Resumen

1. Introducción

Supongamos que conocemos N+1 puntos (x_0, y_0), (x_1, y_1), ..., (x_N, y_N), donde las x se distribuyen en un intervalo [a, b], y nos interesan puntos dentro del intervalo, de los cuales no sabemos su valor en y. En estos casos, se pueden utilizar métodos de interpolación, que usan la información de los pares de puntos conocidos, para encontrar los valores de interés. Si nos interesa aproximar la información a una función arbitraria, se pueden utilizar aproximación fragmentaria, que construye un polinomio diferente en cada subintervalo, un método que utiliza este procedimiento son los splines.

2. Metodología

2.1. Splines continuidad 0

La forma más simple de unir puntos por medio de un polinomio, es que este sea de grado 1, es decir, unir una recta entre los dos puntos. La forma de construcción es calcular la ecuación de la recta que pasa por los dos puntos:

$$y = \left(\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}\right)(x - x_i) + y_i \tag{1}$$

Algorithm 1 Spline - recta

Entrada: Intervalo [a, b]; int size: cantidad de puntos conocidos; arreglos: puntos x, puntos y; int sizeIntervalo: cantidad de puntos para calcular la recta; puntos para malla: double X[][], double Y[][]
Salida: Malla de puntos

```
function ECUACIONRECTA((x, x1, y1, x2, y2))
    y = ((y2-y1)/(x2-x1))*(x-x1)+y1
    return: y
for (i=0; i<cant puntos; i++) do
    h = (x[i]-x[i+1])/sizeIntervalo
    for (j=0; j<)sizeIntervalo; j++ do
        X[i][j] = x[i] + h*j;
        Y[i][j] = EcuacionRecta(X[i][j], x[i], y[i], x[i+1], y[i+1])
return: X, Y
```

El objetivo de hacer una malla de puntos es la graficación.

2.2. Splines de continuidad 1

Un spline de continuidad 1 es aquel cuyos datos consecutivos se ajustan con un polinomio cuadrático [1]. Sea $S_i(x)$ el ajuste del spline,

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2, \tag{2}$$

Si tenemos n puntos, se pondrán hacer n-1 parábolas que se ajusten entre los puntos, por ejemplo, si se tienen tres puntos, se tendrán dos ecuaciones:

$$[x_1, x_2] : S_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2,$$
$$[x_2, x_3] : S_2(x) = a_2 + b_2(x - x_2) + c_2(x - x_2)^2,$$

con seis incógnitas, por lo tanto, se necesitan cuatro ecuaciones más, o reducir el número de incógnitas. Si se evalúa $S_i(x_i) = a_i = f(x_i)$, se tendría,

$$S_i(x) = f(x_i) + b_i(x - x_i) + c_i(x - x_i), \tag{3}$$

con lo que se redujo la cantidad de ecuaciones. Además, las primeras derivadas son continuas en los puntos interiores, por lo que:

$$\begin{aligned} &S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \\ &S_i'(x_{i+1}) = b_i + 2c_i(x_{i+1} - x_i), \\ &S_{i+1}'(x_{i+1}) = b_{i+1}, \\ &b_{i+1} = b_i + 2c_i(x_{i+1} - x_i). \end{aligned} \tag{4}$$

Utilizando estas ecuaciones, se construye una matriz de la forma:

$$\begin{pmatrix} 1 & 0 & 0 & & & & & & \\ 0 & (x_1 - x_0) & (x_1 - x_0)^2 & & & & & & \\ 0 & 0 & 0 & 1 & 0 & 0 & & & \\ & & & 0 & (x_2 - x_1) & (x_2 - x_1)^2 & & & \\ & & & & & \ddots & \ddots & & \\ & & & & & 1 & 0 & 0 & \\ & & & & & 0 & (x_n - x_{n-1}) & (x_n - x_{n-1})^2 & \\ 0 & 1 & 2(x_1 - x_0) & 0 & -1 & & & & \\ 0 & 0 & 0 & 0 & 1 & 2(x_2 - x_1) & & & \\ & & & & & & & & \\ 0 & 0 & 1 & \dots & & \dots & \dots & \dots & \end{pmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ a_1 \\ b_1 \\ c_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 - f_1 \\ f_3 - f_2 \\ f_4 - f_3 \\ \vdots \\ \vdots \\ \vdots \\ f_n - f_{n-1} \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix}, \tag{5}$$

que se puede resolver utilizando cualquier método de factorización.

Algorithm 2 Spline - parábola entre puntos

Entrada: int size: cantidad de puntos; arreglos: x, y
Salida: Coeficientes de los polinomios entre puntos

Armar la matriz:
int M = size*3: hay tres coeficientes por buscar para cada parábola
Declarar una matriz vacia de tamaño M*M
Construir la matriz:
int count, bcount, bbcount: constantes para armar la matriz
for i=0; i<M; i++ **do**
    if i%3 == 0 **then**
        matriz[i-count][i] = 1.0
        matriz[i-count+1][i+1] = x[count+1]-x[count]
        matriz[i-count+1][i+2] = (x[count+1]-x[count])^2 count++
    if i>= (size-1)*2 **then**
        matriz[i][bcount+1] = 1.0
        matriz[i][bcount+2] = 2*(x[bbcount+1]-x[bbcount])
        bcount += 3
        bbcount ++
matriz[M-1][2] = 1.0

Construir b
int countb, countbb
for i=0; i<M; i++ **do**
    if i < (N-1)*2 **then**
        if i%2 == 0 **then**
            b[i] = y[countb]
            countb ++
        else
            b[i] = y[countbb+1]-y[countbb]
            countbb ++

2.3. Spline continuidad 2

En este caso, buscamos ajustar un polinomio cúbico entre los puntos.

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \tag{6}$$

y buscamos reducir el número de incógnitas del sistema. Primero, se evalúa:

$$\begin{aligned} &s_i(x_i) = f(x_i) = a_i, \\ &s_i(x_{i+1}) = f(x_{i+1}) = a_i + b_i(x_{i+1} - x_i) + c_i(x - x_i)^2 + d_i(x_{i+1} - x_i)^3, \\ &f(x_{i+1}) = f_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3. \end{aligned} \tag{7}$$

Las primeras derivadas en los puntos interiores deben ser continuas, por lo tanto,

$$\begin{aligned} &s_i(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2, \\ &s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \\ &b_{i+1} = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2. \end{aligned} \tag{8}$$

y en este caso, también las segundas derivadas deben ser iguales:

$$\begin{aligned} &s_i''(x) = 2c_i + 6d_i(x - x_i), \\ &s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \\ &c_{i+1} = c_i + 3d_i(x - x_i). \end{aligned} \tag{9}$$

Sin embargo, se necesitan condiciones extra para que se pueda resolver el sistema, en este caso, se escogió que la segunda derivada del primer y último término fueran iguales a cero. En este caso, el sistema se resolverá para encontrar las constantes c_i, ya que al sustituir términos se encuentran relaciones entre las demás constantes:

$$a_i = f(x_i) \tag{10}$$

$$b_i = \frac{f(x_{i+1} - x_i)}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{3}(2c_i + c_{i+1}) \tag{11}$$

$$d_i = \frac{c_{i+1} - c_i}{3(x_{i+1} - x_i)} \tag{12}$$

de esta forma, se construye una matriz:

$$\begin{pmatrix} 1 & 0 & 0 & & & \\ x_1 - x_0 & 2(x_1 - x_0 + x_2 - x_1) & (x_2 - x_1) & & & \\ 0 & x_2 - x_1 & 2(x_2 - x_1 + x_3 - x_2) & x_3 - x_2 & & \\ 0 & 0 & x_3 - x_2 & 2(x_4 - x_3 + x_3 - x_2) & & \\ & & & & & \\ & & \ddots & \ddots & \ddots & \\ & & x_{n-2} - x_{n-1} & 2(x_{n-2} - x_{n-1} + x_{n-1} - x_n) & x_{n-1} - x_n & \\ & & & & 1 & \end{pmatrix} \tag{13}$$

y el vector b para solucionar el sistema:

$$\begin{bmatrix} 0 \\ 3\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} - \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 3\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} - \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}} \\ 0 \end{bmatrix}. \tag{14}$$

Así, se soluciona el sistema y se encuentran las constantes restantes.

Algorithm 3 Spline - cúbico

Entrada: int M: cantidad de elementos; arreglos: x, y
Salida: constantes de los polinomios cúbicos

```
1: Construir la matriz:
2: matriz[0][0] = 1.0
3: [M-1][M-1] = 1.0
4: for i=1; i<M-1; i++ do
5:     matriz[i][i] = 2*(x[i+1]-x[i-1])
6:     matriz[i][i-1] = x[i]-x[i-1]
7:     matriz[i][i+1] = x[i+1]-x[i]

8: Constuir b
9: b[0] = 0, b[M-1] =
10: for i = 1; i<M-1; i++ do
11:     3*(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-1])/(x[i]-x[i-1])
12: Resolver el sistema de ecuaciones.
13: Encontrar los coeficientes restantes.
```

3. Resultados

Se utilizaron los mismos datos aleatorios para todos los métodos para observar cómo se suavizaba conforme aumentaba el grado del polinomio, así, para la ejecución de los tres splines, el tiempo total fue de:

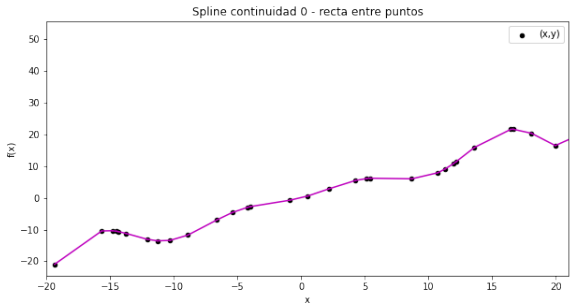
```
zaira@debian:~/Documentos/metodos_numericos/tarea9$ gcc splines_todos.c -lm
zaira@debian:~/Documentos/metodos_numericos/tarea9$ ./a.out

 Segundos: 0.017182
zaira@debian:~/Documentos/metodos_numericos/tarea9$
```

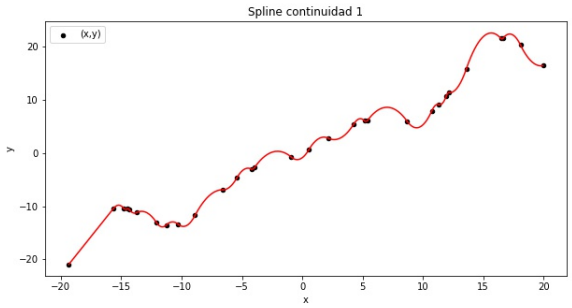
Figura 1: Tiempo total de ejecución

y se utilizó la función:

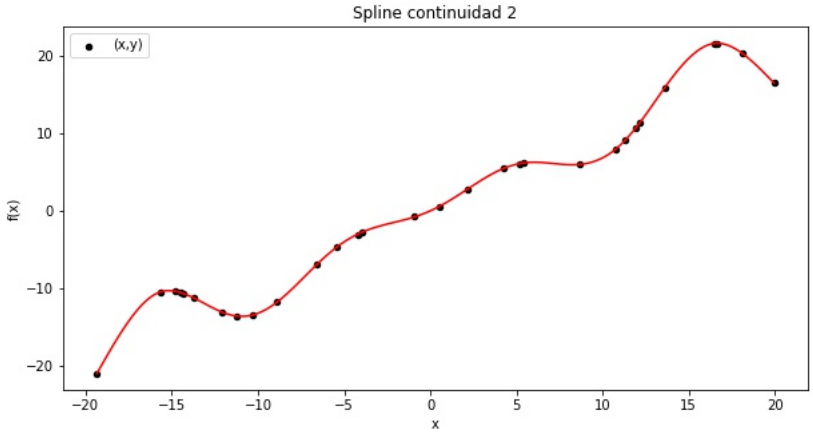
$$f(x) = x + \frac{xsin(\frac{x}{2})}{3}. \tag{15}$$



(a)



(b)



(c)

Figura 2: Comportamiento de la interpolación al aumentar el grado del polinomio.

Además, se compararon los resultados con su solución exacta, y se encontró:

Método	Error promedio (n = 3000)
Spline - recta	0.3682
Spline - parábola	0.5682
Spline - cúbico	0.0119

Cuadro 1: Comparación de métodos

4. Conclusiones

Una de las ventajas de los splines cúbicos, es que se busca solucionar una matriz tridiagonal, que se puede resolver de forma rápida con la factorización de Cholesky, por lo que su costo computacional no es alto. Sin embargo, no siempre es adecuado pasar por todos los puntos. Además, se observa en el Cuadro. 1, que los resultados con el spline cúbico son los mejores, como se esperaba.

Referencias

[1] Curve fitting: Splines - ppt video online download.

Apéndice

En esta sección extra, se presentan los códigos de Python para graficar las rectas.

Spline-recta

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

df = pd.read_csv("splines0_n50_v15.txt", sep=' ', names=['x', 'y'])
X = np.array(pd.read_csv("xsplines0_v15.txt", sep = ' ', header = None).dropna(axis = 1))
Y = np.array(pd.read_csv("ysplines0_v15.txt", sep = ' ', header = None).dropna(axis = 1))

df.plot(figsize = (10, 5), kind='scatter', x = 'x', y = 'y', label = '(x,y)', color = 'black')
#plt.plot(X[0], Y[0], color='m', label = 'Spline - recta')
for i in range(30):
    plt.plot(X[i], Y[i], 'm-')
plt.xlim(-20.0, 21.0)
#plt.ylim(0, 1.5)
#plt.text(4, 40.0, r'$f(x) = x^{3}-5x^{2}+3x+4$', fontsize = 10)
plt.ylabel('f(x)')
plt.legend()
plt.title('Spline continuidad 0 - recta entre puntos')
```

Listing 1: Graficar recta entre puntos

Spline-parábola

```
df = pd.read_csv("xspline1_v15.txt", sep=' ', names=['x', 'y'])
dfCoef = pd.read_csv("xsol_spline1_v15.txt", names = ['xcoef'])
xCoef = np.array(dfCoef['xcoef'])
def fun(x, x1, i):
    ycuadratica = xCoef[i] + (xCoef[i+1])*(x-x1) + (xCoef[i+2])*(x-x1)*(x-x1)#(xCoef[i+1]-x1)*x + (xCoef[i+2]-x1)
    return ycuadratica

df.plot(figsize = (10, 5), kind='scatter', x = 'x', y = 'y', label = '(x,y)', color = 'black')
count = 0
n = len(df)
for i in range(n):
    if ((i+1)<n):
        x = np.linspace(df['x'][i], df['x'][i+1], 50)
        plt.plot(x, fun(x, df['x'][i], count), 'red')
        count += 3

plt.title('Spline continuidad 1')
plt.savefig('spline1_ej1_v1.jpg')
```

Listing 2: Parábola entre puntos

Spline-cúbico

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

df.plot(figsize = (10, 5), kind='scatter', x = 'x', y = 'y', label = '(x,y)', color = 'black')
count = 0
n = len(df)
for i in range(n):
    if ((i+1)<n):
        x = np.linspace(df['x'][i], df['x'][i+1], 50)
        plt.plot(x, fun(x, df['x'][i], count), 'red')
        count += 1
plt.ylabel('f(x)')
```

```
14 plt.title('Spline continuidad 1')
```

Listing 3: Cúbica entre puntos