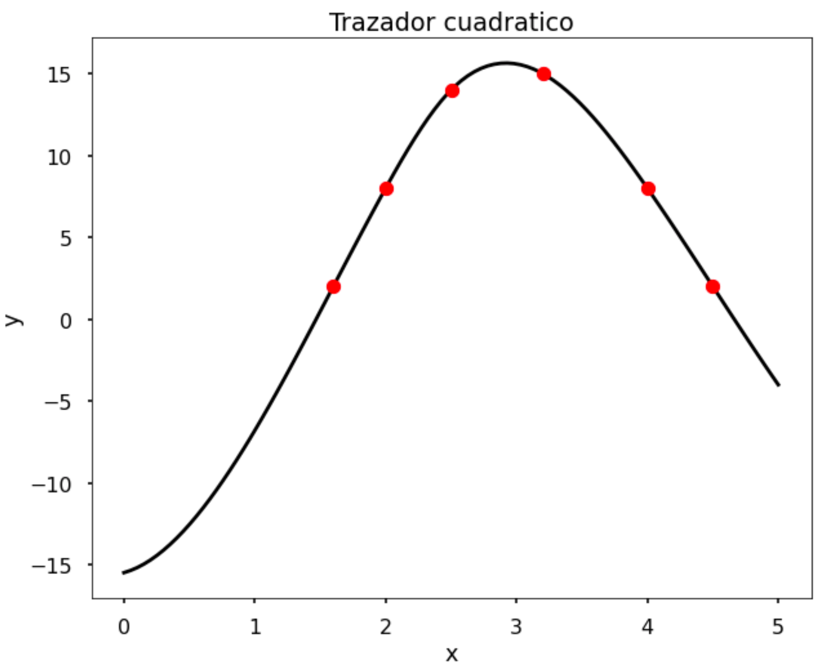

Figura 2.1: Splin cuadrático

a) La función evaluada en 3.4 es: 13.84787003360803

b) La función evaluada en 2.2 es: 10.784681414299527

2.
```
import numpy as np
from math import sqrt

def cubic_interp1d(x0, x, y):
    x = [1, 2, 3,5,7,8]
    y = [2, 6, 19, 99, 291, 444]

    if np.any(np.diff(x) < 0):
        indexes = np.argsort(x)
        x = x[indexes]
        y = y[indexes]

    size = len(x)

    xdiff = np.diff(x)
    ydiff = np.diff(y)

    # allocate buffer matrices
    Li = np.empty(size)
    Li_1 = np.empty(size-1)
    z = np.empty(size)

    # fill diagonals Li and Li-1 and solve [L][y] = [B]
    Li[0] = sqrt(2*xdiff[0])
    Li_1[0] = 0.0
    B0 = 0.0 # natural boundary
    z[0] = B0 / Li[0]

    for i in range(1, size-1, 1):
        Li_1[i] = xdiff[i-1] / Li[i-1]
        Li[i] = sqrt(2*(xdiff[i-1]+xdiff[i]) - Li_1[i-1]
        * Li_1[i-1])
        Bi = 6*(ydiff[i]/xdiff[i] - ydiff[i-1]/xdiff[i-1])
        z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    i = size - 1
    Li_1[i-1] = xdiff[-1] / Li[i-1]
    Li[i] = sqrt(2*xdiff[-1] -
    Li_1[i-1] * Li_1[i-1])
    Bi = 0.0 # natural boundary
    z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    # solve [L.T][x] = [y]
    i = size-1
    z[i] = z[i] / Li[i]
    for i in range(size-2, -1, -1):
        z[i] = (z[i] - Li_1[i-1]*z[i+1])/Li[i]
```

```python

    # find index
    index = x.searchsorted(x0)
    np.clip(index, 1, size-1, index)

    xi1, xi0 = x[index], x[index-1]
    yi1, yi0 = y[index], y[index-1]
    zi1, zi0 = z[index], z[index-1]
    hi1 = xi1 - xi0

    # calculate cubic
    f0 = zi0/(6*hi1)*(xi1-x0)**3 + \
         zi1/(6*hi1)*(x0-xi0)**3 + \
         (yi1/hi1 - zi1*hi1/6)*(x0-xi0) + \
         (yi0/hi1 - zi0*hi1/6)*(xi1-x0)
    return f0

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    x = np.linspace(0, 10, 11)
    y = np.sin(x)
    plt.scatter(x, y)

    x_new = np.linspace(0, 10, 201)
    plt.plot(x_new, cubic_interp1d(x_new, x, y))

    plt.show()
```

2.2. Trazador cúbico

Problema 18.14

Obtenga trazadores cúbicos para los datos del problema 18.6, y a) pronostique $f(4)$ y $f(2.5)$, y b) verifique que $f_2(3)$ y $f_3(3) = 19$.

x	1	2	3	5	7	8
f(x)	2	6	19	99	291	444

1. Gráfica

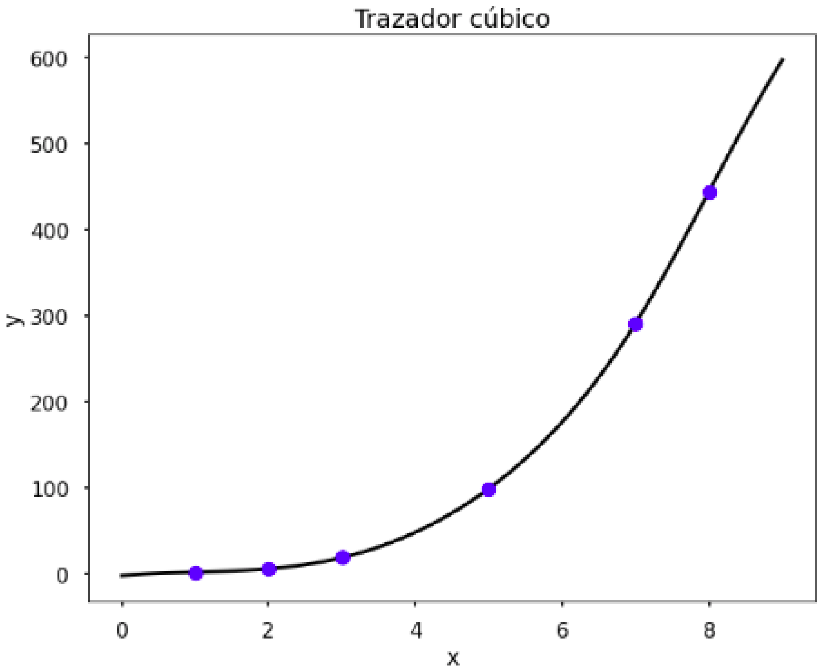

Figura 2.2: Trazador cúbico

2.

```python
import numpy as np
from math import sqrt

def cubic_interp1d(x0, x, y):
    x = [1, 2, 3,5,7,8]
    y = [2, 6, 19, 99, 291, 444]

    if np.any(np.diff(x) < 0):
        indexes = np.argsort(x)
        x = x[indexes]
        y = y[indexes]

    size = len(x)

    xdiff = np.diff(x)
    ydiff = np.diff(y)

    # allocate buffer matrices
    Li = np.empty(size)
    Li_1 = np.empty(size-1)
    z = np.empty(size)

    # fill diagonals Li and Li-1 and solve [L][y] = [B]
    Li[0] = sqrt(2*xdiff[0])
    Li_1[0] = 0.0
    B0 = 0.0 # natural boundary
    z[0] = B0 / Li[0]

    for i in range(1, size-1, 1):
        Li_1[i] = xdiff[i-1] / Li[i-1]
        Li[i] = sqrt(2*(xdiff[i-1]+xdiff[i]) - Li_1[i-1] * Li_1[i-1])
        Bi = 6*(ydiff[i]/xdiff[i] - ydiff[i-1]/xdiff[i-1])
        z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    i = size - 1
    Li_1[i-1] = xdiff[-1] / Li[i-1]
    Li[i] = sqrt(2*xdiff[-1] - Li_1[i-1] * Li_1[i-1])
    Bi = 0.0 # natural boundary
    z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    # solve [L.T][x] = [y]
    i = size-1
    z[i] = z[i] / Li[i]
    for i in range(size-2, -1, -1):
        z[i] = (z[i] - Li_1[i-1]*z[i+1])/Li[i]

```

```
    # find index
    index = x.searchsorted(x0)
    np.clip(index, 1, size-1, index)

    xi1, xi0 = x[index], x[index-1]
    yi1, yi0 = y[index], y[index-1]
    zi1, zi0 = z[index], z[index-1]
    hi1 = xi1 - xi0

    # calculate cubic
    f0 = zi0/(6*hi1)*(xi1-x0)**3 + \
         zi1/(6*hi1)*(x0-xi0)**3 + \
         (yi1/hi1 - zi1*hi1/6)*(x0-xi0) + \
         (yi0/hi1 - zi0*hi1/6)*(xi1-x0)
    return f0

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    x = np.linspace(0, 10, 11)
    y = np.sin(x)
    plt.scatter(x, y)

    x_new = np.linspace(0, 10, 201)
    plt.plot(x_new, cubic_interp1d(x_new, x, y))

    plt.show()
```

2.3. Implementación del splin cúbico

Ejercicio 18.23

Desarrolle, depure y pruebe un programa en cualquier lenguaje de alto nivel o de macros de su elección, para implantar la interpolación con splin cúbico con base en la figura 18.18. Pruebe el programa con la repetición del ejemplo 18.10.

- Gráfica del splin cúbico

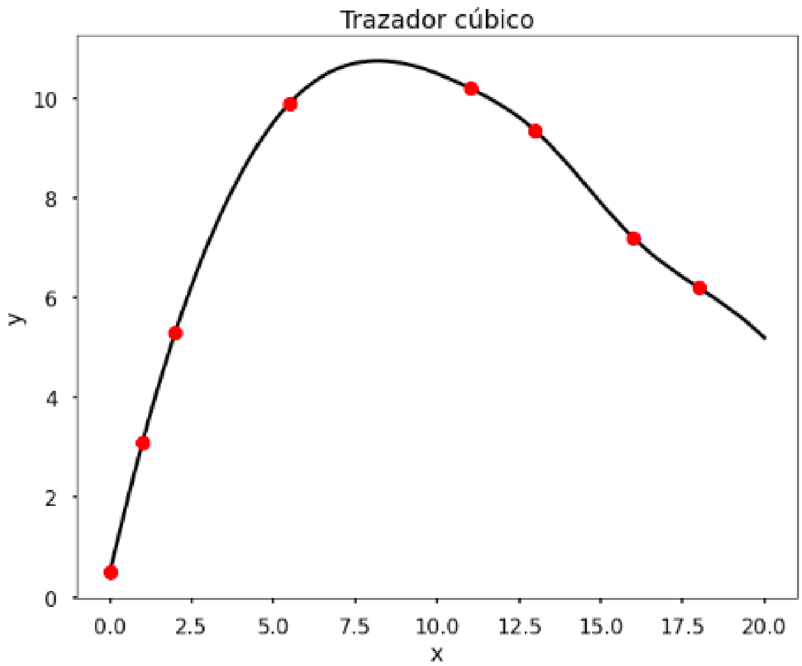

Figura 2.3: Splin cúbico

```
import numpy as np
from math import sqrt

def cubic_interp1d(x0, x, y):
    x = [1, 2, 3,5,7,8]
    y = [2, 6, 19, 99, 291, 444]

    if np.any(np.diff(x) < 0):
        indexes = np.argsort(x)
        x = x[indexes]
        y = y[indexes]

    size = len(x)

    xdiff = np.diff(x)
    ydiff = np.diff(y)

    # allocate buffer matrices
    Li = np.empty(size)
    Li_1 = np.empty(size-1)
    z = np.empty(size)

    # fill diagonals Li and Li-1 and solve [L][y] = [B]
    Li[0] = sqrt(2*xdiff[0])
    Li_1[0] = 0.0
    B0 = 0.0 # natural boundary
    z[0] = B0 / Li[0]

    for i in range(1, size-1, 1):
        Li_1[i] = xdiff[i-1] / Li[i-1]
        Li[i] = sqrt(2*(xdiff[i-1]+xdiff[i]) - Li_1[i-1] * Li_1[i-1])
        Bi = 6*(ydiff[i]/xdiff[i] - ydiff[i-1]/xdiff[i-1])
        z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    i = size - 1
    Li_1[i-1] = xdiff[-1] / Li[i-1]
    Li[i] = sqrt(2*xdiff[-1] - Li_1[i-1] * Li_1[i-1])
    Bi = 0.0 # natural boundary
    z[i] = (Bi - Li_1[i-1]*z[i-1])/Li[i]

    # solve [L.T][x] = [y]
    i = size-1
    z[i] = z[i] / Li[i]
    for i in range(size-2, -1, -1):
        z[i] = (z[i] - Li_1[i-1]*z[i+1])/Li[i]

    # find index
```

```
    index = x.searchsorted(x0)
    np.clip(index, 1, size-1, index)

    xi1, xi0 = x[index], x[index-1]
    yi1, yi0 = y[index], y[index-1]
    zi1, zi0 = z[index], z[index-1]
    hi1 = xi1 - xi0

    # calculate cubic
    f0 = zi0/(6*hi1)*(xi1-x0)**3 + \
         zi1/(6*hi1)*(x0-xi0)**3 + \
         (yi1/hi1 - zi1*hi1/6)*(x0-xi0) + \
         (yi0/hi1 - zi0*hi1/6)*(xi1-x0)
    return f0

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    x = np.linspace(0, 10, 11)
    y = np.sin(x)
    plt.scatter(x, y)

    x_new = np.linspace(0, 10, 201)
    plt.plot(x_new, cubic_interp1d(x_new, x, y))

    plt.show()
```

2.4. Splin cúbico

Ejercicio 18.24

Emplee el software desarrollado en el problema 18.23 para ajustar trazadores cúbicos para los datos de los problemas 18.5 y 18.6. Para ambos casos, pronostique f(2.25).

- Gráfica del ajuste por trazador a los datos del Problema 18.5

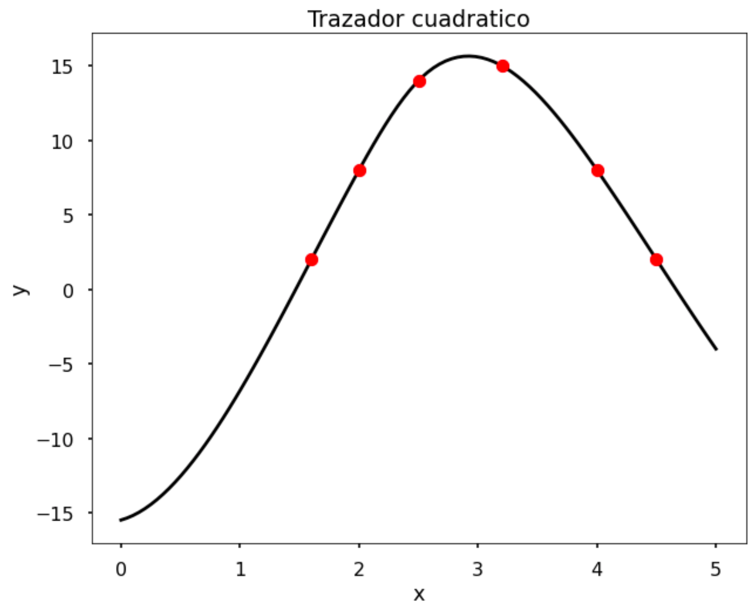

Figura 2.4: Splin cúbico

- Gráfica del problema 18.6

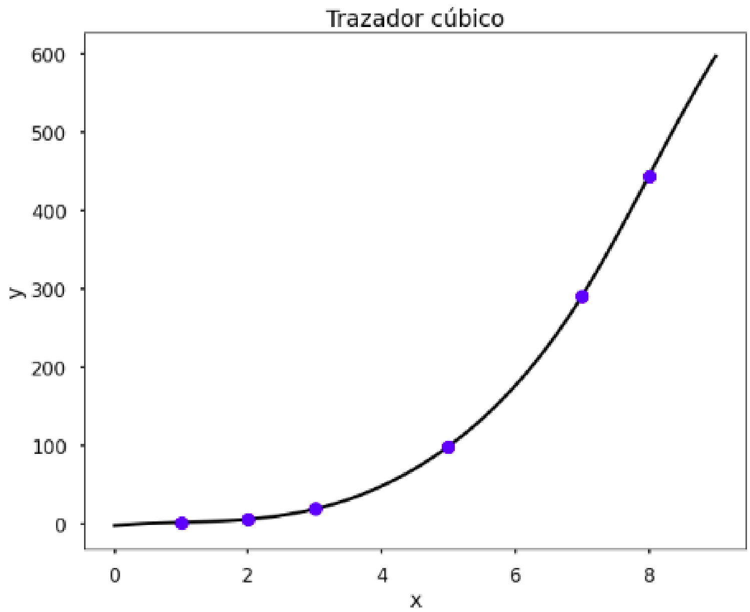

Figura 2.5: Trazador cúbico

Los códigos son los mismos para los Problemas 18.3 y 18.4

CAPÍTULO 3

BIBLIOGRAFÍA

1. Chapra, S. C., Canale, R. P. (2011). *Numerical methods for engineers* (Vol. 1221). New York: Mcgraw-hill.