Regresión y Splines

July 20, 2022

0.1 Taller 5 - Métodos Númericos

Néstor Heli Aponte Ávila - Cod. 20182167052

El taller también esta aquí https://yoriichinara.github.io/Taller5Metodos/

! Trabaje en Jupyter Notebook y me gusta mas como salió en HTML pero el moodle no me adjuntar links ni nada.

1 Regresión

1. Para el conjunto de datos de la tabla, determine la curva de cada familia que mejor se ajusta en el sentido de mínimos cuadrados. Gráfique y ponga título y nombre a los ejes de la gráfica.

x_k	y_k
1	0.6
2	1.9
3	4.3
4	7.6
5	12.6

- $f(x) = Ce^{Ax}$, usando las sustituciones $X = x$, $Y = \ln(y)$, $C = e^B$ para linealizar los datos.

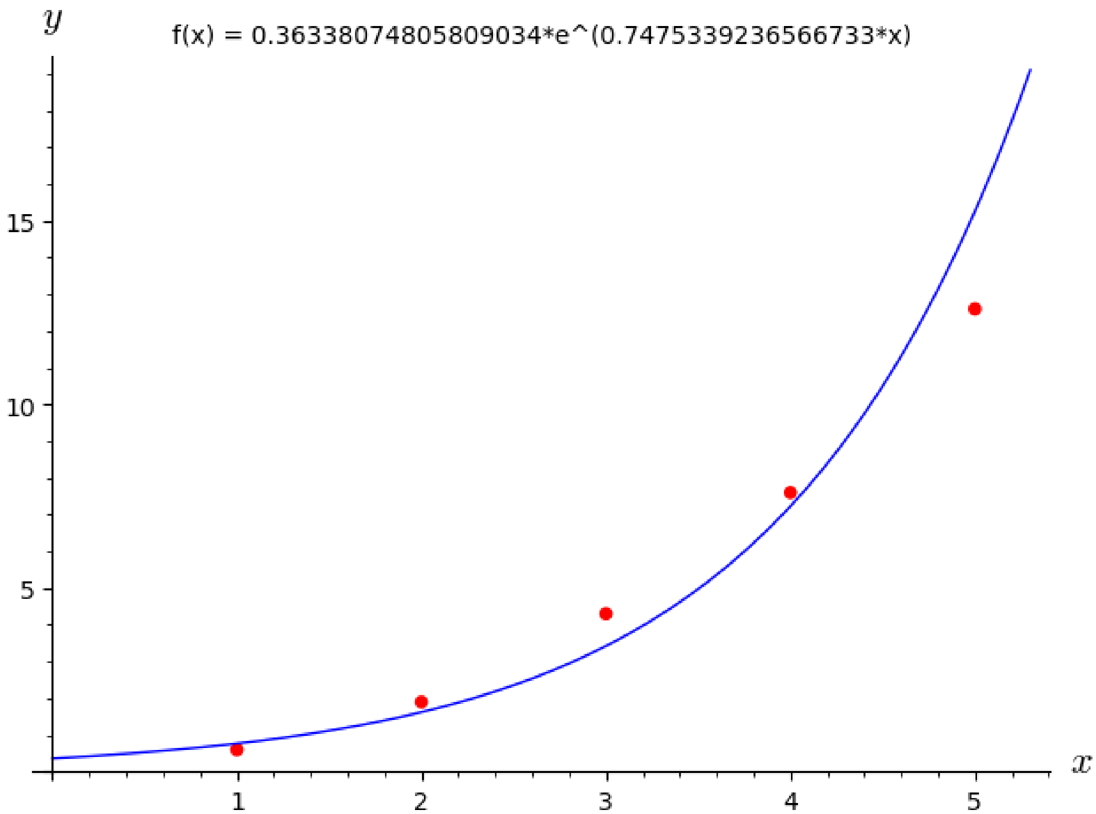
[ ]:
```
# INPUT
x_k = [1, 2, 3, 4, 5] # Abscisas
y_k = [0.6, 1.9, 4.3, 7.6, 12.6] # Ordenadas

def adjexpnolineal(x_k, y_k): # Definición de la función
    N = len(x_k) # len() indica la dimesión del vector -- número de puntos␣
 ↪muestra
    X_k = x_k # Para este caso la variable X = x -- no cambia
    Y_k = [] # Iniciación de la variable que recoja la sustitución de las y_k␣
 ↪chicas

    # Pequeño ciclo for para cálcular el ln -- por alguna razón no me dejo␣
 ↪cálcularlo a todo el vector de una
    for i in range(N): # recorrido
        Y_k.append(0) # .append(0) agrega 0's para poder acceder y editar esa␣
 ↪posición con el ciclo for
```

```
        Y_k[i] = log(y_k[i]) # Cálculo Y_k = ln(y_k)

    # COEFICIENTES PARA LAS ECUACIONES NORMALES DE GAUSS
    sumX_k = sum(X_k) # Sumatoria de las X_k
    sumY_k = sum(Y_k) # Sumatoria de las Y_k (con sustitución a bordo)
    sumX_k2 = vector(X_k) * vector(X_k) # Sumatoria de los X_k^2 -- Hago el␣
 ↪producto escalar pensadoles como vectores
    sumX_kY_k = vector(X_k) * vector(Y_k) # Sumatoria de X_kY_k -- De nuevo␣
 ↪pensandolos como vectores

    # CONSTRUCCIÓN Y SOLUCIÓN DEL SISTEMA DE ECUACIONES NORMALES DE GAUSS
    M1 = matrix(RDF,2,2,[sumX_k2,sumX_k,sumX_k,N]) # Matriz asociada a las␣
 ↪ecuaciones normales de Gauss
    b1 = vector(RDF,[sumX_kY_k,sumY_k]) # Vector al que se igualan las␣
 ↪ecuaciones de Gauss
    solution1 = M1 \ b1 # El operador '\' resuelve el sistema M1 x = b1
    A = solution1[0] # Solucion para A
    C = e^(solution1[1]) # Solución para C devolviendo la sustitución

    # OUTPUT
    return A,C #

# Utilizando la función
A, C = adjexpnolineal(x_k, y_k)

# Imprimiendo los valores que se buscaban
print(f"Solución para A: {A}")
print(f"Solución para C: {C}")

#Planteamiento de la función
f = C*e**(A*x)

# CONSTRUCCIÓN DEL GRÁFICO Y CÁLCULO DEL ERROR
puntos = [] # Vector que recoja los puntos muestra -- los necesito como puntos␣
 ↪(x,y) para el plot
fx_k = [] # Vector que recoje la función evaluada en los nodos para cálcular el␣
 ↪error

# Ciclo for para construir esos dos vectores anteriores
for i in range(len(x_k)):
    p = (x_k[i],y_k[i]) # Lo que decía, p es el punto muestra en R2 de la forma␣
 ↪(x,y)
    puntos.append(p) # .append() Agrega p al final del vector puntos
    fx_k.append(0) # Agregos 0's con .append() para poder acceder y editar esa␣
 ↪posición con el ciclo
    fx_k[i] = f(x = x_k[i]) # Evalua la función obtenida en los x_k
```

```
error = sqrt(  (1/len(x_k)) * ( (vector(fx_k) - vector(y_k)) *␣
 ↪(vector(fx_k)-vector(y_k)) )  ) # Fórmula para cálcular el error de nuevo␣
 ↪pensandolo en términos de vectores
print(f"E_2: {error}") # Imprime el error en pantalla
# Construcción del plot
plot(f,(x,0,5.3), axes_labels = ['$x$', '$y$'], title = f"f(x) = {f}") +␣
 ↪points(puntos, color = 'red', size = 30)
```

```
Solución para A: 0.7475339236566733
Solución para C: 0.36338074805809034
E_2: 1.272937208549316
```

[ ]:



- $f(x) = Cx^A$, usando las sustituciones $X = \ln(x)$, $Y = \ln(y)$, $C = e^B$ para linealizar los datos.

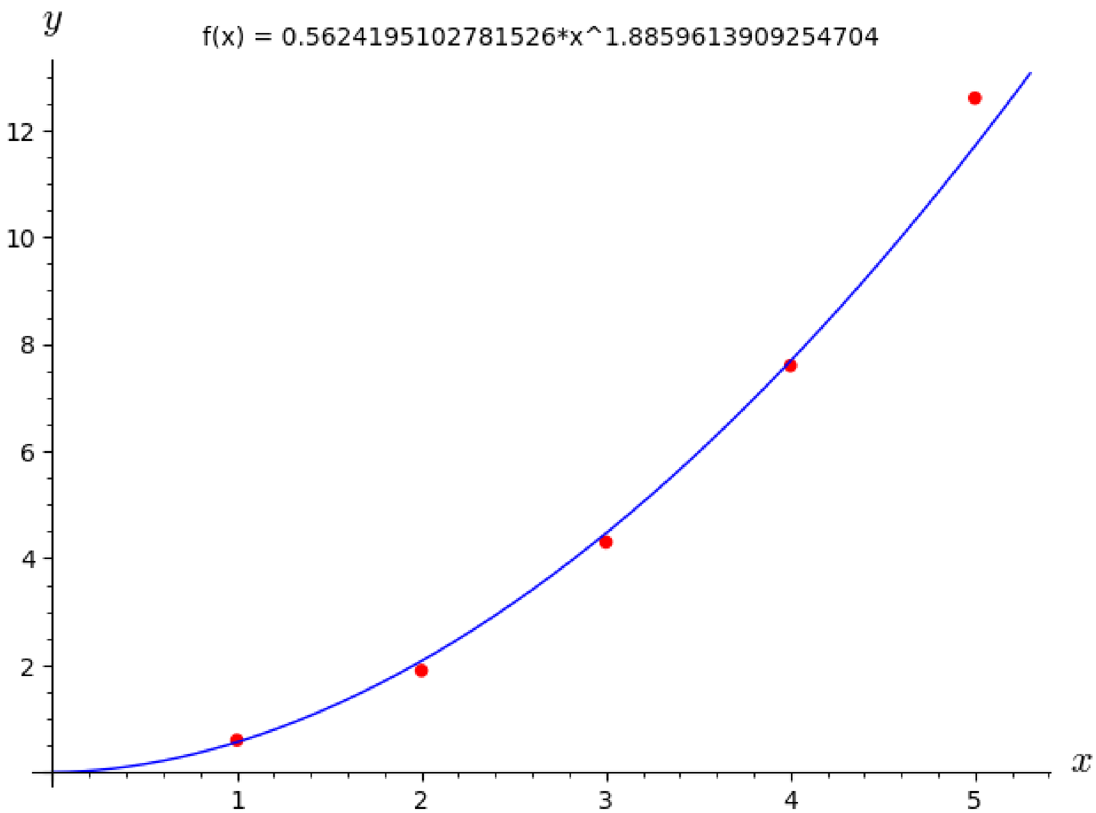
[ ]:
```
# INPUT
x_k = [1, 2, 3, 4, 5]
y_k = [0.6, 1.9, 4.3, 7.6, 12.6]

# FUNCIÓN
```

```python
def adjexpcasilineal(x_k,y_k):
    N = len(x_k) # .len() para ver el numero de puntos muestra con los que trabajo
    X_k = [] # Inicia vector para que recoja la sustitución X_k
    Y_k = [] # Inicia vector para que recoja la sustitución Y_k

    # Ciclo para calcular las sustituciones -- de nuevo, no pude aplicarle el ln a todo el vector de una vez
    for i in range(N):
        X_k.append(0) # Agrego 0's para poder acceder y editar esa posición dentro del ciclo
        Y_k.append(0) # Same
        X_k[i] = log(x_k[i]) # Formulilla X_k = ln(x_k)
        Y_k[i] = log(y_k[i]) # Same Y_k = ln(y_k)

    # CÁLCULO DE LOS COEFICIENTES DE LAS ECUACIONES DE GAUSS
    sumX_k = sum(X_k) # Sumatoria de los X_k sustitución a bordo
    sumY_k = sum(Y_k) # Sumatoria de los Y_k sustitución a bordo
    sumX_k2 = vector(X_k) * vector(X_k) # Sumatoria de X_k^2 de nuevo pensandolos como vectores -- producto punto
    sumX_kY_k = vector(X_k) * vector(Y_k) # Sumatoria de los X_kY_k como pensandoles como producto escalar de vectores de nuevo

    # CONSTRUCCIÓN Y SOLUCIÓN DE SISTEMA DE ECUACIONES NORMALES DE GAUSS
    M1 = matrix(RDF,2,2,[sumX_k2,sumX_k,sumX_k,N]) # Matriz asociada
    b1 = vector(RDF,[sumX_kY_k,sumY_k]) # Valores independientes a los que se iguala
    solution1 = M1 \ b1 # Solución del sistema, con el operador '\'
    A = solution1[0] # Solución para A
    C = e^(solution1[1]) # Solución para C devolviendo la sustitución

    # OUTPUT
    return A,C

# Usando la función
A,C = adjexpcasilineal(x_k,y_k)

# Imprime en pantalla los valores buscados
print(f"Solución para A: {A}")
print(f"Solución para C: {C}")

# Planteamiento de la función
f = C*x**(A)

# CONSTRUCCIÓN DEL GRÁFICO Y CÁLCULO DEL ERROR
```

```
puntos = [] # Vector en el que guardare los puntos muestra pero de la forma (x,y) para facilitar el gráfico
fx_k = [] # Función evaluada en los nodos para cácular el error

# Ciclo for para llenar los dos vectores anteriores
for i in range(len(x_k)):
    p = (x_k[i],y_k[i]) # p es el punto en R2 en la forma que me interesa (x,y)
    puntos.append(p) # Va guardando los p en el vector puntos
    fx_k.append(0) # agrego 0's para poder acceder y editar esa posición dentro del ciclo
    fx_k[i] = f(x = x_k[i]) # Evalua la función y va guardando los valores

# Cálculo del ERROR
error = sqrt(  (1/len(x_k)) * ( (vector(fx_k) - vector(y_k)) * (vector(fx_k)-vector(y_k)) )  ) # formulilla de nuevo pensandoles en forma de vectores
# Imprime en pantalla el error
print(f"E_2: {error}")

# Construcción del gráfico
plot(f,(x,0,5.3), axes_labels = ['$x$', '$y$'], title = f"f(x) = {f}") + points(puntos, color = 'red', size = 30)
```

```
Solución para A: 1.8859613909254704
Solución para C: 0.5624195102781526
E_2: 0.4177544295676972
```

[ ]:



- Use el error cuadrático medio para determinar cual de las curvas se ajusta mejor.
- – El error para $f(x) = 0.36338074805809034e^{0.7475339236566733x}$ es

$$E_2 = 1.272937208549316$$

- – Mientras que para $f(x) = 0.5624195102781526x^{1.8859613909254704}$ es de

$$E_2 = 0.4177544295676972$$

Por lo que esta segunda resultó ser mejor aproximación

2 Splines

2. Un automovil va por una carretera recta y su velocidad se cronometra en varios puntos. Los datos se muestran en la tabla, donde el tiempo se mide en segundos, la distancia en pies y la velocidad en pies por segundo. Gráfique y ponga titulo y nombre en los ejes de la gráfica.

Tiempo	0	3	5	8	13
Distancia	0	225	383	623	993

Tiempo	0	3	5	8	13
Velocidad	75	77	80	74	72

- Use un trazador cúbico sujeto para pedecir la posición del automóvil y su velocidad cuando $t = 10$ segundos. (Contruya un programa para calcular los coeficientes de spline cúbico)

Construire en base a las siguientes fórmulas:

$$h_k = x_{k+1} - x_k$$

$$d_k = \frac{y_{k+1} - y_k}{h_k}$$

$$u_k = 6(d_k - d_{k-1})$$

Sistema de ecuaciones a resolver

$$\begin{bmatrix} \left(\frac{3}{2}h_0 + 2h_1\right) & h_1 & \cdots & 0 & 0 \\ h_{k-1} & 2(h_{k-1} + h_k) & \cdots & 2(h_{k-1} + h_k) & h_k \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & h_{n-2} & \left(2h_{n-2} + \frac{3}{2}h_{n-1}\right) \end{bmatrix} \begin{bmatrix} m_1 \\ m_k \\ \vdots \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} u_1 - 3(d_0 - S'(x_0)) \\ u_k \\ \vdots \\ u_{n-1} - 3(S'(x_n) - d_{n-1}) \end{bmatrix}$$

Fórmulas para el primer y último m_k

$$m_0 = \frac{3}{h_0}(d_0 - S'(x_0)) - \frac{m_1}{2}$$

$$m_n = \frac{3}{h_{n-1}}(S'(x_n) - d_{n-1}) - \frac{m_{n-1}}{2}$$

Forma de las constantes

$$s_{k,0} = y_k, \qquad s_{k,1} = d_k \frac{h_k(2m_k + m_{k+1})}{6}, \qquad s_{k,2} = \frac{m_k}{2}, \qquad s_{k,3} = \frac{m_{k+1} - m_k}{6h_k}$$

Forma del polinomio

$$S(x) = S_k(x) = s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$$

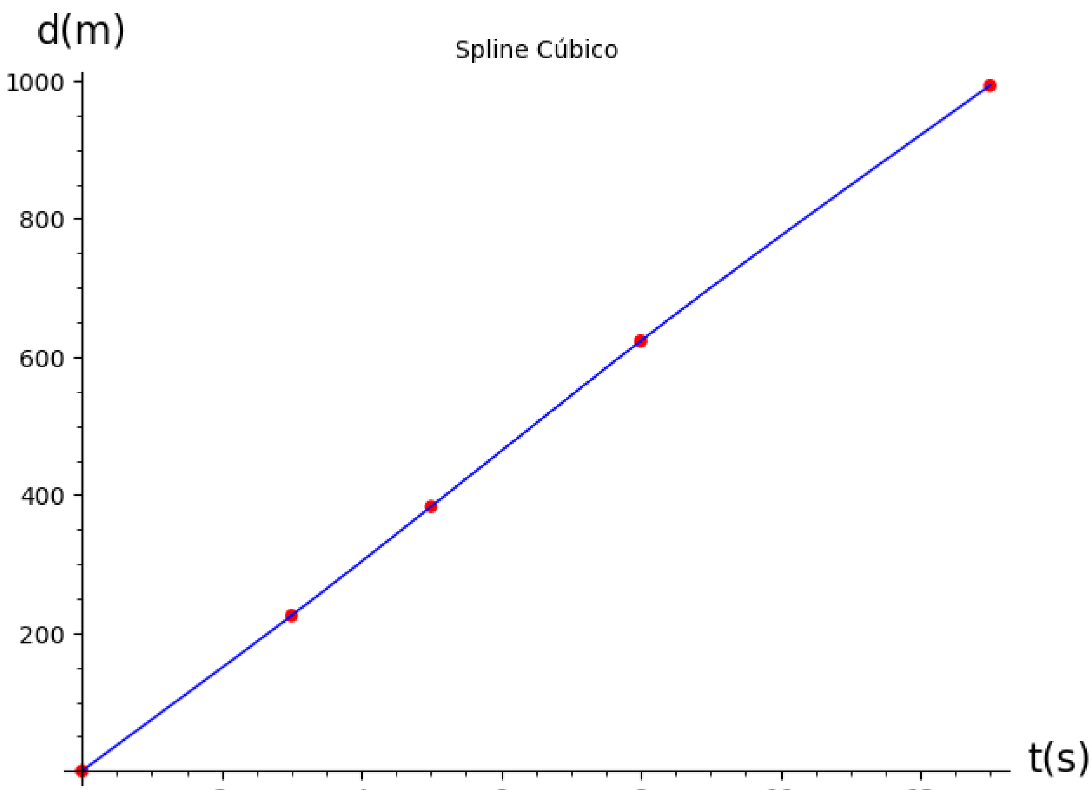
[ ]:
```
# INPUT
x_k = [0, 3, 5, 8, 13] # Vector de abscisas
y_k = [0, 225, 383, 623, 993] # Vector de ordenadas
vi = 75 # Velocidad inicial -- derivada S'(x_0)
vf = 72 # Velocidad final -- derivada en S'(x_N)

# FUNCIÓN
# Todas las variables siguen la notación del libro o eso trate dx.
def sp3(x_k,y_k,vi,vf):
    # INICIACIÓN DE VARIABLES
    N = len(x_k) # Longitud de x_k -- número de punto muestra
    h_k = [0] * (N-1) # Vector de ceros que recogerá los h_k
    d_k = [0] * (N-1) # Vector de ceros que recogerá d_k
```

```
    u_k = [0] * (N-2) # Vector ce ceros que recogerá los u_k

    # OBTECIÓN DE VALORES PREVIOS
    for i in range(N-1): # recorrido h_k y d_k tienen la longitud de x_k - 1, y␣
↪u_k un elemento menos
        h_k[i] = x_k[i+1] - x_k[i] # formulilla
        d_k[i] = (y_k[i+1]-y_k[i])/h_k[i] # formulilla
        if i != 0: # Condicional, ya que u_k se construye con los d_k -- de a␣
↪pares, por eso se salta la 1ra iteración
            u_k[i-1] = 6*(d_k[i]-d_k[i-1]) # formulilla

    # CONSTRUCCIÓN DE LA MATRIZ
    M1 = matrix(RDF,len(u_k)) # RDF trabaja con exactitud soluciones reales --␣
↪crea una matriz de ceros cuadrada (len(u_k)) que vamos a rellenar

    # Elementos de la 1ra fila -- en los que formula no varia según posición
    M1[0,0] = (3/2*h_k[0]) + 2*h_k[1] # formulilla
    M1[0,1] = h_k[1] # formulilla

    # Filas intermedias
    if len(u_k) != 2: # Condicional para que solo se ejecute en caso de que␣
↪existan esas fila intermedias
        for i in range(len(u_k)-2): # Plantea el recorrido de manera que no␣
↪toque ni la 1ra ni la ultima fila
            for j in range(len(u_k)): # Recorrido por columnas -- de la matriz
                if j == 0: # Primer elemento de la fila
                    M1[i+1,j] = h_k[j+1] # formulilla -- notese que empieza en␣
↪la fila i+1 = 1, osea la fila 2 -- python cuenta desde 0
                elif j == len(u_k) - 1: # Ultimo elemento de la fila
                    M1[i+1,j] = h_k[j] # formulilla -- notese que es unicamente␣
↪para los de la ultima columna
                else: # En otro caso, osea las columnas intermedias
                    M1[i+1,j] = 2*(h_k[j]+h_k[j+1]) # formulilla

    # Elementos de la ultima fila
    M1[-1,-2] = h_k[-2] # formulilla -- ultima fila, penultima columna
    M1[-1,-1] = (2*h_k[-2] + (3/2)*h_k[-1]) # formulilla -- ultima fila, ultima␣
↪columna

    # CONSTRUCCIÓN DEL VECTOR AL QUE SE IGUALA LA MATRIZ
    b1 = [0] * (len(u_k)) # Inicia el vector con ceros

    # Primer elemento
    b1[0] = u_k[0] - 3*(d_k[0]-vi) # formulilla

    # Elementos intermedios
```

```
    for i in range(len(b1)): # Plantea el recorrido para llenarle
        if 0 < i < len(b1) - 1: # Condicional, todos menos el primero y el último
            b1[i] = u_k[i] # formulilla

    # Ultimo elemento
    b1[-1] = u_k[-1] - 3*(vf-d_k[-1]) # formulilla

    # SOLUCIÓN DEL SISTEMA
    answer = M1 \ vector(b1) # El operador '\' resuelve M1 para b1

    # OBTENCIÓN DE m_0 y m_N
    m_k = [0] * len(answer) # Aqui quiero recoger el vector answer, m_0, m_N -- pero como lista no vector, que me complica las cosas
    for i in range(len(m_k)): # Ciclo para convertir el vector en lista
        m_k[i] = answer[i] # Ya tengo mis m_k en una lista, como queria

    m_k.insert(0, (3/h_k[0])*(d_k[0]-vi) - (m_k[0]/2)) # formulilla para m_0 -- lo agrego al inicio de los m_k
    m_k.append( (3/h_k[-1])*(vf -d_k[-1]) - (m_k[-1]/2) ) # formulilla para m_N -- lo agrego al final con .append()

    # CONSTRUCCIÓN DEL SPLINE
    S = [0] * (N-1) # Vector de 0's para recoger las funciones
    for i in range(len(S)): # recorrido para llenar a S
        ache = x - x_k[i] # Variable auxiliar para facilitar la legibilidad de la fórmula -- puse 'ache' para no confundir con los h_k
        S[i] = y_k[i]      +    (d_k[i] - (h_k[i] * (2*m_k[i]+m_k[i+1]) )/6 ) * ache       +     m_k[i]/2 * ache**2     +    (m_k[i+1]-m_k[i]) / (6*h_k[i]) * ache**3 # formulilla

    # OUTPUT
    print(f"h: {h_k}\nd: {d_k}\nu: {u_k}\nMatriz:\n{M1}\nm_k:{m_k}\nFunción S(x)") # Salida en pantalla de lo calculado en el camino al polinomio final
    for i in range(len(S)):
        print(f"S{i}: {S[i]}   {x_k[i]} < x < {x_k[i+1]}")

    # Construcción de la gráfica
    puntos = [0] * (N) # Vector para guardar los puntos muestra en la forma (x_k,y_k) -- me facilita el plot
    for i in range(len(x_k)): # Recorrido
        puntos[i] = (x_k[i],y_k[i]) # Lo que decia antes

    # Puntos muestra
    P = points(puntos, color = 'red', size = 30, axes_labels = ["t(s)","d(m)"], title = "Spline Cúbico")
```

```
    # Función S(x) con un for ya que es a trozos
    for i in range(len(S)): # recorrido
        P += plot(S[i],(x,x_k[i],x_k[i+1]), color = 'blue') # Agrega los plots␣
 ↪por trozos a la variable P

    P.show() # Lanza la interfaz para ver el gráfico

    return S # ¡Importante! Es la salida que puedo guardar en una variable,␣
 ↪todo lo demás solo trabaja dentro de la función

#␣
 ↪-------------------------------------------------------------------------------------

# EJERCICIO

# Usando la función
S = sp3(x_k,y_k,vi,vf) # S guarda mi polinomio por pedazos en una lista

# Cálculo de la derivada con un for, ya que esta por pedazos
dS = [0] * len(S) # Vector de 0's recogera la derivada a trozos
for i in range(len(S)): # Recorrido
    dS[i] = diff(S[i],x) # Calcula la derivada en la entrada [i] de S

# Posición y velocidad en t = 10
print(f"Distancia estimada en t = 10: {S[3](x = 10)} ft\nVelocidad estimada en␣
 ↪t = 10: {dS[3](x = 10)} ft/s") # Salida en pantalla -- evaluando primero en␣
 ↪el polinomio, luego en su derivada
```

```
h: [3, 2, 3, 5]
d: [75, 79, 80, 74]
u: [24, 6, -36]
Matriz:
[ 8.5  2.0  0.0]
[ 2.0 10.0  3.0]
[ 0.0  3.0 13.5]
m_k:[-1.31858407079646, 2.63716814159292, 0.7920353982300885,
-2.398230088495575, -0.0008849557522123686]
Función S(x)
S0: 0.21976401179941002*x^3 - 0.65929203539823*x^2 + 75.0*x   0 < x < 3
S1: -0.15376106194690262*(x - 3)^3 + 1.31858407079646*(x - 3)^2 +
76.97787610619469*x - 5.933628318584056   3 < x < 5
S2: -0.1772369714847591*(x - 5)^3 + 0.39601769911504425*(x - 5)^2 +
80.40707964601769*x - 19.035398230088447   5 < x < 8
S3: 0.07991150442477875*(x - 8)^3 - 1.1991150442477876*(x - 8)^2 +
77.99778761061947*x - 0.9823008849557482   8 < x < 13
```




```
Distancia estimada en t = 10: 774.838407079646 ft
Velocidad estimada en t = 10: 74.16026548672566 ft/s
```

- Use la derivada del Spline para determinar la velocidad máxima predecible del automóvil.

[ ]:
```
# Calculo de la segunda derivada, de nuevo toca con for
d2S = [0] * len(S) # Vector que recogera la 2da derivada
maximos = [0] * len(S) # Vector que recogera los x_i donde S''(x_i) = 0

for i in range(len(S)): # Recorrido
    d2S[i] = diff(S[i], x, 2) # Calcula la 2da derivada
    maximos[i] = solve(d2S[i] == 0, x) # Resuelve cuando la 2da derivada es␣
 ↪igual a 0

print(f"Candidatos a maximo para cada pedazo: {maximos}") # Para ver en␣
 ↪pantalla los candidatos

# Candidatos a maximo
candidatos = [1, 2443/417, 4142/721, 35222/2709] # Guarde los candidatos a mano␣
 ↪porque me fue dificil manipular el array que me devuelve solve
# Evaluando los candidatos en la función velocidad, S'(x)
```

```python
for i in range(len(S)): # Recorrido -- toca ir por pedazos
    candidatos[i] = dS[i](x = maximosAux[i]) # Evalua los candidatos y los␣
 ↪guarda en candidatos xd

tope = max(candidatos) # max() busca el maximo en la lista candidatos

# salida en consola
print(f"S'(x) evaluada en los candidatos: {candidatos}\nVelocidad máxima␣
 ↪estimada: {tope} ft/s")
```

```
Candidatos a maximo para cada pedazo: [[x == 1], [x == (2443/417)], [x ==
(4142/721)], [x == (35222/2709)]]
S'(x) evaluada en los candidatos: [74.34070796460176, 80.74706606396299,
80.7020331889092, 71.9999991833188]
Velocidad máxima estimada: 80.74706606396299 ft/s
```