Aproximación de Funciones

Carlos Aguirre Maeso
Escuela Politécnica superior

APROXIMACION DE FUNCIONES

- En este parte se estudiará la aproximación de funciones disponibles en forma discreta (puntos tabulados), con funciones analíticas sencillas, o bien de aproximación de funciones cuya complicada naturaleza exija su reemplazo por funciones más simples, usualmente polinomios.
- Una vez que se ha determinado un polinomio una función $\tilde{\boldsymbol{f}}(\boldsymbol{x})$ o un polinomio $\mathbf{P_n(x)}$ de manera que aproxime satisfactoriamente una función dada **f(x)** sobre un intervalo de interés, puede esperarse que al diferenciar $\tilde{\boldsymbol{f}}(\boldsymbol{x})$ o $\mathbf{P_n(x)}$ o integrarlas, también aproxime la derivada o integral correspondiente a **f(x)**.

Polinomio de Taylor

- Una de las aproximaciones más conocidas para una función f(x) es su polinomio de Taylor.
- Para calcular el polinomio necesito saber no solo el valor de la función en un punto, sino también el valor de sus derivadas.

$$P_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Se demomina el polinomio de Taylor de orden k.

Polinomio de Taylor

- Se puede demostrar que para una función k+1 veces diferenciable se verifica:

$$f(x)=P_k(x)+R_k(x)$$

Donde $R_k(x)$ se denomina resto de Taylor. Hay muchas expresiones para dicho resto, siendo la más conocida la de Peano

$$R_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!}(x-a)^{k+1}$$

Donde $\xi \in [x, a]$.

Polinomio de Taylor

- Aunque el polinomio de Taylor tiene un valor teórico interesante, en la práctica no es usual aproximar una función por su polinomio de Taylor.
  - Usualmente de una función no tenemos sus derivadas, sino un conjunto de pares (x_i,y_i) que obtenemos mediante experimentación o mediante evaluación de la función (no tenemos la función, pero podemos evaluarla).
  - El polinomio de Taylor es computacionalmente muy costoso, lo cual, además implica un error de aproximación alto por las operaciones aritméticas.

Aproximación polinómica

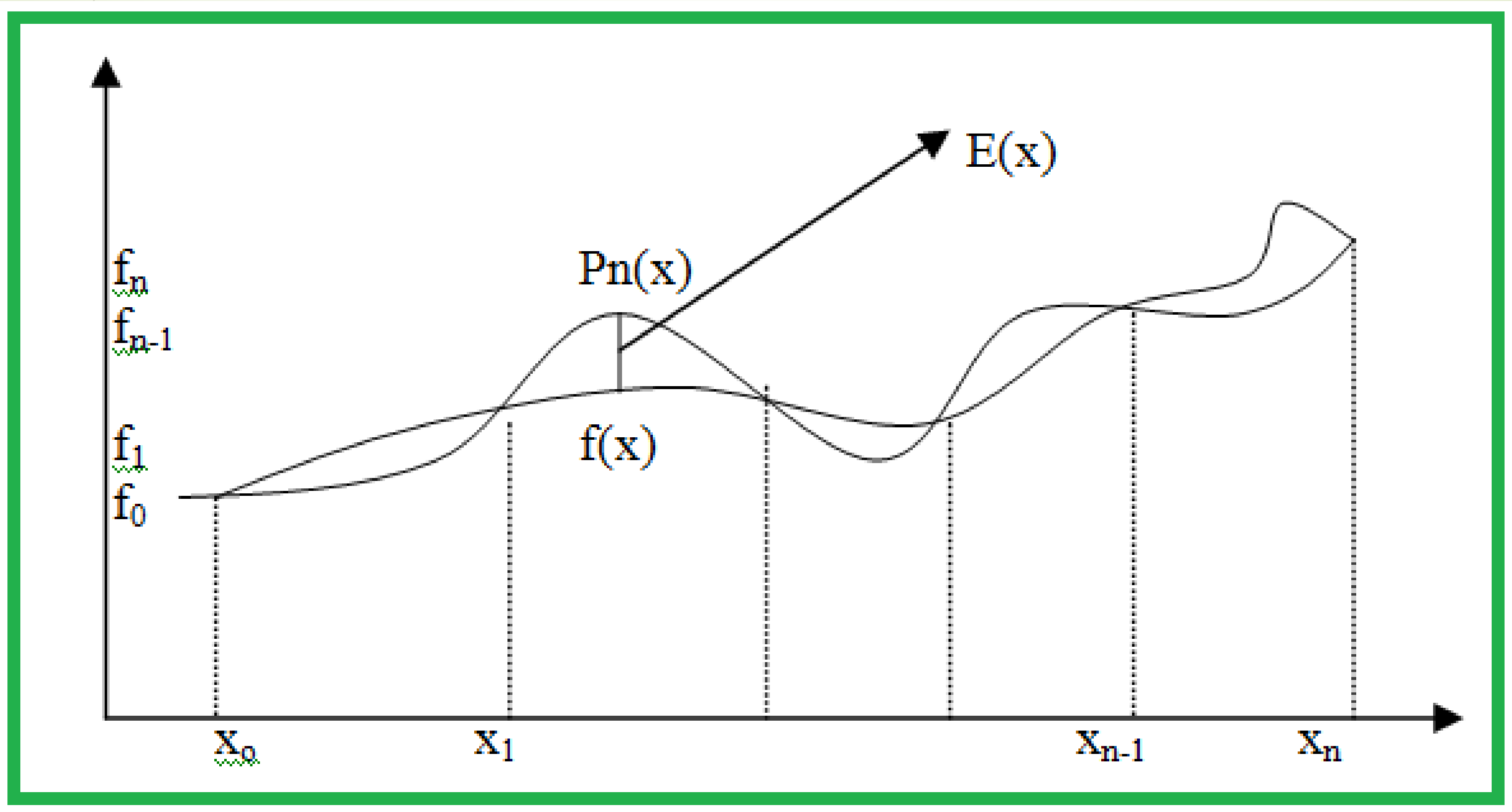
- Se realiza tanto cuando la función puede ser conocida en forma explícita o mediante un conjunto de valores tabulados para cada uno de los argumentos por donde pasa la función (valores funcionales).

x_i	x_0	x_1	...	x_n
$f(x_i)$	f_0	f_1	...	f_n

- Normalmente se acepta aproximar a la función tabulada en puntos coincidentes mediante un polinomio de grado "n" (condición de aproximación):

$f(x_i) \approx P_n(x_i)$; para todo x_i en $[x_o, x_n]$
Donde: $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_o$, **con** $a_n \neq 0$

Aproximación polinómica



Aproximación polinómica

- Donde: $E_{abs}(x) = P_n(x)-f(x)$;   Para todo x en $[x_0,x_n]$

Observaciones:

1) Los polinomios son funciones fáciles de derivar, integrar, evaluar y de programar en un computador. Véase :

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1}+\ldots+a_1x+a_o$$

$$P'_n(x) = na_n x^{n-1} + (n-1)a_{n-1}x^{n-2}+\ldots+a_1$$

2) Los polinomios presentan propiedades analíticas importantes que facilitan el cálculo de las raíces del polinomio, así mismo nos permite identificar el tipo de raíz (Real ó complejo).

Cálculos Analíticos

- Interpolación : $\mathbf{f(x) \approx P_n(x)}$, x en $[x_o, x_n]$
- Extrapolación : $\mathbf{f(x) \approx P_n(x)}$, $x<x_0$ o $x>x_n$
- Diferenciación : $\mathbf{f'(x) \approx P'_n(x)}$
- Integración : $\int_a^b f(x)dx \approx \int_a^b P_n(x)dx$

Cálculo de Polinomio Interpolante

$$P_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n$$

$$y_i = f(x_i) = P_n(x_i) \quad para \quad i = 0 \cdots n$$

$$Sistema\ de\ Ecuaciones\ Lineales\ de\ Vandermonde$$

$$\begin{bmatrix} x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Este procedimiento en la practica no es muy usual debido a que la matriz de Vandermonde es mal condicionada (es decir $\|A\|\|A^{-1}\| > 1, con \|A\|$ norma de la matriz).

$$\|A\|_2 = \sqrt{traza(A^*A)}, \quad \|A\|_1 = \underbrace{max}_{1\le j\le m}\left(\sum_{i=1}^{n}|a_{ij}|\right), \quad \|A\|_\infty = \underbrace{max}_{1\le i\le n}\left(\sum_{j=1}^{m}|a_{ij}|\right)$$

Propiedades de Aproximación

1) Siempre que se acepte aproximar la función f(x) mediante un polinomio de grado n: $P_n(x)$ que pase por (n+1) puntos coincidentes, se puede construir un polinomio que es único (propiedad de existencia y unicidad).

2) El error de aproximación viene dado por:

$$E_n = P_n(x) - f(x) = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}(x-x_0)(x-x_1)\ldots(x-x_n)$$

$$Para\ algún\ \varepsilon \in <x_0, x_n> \quad ; \; x \in \; [x_0, x_n]$$

3) Cota superior de error (M):

$$|E_n(x)| = |P_n(x) - f(x)| \leq \frac{M}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

$$Donde: \quad M = máx\{|f^{(n+1)}(x)|\} \quad para\ x \in [x_0, x_n]$$

Polinomios de interpolación de Lagrange

Para intervalos iguales o no.

$$P_n(x) = \sum_{i=0}^{n} f(x_i)L_i(x) \;= f(x_0)L_0(x) \;+ f(x_1)L_1(x) + \ldots + f(x_1)L_n(x)$$

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j}\right)$$

$$E_n = f(x) - P_n(x) = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}(x - x_0)(x - x_1)\ldots(x - x_n)$$

para algún:

$$\varepsilon \;\in < x_0, x_n > \quad ; \; x \in \; [x_0, x_n]$$

- Ejercicio: Comprobar que el Polinomio interpolador de Lagrange pasa por los puntos $(x_i, f(x_i))$

Ejemplo

Obtener el Polinomio de Lagrange a partir de los siguientes datos:

X	Y
0	-2
2	2
5	6

$$P_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

$$= \frac{(x-2)(x-5)}{(0-2)(0-5)}(-2) + \frac{(x-0)(x-5)}{(2-0)(2-5)}(2) + \frac{(x-0)(x-2)}{(5-0)(5-2)}(6)$$

$$P_2(x) = -\frac{2}{15}x^2 + \frac{34}{15}x - 2$$

Ejemplo 1

Ejecutad ahora el siguiente código Python:

```
import numpy as np
from scipy.interpolate import lagrange
from numpy.polynomial.polynomial import Polynomial

x = np.array([0, 2, 5])
y = np.array([-2,2,6])
poly = lagrange(x, y)

print(Polynomial(poly).coef)
```

Ejemplo 2

Ejecutad ahora el siguiente código Python:

```
import numpy as np
from scipy.interpolate import lagrange
import matplotlib.pyplot as plt

x = np.arange(-1,1.1,.01)
y = -x**2
plt.plot(x,y,'r')
y = -x**2 + np.random.normal(0,0.15,len(x))
plt.plot(x[::10],y[::10],'ro')

X=x[::10]
Y=y[::10]
poly = lagrange(X, Y)

Y=poly(x)
plt.plot(x,Y,'g')
plt.ylim(-1.5,.5)
```

¿Consideráis que el polinomio de Lagrange ha hecho una buena aproximación ?

Herramientas de Interpolación

- A continuación definiremos algunas herramientas que nos permitirán más adelante construir otros polinomios de interpolación:
  - Diferencias Finitas
  - Diferencias Divididas

Diferencia Finita hacia adelante o progresiva

- Se emplean cuando los valores x están igualmente espaciados
- Diferencia finita de primer orden:

$$\boxed{\Delta f_k = f_{k+1} - f_k}$$

- Diferencia finita de segundo orden:

$$\boxed{\Delta^2 f_k = \Delta f_{k+1} - \Delta f_k}$$

- Diferencia Finita de orden n:

$$\boxed{\Delta^n f_k = \Delta^{n-1} f_{k+1} - \Delta^{n-1} f_k}$$

Tabla de diferencias finitas hacia adelante (h=constante)

x_k	$f(x_k)$	Δf_k	$\Delta^2 f_k$	$\Delta^3 fk$	$\Delta^4 f_k$
x_o	f_0				
		Δf_0			
x_1	f_1		$\Delta^2 f_0$		
		Δf_1		$\Delta^3 f_0$	
x_2	f_2		$\Delta^2 f_1$		$\Delta^4 f_0$
		Δf_2		$\Delta^3 f_1$	
x_3	f_3		$\Delta^2 f_2$		
		Δf_3			
x_4	f_4				

Diferencia finita hacia atrás o regresiva:

$$\nabla^n f_k = \nabla^{n-1} f_k - \nabla^{n-1} f_{k-1}$$

Diferencia Finita Central:

$$\delta^{\,n} f_k = \delta^{\,n-1} f_{k+1/2} - \delta^{\,n-1} f_{k-1/2}$$

Diferencias Divididas

- Se define para puntos o argumentos desigualmente espaciados:
- Diferencia dividida de Primer orden:

$$\boxed{f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}}$$

- Diferencia dividida de segundo orden:

$$\boxed{f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}}$$

- Diferencia dividida de orden “n”:

$$\boxed{f[x_i, x_{i+1}, \ldots, x_{i+n-1}, x_{i+n}] = \frac{f[x_{i+1}, \ldots, x_{i+n}] - f[x_i, \ldots, x_{i+n-1}]}{x_{i+n} - x_i}}$$

Polinomio de interpolación de Newton basado en diferencias Divididas

- Sea la función f(x) tabulada para (n+1) puntos, siempre es posible construir un polinomio de grado "n" (o menor) que pase por dichos puntos y se le puede dar la forma:

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

- Se trata ahora de determinar los coeficientes a_k.

  Si $x=x_0$, $P_n(x_0)=a_0 \approx f(x_0)$

  Si $x=x_1$, $P_n(x_1)=f(x_0)+a_1(x_1-x_0) \approx f(x_1)$

  $a_1=(f(x_1)-f(x_0))/(x_1-x_0)=f[x_0,x_1]$

- Se puede demostrar que en general se cumple:

$$\boxed{a_k = f[x_0, x_1, \ldots, x_k]}$$

Por lo tanto:

$$P_n(x) = f(x_0) + f[x_0x_1](x-x_0) + f[x_0x_1x_2](x-x_0)(x-x_1) + f[x_0x_1\ldots x_n](x-x_0)(x-x_1)\ldots(x-x_{n-1})$$

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} f[x_0\ldots x_k](x-x_0)\ldots(x-x_{k-1}) = f(x_0) + \sum_{i=0}^{n} f[x_0\ldots x_i]\prod_{j=0}^{i-1}(x-x_j)$$

Error de Interpolación

$$e_n(x) = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}(x-x_0)(x-x_1)\ldots(x-x_n) = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}\prod_{i=0}^{n}(x-x_i) \quad \varepsilon \in [x_0, x_n]$$

$$e_n(x) = f[x_0x_1\ldots x_nx]\prod_{i=0}^{n}(x-x_i)$$

Se suele aproximar el error considerando $x=x_{n+1}$, es decir, se requiere un punto adicional.

Ejemplo.- Obtener el polinomio interpolante

x	0	1	2	4	5
y	2	3	10	66	127

Estime y(2.5)

Tabla de diferencias divididas

x	y	y[,]	y[, ,]	y[, , ,]	y[, , , ,]
0	2				
		1			
1	3		3		
		7		1	
2	10		7		0
		28		1	
4	66		11		
		61			
5	127				

De la tabla anterior, obtenemos los coeficientes del polinomio interpolante:

$$P(x) = y_0 + y[x_0, x_1](x - x_0) + y[x_0, x_1, x_2](x - x_0)(x - x_1) +$$
$$+y[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) +$$
$$+y[x_0, x_1, x_2, x_3, x_4](x - x_0)(x - x_1)(x - x_2)(x - x_3)$$

$$P(x) = 2 + (1)(x - 0) + 4(x - 0)(x - 1) +$$
$$+(1)(x - 0)(x - 1)(x - 2)$$
$$+ (0)(x - 0)(x - 1)(x - 2)(x - 4)$$
$$P(x) = x^3 + 2$$

$$y(2.5) \approx P(2.5) = 2.5^3 + 2$$
$$y(2.5) \approx 17.625$$

Python no tiene un función que implemente el polinomio interpolador de Newton, pero se puede implementar fácilmente

```
def interp_newton_coeffs(xvals,yvals):
    nbr_data_points = len(xvals)
    depth =1
    coeffs = [yvals[0]]
    iter_yvals = yvals

    while depth < nbr_data_points:
        iterdata=[]
        for i in range(len(iter_yvals)-1):
            delta_y= iter_yvals[i+1]-iter_yvals[i]
            delta_x= xvals[i+depth]-xvals[i]
            interval = (delta_y/delta_x)
            iterdata.append(interval)
            if i==0: coeffs.append(interval)
        iter_yvals=iterdata
        depth+=1
    return coeffs
```

Una vez definida la función, ejecutad el siguiente código en Python

```
def newton_pol(xvals,coeffs):
    def f(i):
        terms  = []
        retval = 0

        for j in range(len(coeffs)):

            iterval   = coeffs[j]
            iterxvals = xvals[:j]
            for k in iterxvals: iterval*=(i-k)
            terms.append(iterval)
            retval+=iterval
        return(retval)
    return(f)

xvals=[0,1,2,4,5]
yvals=[2,3,10,66,127]
coeffs = interp_newton_coeffs(xvals, yvals)
pol=newton_pol(xvals,coeffs)

print(pol(2.5))
```

Ejercicio: Construid una función Python que a partir de los coeficientes de Newton devuelva los coeficientes del polinomio de la forma $\mathbf{P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_o}$

Polinomio de interpolación basado en Diferencias Finitas Progresivas

- Se debe hallar una relación entre las diferencias finitas y divididas; se deja como ejercicio la demostración que:

$$\boxed{f[x_0, x_1, x_2, \ldots. x_k] = \frac{\Delta^k f_0}{k!\, h^k}}$$

- Reemplazando en el polinomio basado en diferencias divididas se tiene:

$$\boxed{P_n(x) = f_0 + \frac{\Delta f_0}{1!\, h^1}(x - x_0) + \frac{\Delta^2 f}{2!\, h^2}(x - x_0)(x - x_1) + \ldots + \frac{\Delta^n f_0}{n!\, h^n}(x - x_0)\ldots(x - x_{n-1})}$$

Polinomio de interpolación basado en Diferencias Finitas Progresivas

- Teniendo en cuenta que los intervalos se tomarán igualmente espaciados (h=cte) para x, y haciendo el cambio de variable, se demuestra que:

$$s = \frac{x - x_0}{h}$$

$$P_n(s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \ldots + \frac{s(s-1)\ldots(s-n+1)}{n!}\Delta^n f_0$$

$$P_n(s) = \sum_{i=0}^{n} \Delta^i f_0 \binom{s}{i}$$

$$P_n(s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \ldots + \frac{s(s-1)\ldots(s-n+1)}{n!}\Delta^n f_0$$

$$P_n(s) = \sum_{i=0}^{n} \Delta^i f_0 \binom{s}{i}$$

- Esta última forma se conoce como polinomio de interpolación de Newton Progresivo con cambio de escala.
- **Ejercicio:** deducir la fórmula de error para el polinomio anterior.

Ejemplo

a) Aproximar los siguientes datos usando un polinomio basado en diferencias finitas:

X	2	3	4
Y	0	-1	0

b) Estime Y(2.5):

c) Calcule el error cometido, si esta data se obtuvo de la función Y=sen(pi*X/2)

Solución

Tabla de diferencias finitas:

X	Y	ΔY	$\Delta^2 Y$
2	0		
		-1	
3	-1		2
		1	
4	0		

$$P(s) = Y_0 + s\Delta Y_0 + \frac{s(s-1)}{2!}\Delta^2 Y_0$$
$$P(s) = 0 + s(-1) + \frac{s(s-1)}{2!}(2)$$
$$P(s) = s^2 - 2s$$

$$X = 2.5$$
$$s = \frac{X - X_0}{h} = \frac{X-2}{1}$$
$$s = \frac{2.5-2}{1} = 0.5$$
$$P(s = 0.5) = (0.5)^2 - 2(0.5)$$
$$= -0.75$$
$$y(2.5) = sen\left(\frac{2.5\ \pi}{2}\right) = -0.7071$$
$$Error = 0.0429$$

Polinomio de interpolación basado en Diferencias Finitas Regresivas

$$P_n(s) = f_n + s\nabla f_n + \frac{s(s+1)}{2!}\nabla^2 f_n + \frac{s(s+1)(s+2)}{3!}\nabla^3 f_n + \ldots + \frac{s(s+1)(s+2)+\ldots+(s+n-1)}{n!}\nabla^n f_n$$

$$Teniendo\ en\ cuenta\ que: s = \frac{x - x_n}{h}$$

Polinomio de interpolación basado en Diferencias Finitas Centrales

Polinomio de Stirling

$$P_{2m}(s) = f_0 + \frac{s}{1!}\frac{[\delta f_{-1/2} + \delta f_{+1/2}]}{2} + \frac{s^2}{2!}\delta^2 f_0 + \frac{s(s^2-1^2)}{3!}\frac{[\delta^3 f_{-1/2} + \delta^3 f_{+1/2}]}{2} + \frac{s^2(s^2-1^2)}{4!}\delta^4 f_0 + \frac{s^2(s^2-1^2)(s^2-2^2)}{5!}\frac{[\delta^5 f_{-1/2} + \delta^5 f_{+1/2}]}{2} + \ldots$$

Queda para el estudiante demostrar que el polinomio anterior puede representarse en la forma siguiente:

$$P_{2n}(s) = f_0 + \binom{s}{1}\delta_{1/2} + \binom{s}{2}\delta_0^2 + \binom{s+1}{3}\delta_{1/2}^3 + \binom{s+1}{4}\delta_0^4 + \ldots + \binom{s+n-1}{2n-1}\delta_{1/2}^{2n-1} + \binom{s+n-1}{2n}\delta_0^{2n}$$

$$P_{2n}(s) = f_0 + \sum_{i=1}^{n}\binom{s+i-1}{2i-1}\delta_{1/2}^{2i-1} + \binom{s+i-1}{2i}\delta_0^{2i} \qquad s = \frac{x - x_0}{h}$$

Interpolación segmentaria o Splines

- Un Spline o trazador es una función que consiste en trozos de polinomios unidos con ciertas condiciones de continuidad.
- Dados los nodos $x_o<x_1<\ldots<x_n$, un spline de grado *k* con esos nodos es una función *S* tal que:
  - En cada sub-intervalo $[t_{i-1},t_i]$ *S* es un polinomio de grado $\leq k$
  - La (k-1)-iésima derivada de *S* es continua en $[x_o, x_n]$

Spline Lineal

$$s_i(x) = m_i x + b_i,\ para\ x \in [x_i, x_{i+1}]\ ,\quad i = 0,\ 1,2,\ \cdots,\ n-1$$

Las condiciones, $s_i(x_i) = y_i$ y $s_i(x_{i+1}) = y_{i+1}$ producen 2n ecuaciones para encontrar 2n incógnitas. Aplicando esto, conseguimos:

$$s_i(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i} = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i),\quad x \in [x_i,\ x_{i+1}]$$

cuyo resultados son líneas rectas que ensamblan puntos vecinos.

Claramente se observa que, $s_i(x)$es la formula de interpolación de Lagrange para un conjunto de datos que consiste en los puntos (x_i,y_i) y (x_{i+1},y_{i+1}), observad que $\frac{y_{i+1} - y_i}{x_{i+1} - x_i}$ es la diferencia divida de Newton.

Ejemplo Encontrar los Splines lineales para el siguiente conjunto de datos:

i	0	1	2	3	4
x	0	5	7	8	10
y	0	2	-1	-2	20

Splines Lineales:

$$s_0(x) = 0\frac{x-5}{0-5} + 2\frac{x-0}{5-0} = \frac{2}{5}x, \quad x \in [0,\ 5]$$

$$s_2(x) = -1\frac{x-8}{7-8} - 2\frac{x-7}{8-7} = -x+6, \quad x \in [7,\ 8]$$

$$s_1(x) = 2\frac{x-7}{5-7} - 1\frac{x-5}{7-5} = -1.5x + 9.5, \quad x \in [5,\ 7]$$

$$s_3(x) = -2\frac{x-10}{8-10} + 20\frac{x-8}{10-8} = 11x - 90, \quad x \in [8,\ 10]$$

Spline cúbico

- Corresponde a la categoría de interpolación segmentaria donde cada tramo es aproximado con polinomios de tercer grado, aplicando condiciones de suavidad que se ven a continuación:

- Consideremos dos puntos consecutivos: $(x_i, y_i,)$ y (x_{i+1}, y_{i+1}), y el polinomio cúbico:

$$S_i(x)= a_i(x-x_i)^3 +b_i(x- x_i)^2 +c_i (x-x_i)+d_i$$

- A continuación impondremos las condiciones de suavidad, esto es, restricciones a las derivadas de primer y segundo orden.

- Garantizamos que el spline pase por todos los puntos de la tabla y además la continuidad.

- Ambos puntos (x_i, y_i) y (x_{i+1}, y_{i+1}) pertenecen a $S_i(x)$

Para (x_i, y_i):

$S_i(x_i)=a_i (x_i - x_i)^3 + b_i (x_i - x_i)^2 + c_i (x_i - x_i) + d_i = d_i = y_i$ (1)

Para (x_{i+1}, y_{i+1}):

$S_i(x_{i+1})=a_i (x_{i+1} - x_i)^3 + b_i (x_{i+1} - x_i)^2 + c_i (x_{i+1} - x_i) + d_i = a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i$ (2)

Donde: $h_i = xi_{+1} - x_i$

Garantizamos la primera y segunda diferenciabilidad en los nodos comunes:

La primera derivada:

$S_i'(x_i)=3a_i (x_i - x_i)^2 + 2b_i (x_i - x_i) + c_i$ (3)

La segunda derivada:

$S_i''(x_i)=6a_i (x_i - x_i) + 2b_i$ (4)

Definiendo: $S_i''(x_i)=Mi$ y $S_i''(x_{i+1})= M_{i+1}$ y reemplazando (4) en x_i y x_{i+1}, entonces:

Si $x=x_i$, $M_i=6a_i/(x_i- x_i)+2b_i = 2b_i$ (5)

Si $x=x_{i+1}$, $M_{i+1}=6a_i/(x_{i+1}-x_i)+2b_i = 6a_ih_i +2b_i$ (6)

Reordenando las ecuaciones (5) y (6) se obtiene:

$$b_i = \frac{M_i}{2}$$

$$a_i = \frac{M_{i+1} - 2b_i}{6h_i} = \frac{M_{i+1} - M_i}{6h_i}$$

Si reemplazamos las ecuaciones (1), (7), (8) en (2) se llega a:

$$y_{i+1} = \frac{M_{i+1} - M_i}{6h_i} h_i^3 + \frac{M_i}{2} h_i^2 + c_i h_i + y_i$$

Con lo cual:

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{M_{i+1} - 2M_i}{6} h_i$$

Ahora impondremos continuidad para la primera derivada:

$S'(x_{i-1})=S'(x_i)$ (10)

Evaluando (3) en x_{i-1} , se obtiene:

$S'(x_{i-1})=3a_{i-1}(x_i-x_{i-1})^2 +2b_{i-1} (x_i-x_{i-1})+c_{i-1} = 3a_{i-1}h_{i-1}^2 +2b_{i-1} h_{i-1}+c_{i-1}$ (11)

y además lo evaluamos en x_i

$$S'(x_i) = 3a_i(x_i - x_i)^2 + 2b_{i-1}(x_i - x_i) + c_i$$
$$= c_i \qquad (12)$$

De la ecuación (10), con reemplazos de (11) y (12)

$$3a_{i-1}h_{i-1}^{\;2} + 2b_{i-1}h_{i-1} + c_{i-1} = c_i \qquad (13)$$

A continuación reemplazamos (7), (8) y (9) en la expresión (13) (para i y i-1)

$$3\frac{M_{i+1} - M_i}{6h_i}h_{i-1}^{\;2} + 2\frac{M_i}{2}h_{i-1} + \left(\frac{y_i - y_{i-1}}{h_{i-1}} - \frac{M_i + 2M_{i-1}}{6}h_{i-1}\right) = \frac{y_{i+1} - y_i}{h_i} - \frac{M_{i+1} + 2M_i}{6}h_i$$

Reordenado la última expresión se concluye:

$$h_{i-1}M_{i-1} + 2(h_{i-1} + h_i)M_i + h_iM_{i+1} = 6(y[x_i, x_{i+1}] - y[x_{i-1}, x_i])$$

Con i=1, 2, ..., n-1. Además: $y[x_i, x_{i+1}] = \dfrac{y_{i+1} - y_i}{h_i}$

Asi se define un sistema de n-1 ecuaciones para n+1 incognitas (los M_i).

En general, para resolver el sistema se debe imponer condiciones externas. Existen dos posibilidades:
a) Spline de frontera libre o natural

Sea el conjunto de datos: (x_0,y_0), (x_1,y_1), (x_2,y_2), … , (x_n,y_n)

Donde cada segmento puede ser aproximado con un polinomio cúbico de la forma:

$S_i(x)= a_i(x-x_i)^3 +b_i(x- x_i)^2 +c_i (x-x_i)+d_i$, ¡=0,1:,n-1

Haciendo: $h_i=x_{i+1}-x_i$, $M_i=S''(x_i)$

Para el spline natural: $M_0 =M_n =0$

Debemos primero resolver el siguiente sistema tridiagonal:

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{bmatrix} = 6 \begin{bmatrix} y[x_1,x_2]-y[x_0,x_1] \\ y[x_2,x_3]-y[x_1,x_2] \\ \vdots \\ y[x_{n-2},x_{n-1}]-y[x_{n-3},x_{n-2}] \\ y[x_{n-1},x_n]-y[x_{n-2},x_{n-1}] \end{bmatrix}$$

Una vez obtenidos $M_1,\cdots M_{n-1}$, obtendremos los coeficientes:

$$a_i = \frac{M_{i+1}-M_i}{6h_i}$$

$$b_i = \frac{M_i}{2}$$

$$c_i = y[x_i,x_{i+1}] - \frac{M_{i+1}+2M_i}{6}h_i$$

$$d_i = y_i$$

b) Spline de frontera sujeta

$S_0'(x_0) = A \quad y \quad S_n'(x_n) = B$, con lo cual se agregan dos ecuaciones:

$$2h_0 M_0 + h_0 M_1 = 6\left(y[x_0, x_1] - A\right)$$
$$h_{n-1} M_{n-1} + 2h_{n-1} M_n = 6\left(B - y[x_{n-1}, x_n]\right)$$

Y llegamos a tener n+1 ecuaciones con n+1 incognitas:

$$\begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = 6 \begin{bmatrix} y[x_0, x_1] - A \\ y[x_1, x_2] - y[x_0, x_1] \\ \vdots \\ y[x_{n-1}, x_n] - y[x_{n-2}, x_{n-1}] \\ B - y[x_{n-1}, x_n] \end{bmatrix}$$

Ejemplo

Obtener un Spline Natural para los siguientes datos:

X	0	1	1.5	2.25
F(x)	2	4.4366	6.7134	13.9130

Solución

i	hi	x	F(x)	f[,]
0	1	0	2	2.4366
1	0.5	1	4.4366	4.5536
2	0.75	1.5	6.7134	9.5995
		2.25	13.9130	

En este caso:

$$\begin{bmatrix} 2(h_0+h_1) & h_1 \\ h_1 & 2(h_1+h_2) \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = 6 \begin{bmatrix} f[x_1x_2]-f[x_0x_1] \\ f[x_2x_3]-f[x_1x_2] \end{bmatrix}$$

Reemplazando:

$$\begin{bmatrix} 3 & 0.5 \\ 0.5 & 2.5 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = 6 \begin{bmatrix} 4.5536-2.4366 \\ 9.5995-4.5536 \end{bmatrix} = \begin{bmatrix} 12.7020 \\ 30.2754 \end{bmatrix}$$

$$M_1 = 2.2921 \quad M_2 = 11.6517 \quad M_0 = M_3 = 0$$

Para i=0, 1 y 2, reemplazamos las siguientes fórmulas para obtener los polinomios segmentarios:

$$a_i = \frac{M_{i+1} - M_i}{6h_i}$$

$$b_i = \frac{M_i}{2}$$

$$c_i = y[x_i, x_{i+1}] - \frac{M_{i+1} + 2M_i}{6} h_i$$

$$d_i = y_i$$

$$S(x) = \begin{cases} x \in [0, \ \ 1] & 0.382(x-0)^3 + 0(x-0)^2 + 2.0546(x-0) + 2 \\ x \in [1, \ \ 1.5] & 3.1199(x-1)^3 + 1.146(x-1)^2 + 3.2005(x-1) + 4.4366 \\ x \in [1.5, \ \ 2.25] & -2.5893(x-1.5)^3 + 5.8259(x-1.5)^2 + 6.6866(x-1.5) + 6.7134 \end{cases}$$

Ejemplo

Obtener una interpolación por Spline Cúbico forzado para $f(x)=(x-1)^4$ en x=0, 1, 1.5.

Se pide:

a) Mostrar las funciones Spline S(x) para cada intervalo.

b) Demuestre que las funciones Spline cumplen las condiciones mínimas.

c) Interpole para x=0.5 y x=1.25 y determine el error cometido en cada caso.

Solución

x	0	1	3/2
y	1	0	1/16

$h_0=1$ $h_1=1/2$

$y[x_0,x_1]=-1$ $y[x_1,x_2]=1/8$

$$\alpha = f'(0) = -4 \quad \beta = f'(3/2) = 1/2$$

$$\begin{bmatrix} 2h_0 & h_0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 \\ 0 & h_1 & 2h_1 \end{bmatrix}\begin{bmatrix} M_0 \\ M_1 \\ M_2 \end{bmatrix} = 6\begin{bmatrix} y[x_0x_1]-\alpha \\ y[x_1x_2]-y[x_0x_1] \\ \beta - y[x_1x_2] \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1/2 \\ 0 & 1/2 & 1 \end{bmatrix}\begin{bmatrix} M_0 \\ M_1 \\ M_2 \end{bmatrix} = \begin{bmatrix} 18 \\ 27/4 \\ 9/4 \end{bmatrix}$$

$$M_0 = 39/4 \quad M_1 = -3/2 \quad M_2 = 3$$

$$a_i = \frac{M_{i+1} - M_i}{6h_i}$$

$$b_i = \frac{M_i}{2}$$

$$c_i = y[x_i x_{i+1}] - \frac{M_{i+1} + 2M_i}{6} h_i$$

$$d_i = y_i$$

$a_0 = -15/8$	$a_1 = 3/2$
$b_0 = 39/8$	$b_1 = -3/4$
$c_0 = -4$	$c_1 = 1/8$
$d_0 = 1$	$d_1 = 0$

f'(0)=-4 f '(1.5)=1/2

Solución

a) $S_0(x)= -15/8x^3+39/8x^2-4x+1 \qquad 0<= x<=1$

$S_1(x)=3/2\ (x-1)^3-3/4\ (x-1)^2+1/8(x-1) \quad 1<= x<=1.5$

ó

$S_1(x)=3/2\ x^3-21/4\ x^2+49/8x -19/8 \quad 1<= x<=1.5$

b) $S_0(1) = S_1(1)= 0 \quad S_0(x_j)=y_j \quad S_1(x_j)=y_j \quad j=0,1,2$

$S'_0(x) = -45/8x^2+39/4x-4 \qquad S'_1(x)= 9/2x^2-21/2x+49/8$

$S'_0(1) = S'_1(1)=1/8$

$S''_0(x) = -45/4x+39/4 \qquad S''_1(x)= 9x-21/2$

$S''_0(1) = S''_1(1)=-3/2$

c)

$S_0(0.5)= -1/64 \qquad f(0.5)= 1/16 \qquad \text{Error1}= |f(0.5)- S_0(0.5)| =0.0781$

$S_1(1.25)= 0.0078 \qquad f(0.5)= 0.0039 \qquad \text{Error2}= |f(1.25)- S_1(1.25)| =0.0039$

Smoothing Splines

- Los Splines están forzados al paso en los nodos.
- Podemos forzar a Splines con derivadas mas suaves, relajando la condición de que el Spline tenga que pasar por los nodos.
- En general se define el Smoothing Spline como la función dos veces diferenciable $\hat{f}$ que minimiza

$$\sum_{i=1}^{n}\{y_i - \hat{f}(x_i)\} + \lambda \int \hat{f}''(x)^2 dx$$

- Para λ=0 tenemos un Spline normal
- Para $\lambda \rightarrow \infty$ tenemos una regresión

Python

- Python incorpora funciones que implementan diferentes tipos de splines mediante la librería **scipy.interpolate**

from scipy import interpolate

- La función **splrep** construye el Spline, algunos de sus parámetros son:
  - **x,y:** Datos que definen la curva f(x).
  - **w:** Array de pesos empleados en Splines con peso.
  - **xb, xe:** Intervalo a aproximar (por defecto x[0],x[-1])
  - **k:** Grado del Spline (por defecto 3)
  - **s:** Factor de suavizado.
- La función devuelve una tupla que contiene:
  - Los nodos
  - Los coeficientes
  - El grado del Spline.
- Una vez creado el spline, se puede evaluar cualquier valor llamando a la función **splev**, cuyos parámetros son:
  - **x:** Array de puntos a evalur.
  - **tck:** Defínción de Spline, usualmente devuelta por **splrep**.

Python

- Probad ahora el siguiente código.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.arange(0, 2*np.pi+np.pi/4, 2*np.pi/8)
y = np.sin(x)
tck = interpolate.splrep(x, y, s=0)
xnew = np.arange(0, 2*np.pi, np.pi/50)
ynew = interpolate.splev(xnew, tck, der=0)

plt.figure()
plt.plot(x, y, 'x', xnew, ynew, xnew, np.sin(xnew), x, y, 'b')
plt.legend(['Linear', 'Cubic Spline', 'True'])
plt.axis([-0.05, 6.33, -1.05, 1.05])
plt.title('Cubic-spline interpolation')
plt.show()
```

Python, Splines cúbicos

- Si lo que se desean es usar Splines cúbicos, existe también la función **CubicSpline**:
  - **x,y:** Datos que definen la curva f(x).
  - **bc_type:** Tupla o cadena que indica el tipo de Spline (Natural, Clamped, etc)
- La función devuelve un objeto de tipo **PPoly.**
- Una vez creado el spline, se puede evaluar cualquier valor llamando directamente al objeto **PPoly** con un array **x** de los puntos a evaluar.

Python

- Probad ahora el siguiente código.

```
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt
x = np.arange(10)
y = np.sin(x)
cs = CubicSpline(x, y)
xs = np.arange(-0.5, 9.6, 0.1)
fig, ax = plt.subplots(figsize=(6.5, 4))
ax.plot(x, y, 'o', label='data')
ax.plot(xs, np.sin(xs), label='true')
ax.plot(xs, cs(xs), label="S")
ax.plot(xs, cs(xs, 1), label="S'")
ax.plot(xs, cs(xs, 2), label="S''")
ax.plot(xs, cs(xs, 3), label="S'''")
ax.set_xlim(-0.5, 9.5)
ax.legend(loc='lower left', ncol=2)
plt.show()
```

Python, Splines cúbicos

- Para usar Smoothing Splines existe la función **UnivariateSpline**:
  - **x,y:** Datos que definen la curva f(x).
  - **w:** Array de pesos empleados en Splines con peso.
  - **k:** Grado del Spline (por defecto 3)
  - **s:** Smoothin coefficient, por defecto s=len(x).
- La función devuelve un objeto de tipo **PPoly.**
- Una vez creado el spline, se puede evaluar cualquier valor llamando directamente al objeto **PPoly** con un array **x** de los puntos a evaluar.

Python

- Probad ahora el siguiente código.

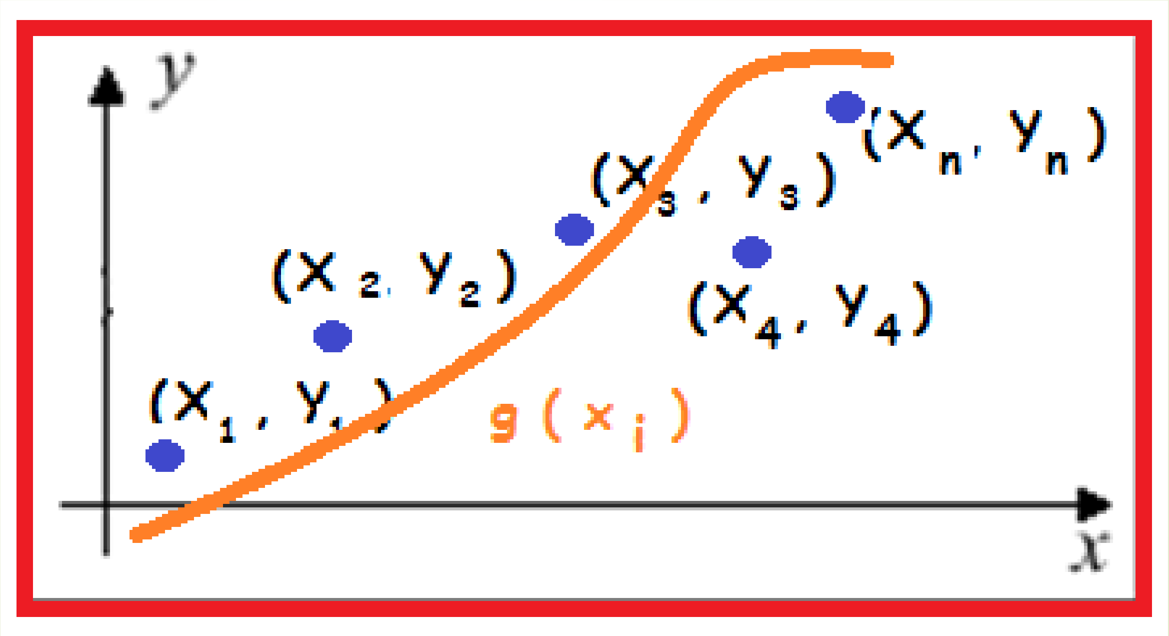
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import UnivariateSpline
x = np.linspace(-3, 3, 50)
y = np.exp(-x**2) + 0.1 * np.random.randn(50)
plt.plot(x, y, 'ro', ms=5)

spl = UnivariateSpline(x, y)
xs = np.linspace(-3, 3, 1000)
plt.plot(xs, spl(xs), 'g', lw=3)

spl.set_smoothing_factor(0.5)
plt.plot(xs, spl(xs), 'b', lw=3)
plt.show()
```

AJUSTE POR MINIMOS CUADRADOS

Dado un conjunto de pares ordenados (x_i, y_i), se

busca una función de aproximación **g**, tal que:

$g(x_i)$ se aproxime a y_i para $i=1, 2, \ldots, n$



De un modo general, una función aproximante dependerá de varias constantes , es decir:

$$g(x) = F(x, c_1, c_2, \ldots, c_k)$$

- Para i=1, 2, ..., n, definimos las desviaciones como:

$$d_i = y_i - F(x_i, c_1, c_2, \ldots, c_k)$$

- La función aproximada deberá ser escogida de forma que tales desviaciones sean pequeñas en valor absoluto.
- Esta función puede ser elegida como una combinación lineal de otras:

$$F(x, c_1, \ldots, c_k) = c_1\varphi_1 + \ldots . c_k\varphi_k$$

- Por ejemplo, la aproximación mediante una recta será:

$$F(x, c_1, c_2) = c_1 x + c_2$$

- El método de los mínimos cuadrados consiste en obtener una función de aproximación, que busca:

$$Minimizar \sum_{i=1}^{n} d_i^2$$

- Se busca entonces, minimizar la suma de los cuadrados de las desviaciones:

$$e(c_1, \ldots, c_k) = \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} \left[y_i - \left(c_1\varphi_1(x_i) + \ldots + c_k\varphi_k(x_i)\right)\right]^2$$

por lo tanto:

$$\nabla e = 0$$

$$\frac{\partial e}{\partial c_j} = 0, \quad j = 1, \ldots, k$$

Aproximación de una recta por mínimos cuadrados:

$$g(x) = c_1 x + c_2$$

$$c_1 \sum_{i=1}^{n} x_i^2 + c_2 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

$$c_1 \sum_{i=1}^{n} x_i + c_2 \sum_{i=1}^{n} 1 = \sum_{i=1}^{n} y_i$$

Forma Matricial del ajuste o regresión por mínimos cuadrados

Sistema sobre-determinado para ajuste de una recta

Escribiendo la ecuación $c_1x + c_2 = y$ para todos los puntos conocidos (x_i, y_i), $i = 1,\ldots,n$ obtenemos un sistema sobre-determinado:

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Forma Matricial del ajuste o regresión por mínimos cuadrados

O: $A\ c = y$

Donde:

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Ecuación normal para el ajuste

El cuadrado de la norma 2 de *r* = *y* – *Ac* es:

$$\begin{aligned}\rho = \|r\|_2^2 = r^T r = (y - Ac)^T (y - Ac) \\ = y^T y - (Ac)^T y - y^T (Ac) + c^T A^T Ac \\ = y^T y - 2y^T Ac + c^T A^T Ac.\end{aligned}$$

La minimización de ρ requiere que:

$$\frac{\partial \rho}{\partial c} = -2A^T y + 2A^T Ac = 0$$

La minimización de ρ requiere que:

$$(A^T A)c = A^T y$$

A esta ecuación se le denomina ECUACION NORMAL.

Factor de regresión:

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_m)^2}{\sum_{i=1}^{n}(y_i - y_m)^2}$$

$\hat{y}_i$ *de la funcion de ajuste*

y_i *de la data*

$$y_m = \frac{\sum_{i=1}^{n} y_i}{n}$$

Factor de regresión:

$$0 \leq R^2 \leq 1$$

- El factor de regresión mide la eficiencia del ajuste,
- Cuando R^2 =1 la función de ajuste coincide con la data.
- Cuando R^2 es cercano a 1 el ajuste se considera aceptable.
- Cuando R^2 es cercano a 0 el ajuste se considera pésimo o deficiente

Reducción a problemas de mínimos cuadrados

- Las funciones:

$$y = ax^b$$
$$y = ae^{bx}$$

- Se puede linealizar:

$$\log(y) = \log(a) + b\ \log(x)$$
$$\log(y) = \log(a) + b\ x$$

Ejemplo

Ajustar los siguientes datos a una recta:

X	0.1	0.4	0.5	0.7	0.7	0.9
Y	0.61	0.92	0.99	1.52	1.47	2.03

Se ajustará a la recta: $\mathbf{y=c_1\,x + c_2}$
se plantea el siguiente sistema
M*C=Y

$$\begin{bmatrix} 0.1 & 1 \\ 0.4 & 1 \\ 0.5 & 1 \\ 0.7 & 1 \\ 0.7 & 1 \\ 0.9 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0.61 \\ 0.92 \\ 0.99 \\ 1.52 \\ 1.47 \\ 2.03 \end{bmatrix}$$

Planteando la ecuación normal:

M^T*M*C=M^T*Y

$$\begin{bmatrix} 0.1 & 0.4 & 0.5 & 0.7 & 0.7 & 0.9 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.1 & 1 \\ 0.4 & 1 \\ 0.5 & 1 \\ 0.7 & 1 \\ 0.7 & 1 \\ 0.9 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.4 & 0.5 & 0.7 & 0.7 & 0.9 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.61 \\ 0.92 \\ 0.99 \\ 1.52 \\ 1.47 \\ 2.03 \end{bmatrix}$$

$$\begin{bmatrix} 2.21 & 3.3 \\ 3.3 & 6 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 4.844 \\ 7.54 \end{bmatrix} \Rightarrow \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1.7646 \\ 0.2862 \end{bmatrix}$$

$$y = 1.7646x + 0.2862$$

$$R^2 = 0.93$$

Ejemplo

Ajustar los siguientes datos a la función **$y=ax^b$**

x	1	1.2	1.6	2
y	1	1.3	1.4	1.7

$$Ln(y)=Ln(a)+b*Ln(x)$$

Y=A+BX

A=0.0514

B=b=0.6874

a=1.0525

$y=1.0525x^{0.6874}$