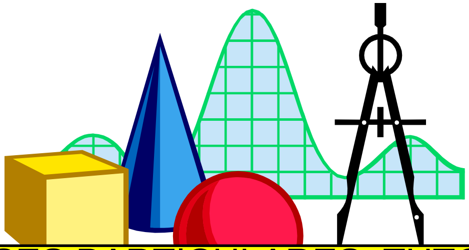
Ajuste e interpolación



Introducción

- En el presente tema estudiamos métodos para obtener el polinomio de aproximación a una función.

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2$$

- La función la podemos conocer
  - explícitamente

$$f(x) = erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Función error (función error de Gauss)

  - una tabla de valores

x	f(x)
x0	f0
x1	f1
x2	f2
x3	f3

Puede ser que nos interese obtener

- **Aproximacion de Taylor**, como método de aproximar una función por un polinomio en las inmediaciones de un punto
- **Polinomio de interpolación**, nos puede interesar obtener el polinomio de

Aproximación de funciones por polinomios

- En numerosas ocasiones es conveniente aproximar una función conocida *f(x)* por una función más simple, como un polinomio *p(x).* Las ventajas y desventajas de esta aproximación son las siguientes:
  - Ventajas
    - Normalmente un polinomio se evalúa por medio de un computador más fácilmente que una función ya que solo son necesarias operaciones básicas, como multiplicaciones y sumas (por ejemplo las funciones trigonométricas son muy costosas de evaluar).
    - Un polinomio se puede derivar e integrar fácilmente y de este modo podemos obtener aproximaciones de la derivada o la integral de la función original, éstas pueden ser muy útiles para programar los métodos.
  - Desventajas
    - Un polinomio tiende a infinito cuando *x* tiene a infinito, por ello no se pueden emplear polinomios para aproximar funciones para *x's* grandes.
    - La función original *f(x)* puede poseer propiedades que el polinomio de aproximación *p(x)* no posee, como ser positiva (un polinomio de orden impar no es positivo), monótona, etc.

Teorema de unicidad: Si x_i, i=0,...,n es un conjunto de n+1 puntos diferentes, para un conjunto f_i, i=0,...,n, solo hay un polinomio *p(x)* de orden n tal que $f_i = p(x_i)$ i=0,...,n.

ningúno que pase por todos los puntos. Si interpolamos con un **grado > n**, nos encontramos con que **no es único**.

El polinomio de Taylor

Supongamos que la función *f(x)* sea infinitamente derivable en un entorno de un punto x_0, su expansión en serie de Taylor se define como:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n + \frac{1}{(n+1)!} f^{(n+1)}(z)(x - x_0)^{n+1}$$

donde *z* es un punto sin determinar situado ente *x* y x_0.

Si eliminamos el último término, la función *f(x)* se puede aproximar por un polinomio *p(x)* de orden (grado) *n* de la forma:

$$f(x) = p(x) = \boxed{f(x_0) + f'(x_0)(x - x_0) + \cdots + \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n}$$

Polinomio de Taylor

(Si x_o=0, el polinomio se llama de Mc-Laurin)

El error al representar una función por un polinomio de Taylor viene dado por el término:

$$|f(x) - p(x)| = \left| \frac{1}{(n+1)!} f^{(n+1)}(z)(x - x_0)^{n+1} \right|$$

El error disminuye con el grado del polinomio empleado (n) y aumenta con la distancia entre *x* y x_o. Además, cuanto más suave sea la función (derivadas más pequeñas) la aproximación es mejor.

- P.ej. la función $f(x) = e^x$ se puede aproximar cerca del punto $x=0$ por el polinomio

valores negativos para un *x* negativo, sin embargo *f(x)* siempre es mayor que cero.

Interpolación polinómica

- El problema que pretendemos resolver en esta sección es aproximar una función por un polinomio utilizando valores de la función $f_i = f(x_i)$ en un conjunto de $n+1$ puntos x_i, $i=0,\ldots,n$. Esto es, encontrar el polinomio de interpolación $p(x)$ de los datos (x_i, f_i) tal que $f_i = p(x_i)$.

  Con $n+1$ puntos se puede calcular un polinomio de interpolación de orden n.

  Este problema suele aparecer cuando, por ejemplo, obtenemos en un **experimento un conjunto de datos** de la forma:

x	f(x)
x0	f0
x1	f1
x2	f2
x3	f3

  y queremos una aproximación polinomial de la función $f(x)$.

- De este modo se puede estimar el valor de la función en otro punto $x4$:
  - Si el polinomio de interpolación se usa para obtener valores de la función dentro del rango de valores de x utilizado para construirlo se llama **interpolación**.
  - Si el polinomio de interpolación se usa para obtener valores de la función fuera del rango de valores de x utilizado para construirlo se llama **extrapolación**.

- **El polinomio de interpolación siempre pasa por los puntos usados para obtenerlo**

con escepticismo ya que las soluciones pueden tener muchos errores

La matriz de Vandermonde

- En general, para que un polinomio:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

satisfaga $f_i = p(x_i)$ ha de cumplir la siguiente condición:

$$a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n = f_i, i = 0, \ldots, n$$

Para obtener los coeficientes a_i del polinomio se ha de resolver el siguiente sistema de ecuaciones:

$$\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{vmatrix} = \begin{vmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \\ f_n \end{vmatrix}$$

- La matriz de este sistema se denomina **matriz de Vandermonde**. **Este método es**

conveniente usar alguno de los métodos siguientes para calcular el polinomio interpolador.

UCM

Diferencias Divididas

- **Un problema** de obtener el polinomio de interpolación por medio de la **matriz de Vandermonde** o por medio del polinomio de **Lagrange** es que si añadimos un dato más (x_{n+1}, f_{n+1}) a la colección de datos ya existente, el polinomio **debe ser recalculado**!

El método de **diferencias divididas** permite obtener el polinomio de interpolación en menor número de operaciones que el método de Lagrange y aprovechando las operaciones realizadas anteriormente. Consideremos el siguiente polinomio de orden n:

$$p_n(x) = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + \cdots + (x - x_0)(x - x_1)\ldots(x - x_{n-2})(x - x_{n-1})a_n$$

La diferencia dividida de orden 1 entre dos puntos s y t se define:

$$f[x_s, x_t] = \frac{f_t - f_s}{x_t - x_s}$$

la de orden 2:

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

Diferencias Divididas

Si sustituimos los datos en el polinomio, llegamos a un sistema de ecuaciones **triangular inferior** en el que las incógnitas son los coeficientes del polinomio:

$$p_n(x) = a_0 + (x - x_0) \cdot a_1 + (x - x_0) \cdot (x - x_1) \cdot a_2 + \cdots + (x - x_0) \cdot (x - x_1) \cdots (x - x_{n-2}) \cdot (x - x_{n-1}) \cdot a_n$$

$$\begin{aligned}
&a_0 && = y_0 \\
&a_0 + (x_1 - x_0)a_1 && = y_1 \\
&a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 && = y_2 \\
&\ldots && \\
&a_0 + (x_n - x_0)a_1 + (x_n - x_0)(x_n - x_1)a_2 + \cdots + (x - x_{n-2})(x - x_{n-1})a_n && = y_n
\end{aligned}$$

Este sistema se resuelve explícitamente empleando un esquema de diferencias divididas.

Si despejamos por sustitución progresiva los coeficientes del polinomio de interpolación del sistema triangular inferior obtenido, cada coeficiente puede asociarse a una diferencia dividida.

$$a_0 = f[x_0] = f_0$$

$$a_n = f[x_0, x_1, \cdots, x_n]$$

Diferencias Divididas

Si hacemos $P_n(x_i)=f_i$, llegamos a un **sistema de ecuaciones triangular inferior** que si resolvemos por sustitución progresiva obtenemos las siguientes expresiones para los coeficientes:

$$\left.\begin{aligned} a_0 &= f[x_0] = f_0 \\ a_1 &= f[x_0, x_1] \\ &\vdots \\ a_n &= f[x_0, x_1, \cdots, x_n] \end{aligned}\right\} \Rightarrow a_i = f[x_0, x_1, \cdots, x_i]$$

Luego, para calcular los coeficientes del polinomio podemos usar las diferencias divididas. Normalmente, para calcular los coeficientes del polinomio **se construye una tabla de diferencias divididas**. Por ejemplo con 5 puntos (x_i, f_i), i=0,...,4 se construye la siguiente tabla:

xi	fi	f[xi, xi+1]	f[xi, xi+1, xi+2]	f[xi, xi+1, xi+2, xi+3]
x0	f0	f[x0, x1]	f[x0, x1, x2]	f[x0, x1, x2, x3]
x1	f1	f[x1, x2]	f[x1, x2, x3]	f[x1, x2, x3, x4]
x2	f2	f[x2, x3]	f[x2, x3, x4]	
x3	f3	f[x3, x4]		

coeficientes a_i:

Diferencias Divididas

- **Veamos un algoritmo para obtener las diferencias divididas o coeficientes *ai*:**

```
diferencias divididas
for i=0,...,n do
   ai = fi
end
for j=1,2,...,n do
   for i=i,j+1,...,n do
```

$$a_i = \frac{a_i - f_{i-1}}{x_i - x_{i-j}}$$

```
   end
   for i=i,j+1,...,n do
     fi = ai
   end
end
```

- **Una vez obtenidos estos coeficientes ya podemos evaluar el polinomio en un punt**

f1=difdiv(x,f,x1) que tenga como entrada *n*+1 datos (*xi*, *fi*) e interpole el valor de la función en los puntos x1:

```
function f1 = difdiv(x,f,x1)
n = size(x,1)+size(x,2)-1;
% crear los coeficientes
a=f;
for j=2:n
    for i=j:n
          a(i) = (a(i)-f(i-1))/(x(i)-x(i-j+1));
    end
    f([j:n]) = a([j:n]);
f=a; end
% evaluar el polinomio en x1
a
f1 = a(1);
for k=1:n-1
    pr = 1;
```



```
end
```


UCM

Diferencias Divididas (otro código)

```
function [f1,p]=difdiv(x,f,x1)
%  [f1,p]=difdiv(x,f,x1)
%  Calcula el valor de la función f1 en los puntos x1, usando el
%  metodo de diferencias divididas para calcular el polinomio p

n=length(x);

% Calcular tabla de diferencias divididas
tabla(:,1)=f;
for i=2:n;
   for j=1:n-i+1
      tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(x(j+i-1)-x(j));
   end
end

p=tabla(1,:);
disp('Los coeficientes del polinomio son: ');  disp(p)

% Calcular los factores (x-x0)*(x-x1)*…
val(1,:)=ones(size(x1));
for k=2:n;
   val(k,:)=val(k-1,:).*(x1-x(k-1));
end
```



```
end
```

Diferencias Divididas (…y otro más) - A --

```
function a=difdiv(x,y)

% este polinomio permite obtener los coeficientes del polinomio de diferencias divididas que interpola
% los datos contenidos el los vectores x e y. Da como resultado un vector fila a con los coeficientes

% miramos cuantos datos tenemos
n=length(x);

% inicializamos el vector de coeficientes con las diferencias de orden 0, es decir los valores de y,
a=y;

% y ahora montamos un bucle, si tenemos n datos debemos calcular n diferencias, como ya tenemos la
% primera, iniciamos el bucle en 2,

for j=2:n
        % en cada iteración calculamos las diferencias de un orden superior,  como solo nos vale la primera
        % diferencia de cada orden empezamos el bucle interior en el valor del exterior j
        for i=j:n
                a(i)=(a(i)-y(i-1))/(x(i)-x(i-j+1));
        end
```

Diferencias Divididas (Evalúa el polinomio) - B --

```
function y=evdif(a,x,x1)

% esta función obtiene el valor de un polinomio de diferencias divididas a
% partir de los coeficientes (a) del polinomio, los puntos (x) sobre los que
% se ha calculado el polinomio y el punto o vector de puntos (x1) para el
% que se quiere calcular el valor que toma el polinomio.

% obtenemos el tamaño del vector de coeficientes del polinomio
n=length(a);

% Construimos un bucle para calcular el valor del polinomio,
y=a(1);
for k=1:n-1
        % calculamos el valor del producto de los binomios que multiplican al
        % coeficiente i
        binprod=1;
        for j=1:k
                binprod=binprod.*(x1-x(j));
```

Diferencias Divididas (Reúne A y B)

```
function y=intdifdiv(x,xp,yp)

% Esta función calcula el valor del polinomio de diferencias divididas que
% interpola los puntos (xp,yp) el el punto, o %los puntos contenidos en x.
% Empleando las funciones, difdiv, para calcular los coeficientes del
% polinomio y evdif para evaluarlo

% llamamos a difdiv

a=difdiv(xp,yp);

% y a continuación llamamos a evdif

Y=evdif(a,xp,x);
```

Diferencias Divididas (Laboratorio 1/2)

Ventajas (respecto a los métodos anteriores):

1. Obtiene $P_n(x)$ con **menos operaciones** que con el método de Lagrange.
2. El polinomio **$P_n(x)$** calculado a partir de (n+1) puntos **sirve para calcular $P_m(x)$**, siendo m>n.

Sean $\{(x_i,f_i)\}_{i=0,\ldots n} \in f(x)$ desconocida,

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

$$\boxed{f_0}$$

$$\boxed{\frac{f_1 - f_0}{x_1 - x_0} \equiv f_{0,1}}$$

Diferencia dividida de <u>orden 1</u> ente x_0 y x_1

$$\boxed{\frac{f_{1,2} - f_{0,1}}{x_2 - x_0} \equiv f_{0,1,2}}$$

Diferencia dividida de <u>orden 2</u> ente x_0 y x_2

$$\boxed{\frac{f_{1,2,\ldots,n} - f_{0,1,\ldots,n-1}}{x_n - x_0} \equiv f_{0,1,\ldots,n}}$$

Diferencia dividida de <u>orden n</u> ente x_0 y x_n

Si disponemos de un nuevo dato (x_{n+1},f_{n+1}), el polinomio $P_{n+1}(x)$ sería:

<u>orden n+1</u> ente x_0 y x_{n+1}

Diferencias Divididas (Laboratorio 2/2)

Esquema para obtener $\{a_i\}_{i=0,1,\dots,n}$ (diferencias divididas de orden i-ésimo entre x_0 y x_i):

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

x_i	f_i	Orden 1	Orden 2	...	Orden n
x_0	f_0	a_0			
x_1	f_1	$\frac{f_1 - f_0}{x_1 - x_0} \equiv f_{0,1}$ a_1			
x_2	f_2	$\frac{f_2 - f_1}{x_2 - x_1} \equiv f_{1,2}$	$\frac{f_{1,2} - f_{0,1}}{x_2 - x_0} \equiv f_{0,1,2}$ a_2		
x_3	f_3	$\frac{f_3 - f_2}{x_3 - x_2} \equiv f_{2,3}$	$\frac{f_{2,3} - f_{1,2}}{x_3 - x_1} \equiv f_{1,2,3}$	...	
...	...	...	...	...	
x_{n-1}	f_{n-1}	$f_{n-1} - f_{n-2}$			
		$x_n - x_{n-1}$	$x_n - x_{n-2}$ $f_{(n-2),(n-1),n}$		$x_n - x_0$ $f_{0,1,\dots,n}$

Ejercicio(1/2)

Dada la siguiente tabla de valores:

	x_0	x_1	x_2
x_i	3.2	2.7	1
f_i	22.0	17.8	14.2
	f_0	f_1	f_2

1) Obtener, mediante el método de diferencias divididas y, redondeando los cálculos a tres decimales, el polinomio interpolador para los datos de la tabla.
2) Si se añade el punto (4.8; 38.3), encontrar el nuevo polinomio interpolador.

Solución:

1) Construyamos la tabla para aplicar el método de diferencias divididas a esos tres puntos.

x_i	f_i	Orden 1	Orden 2
x_0	f_0		
x_1	f_1	$\frac{f_1 - f_0}{x_1 - x_0} \equiv f_{0,1}$	

$$f_0 = 22.0$$

$$f_{0,1} \equiv \frac{f_1 - f_0}{x_1 - x_0} = \frac{17.8 - 22.0}{2.7 - 3.2} = 8.4$$

$$P(x) = 22 + 8.4 \cdot (x - 3.2) + 2.855 \cdot (x - 3.2) \cdot (x - 2.7)$$

Ejercicio(2/2)

2) Añadiendo el nuevo dato a la tabla, no es necesario rehacer los cálculos, basta calcular $f_{2,3}$, $f_{1,2,3}$ y $f_{0,1,2,3}$.

x_i	f_i	Orden 1	Orden 2	Orden 3
x_0	f_0			
x_1	f_1	$\frac{f_1-f_0}{x_1-x_0} \equiv f_{0,1}$		
x_2	f_2	$\frac{f_2-f_1}{x_2-x_1} \equiv f_{1,2}$	$\frac{f_{1,2}-f_{0,1}}{x_2-x_0} \equiv f_{0,1,2}$	
x_3	f_3	$\frac{f_3-f_2}{x_3-x_2} \equiv f_{2,3}$	$\frac{f_{2,3}-f_{1,2}}{x_3-x_1} \equiv f_{1,2,3}$	$\frac{f_{1,2,3}-f_{0,1,2}}{x_3-x_0} \equiv f_{0,1,2,3}$

$$f_{2,3} \equiv \frac{f_3-f_2}{x_3-x_2} = \frac{38.3-14.2}{4.8-1} = 6.342$$

$$f_{1,2,3} \equiv \frac{f_{2,3}-f_{1,2}}{x_3-x_1} = \frac{6.342-2.118}{4.8-2.7} = 2.011$$

$$f_{0,1,2,3} \equiv \frac{f_{1,2,3}-f_{0,1,2}}{x_3-x_0} = \frac{2.011-2.855}{4.8-3.2} = -0.528$$

Por tanto,

$$P(x) = 22 + 8.4\cdot(x-3.2) + 2.855\cdot(x-3.2)\cdot(x-2.7)\boxed{-0.528\cdot(x-3.2)\cdot(x-2.7)\cdot(x-1)}$$

Interpolación por Intervalos (o tramos)

Hasta ahora, hemos visto como interpolar un conjunto de n+1 datos mediante un polinomio de grado n. En muchos caso, especialmente cuando el **numero de datos es suficientemente alto**, los resultados de dicha interpolación **pueden no ser satisfactorios**:

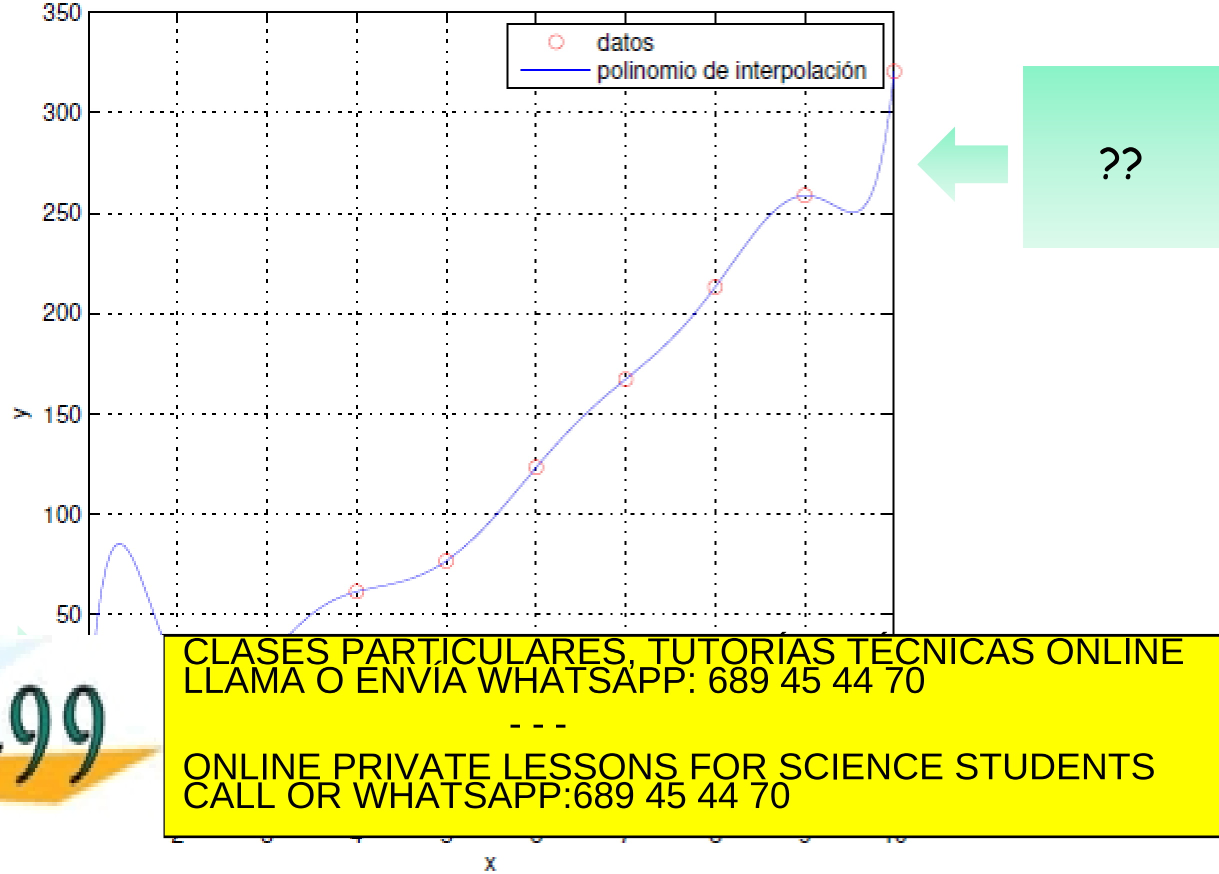
1. La razón es que el grado del polinomio de interpolación crece linealmente con el numero de puntos a interpolar, así por ejemplo para interpolar 11 datos necesitamos un polinomio de grado 10.

2. Desde un punto de vista numérico, este tipo de polinomios pueden dar grandes errores debido al redondeo.

3. Por otro lado, y dependiendo de la disposición de los datos para los que se realiza la interpolación, puede resultar que el polinomio obtenido tome una forma demasiado complicada para los valores comprendidos entres los datos interpolados.

Interpolación por Intervalos

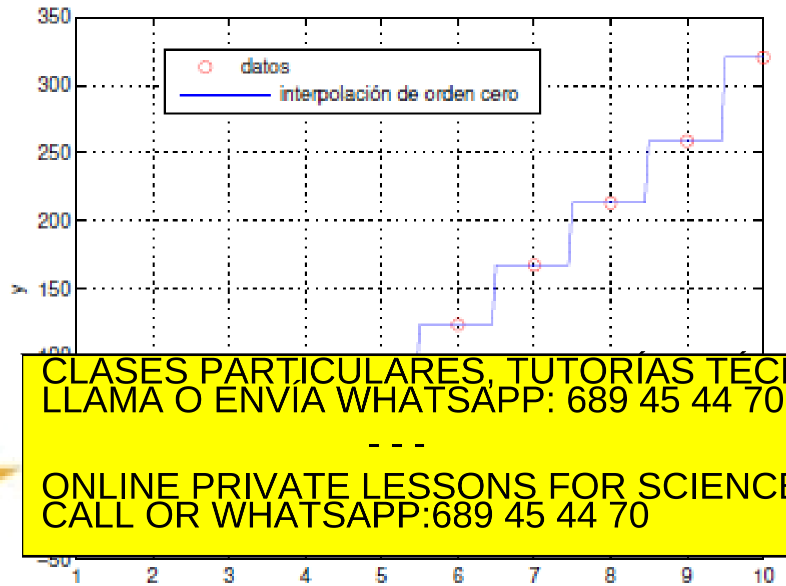
La figura a continuación muestra el polinomio de interpolación de **grado nueve** para un conjunto de **10 datos**. Es fácil darse cuenta, simplemente observando los datos, que no hay ninguna razón que justifique las curvas que traza el polinomio entre los puntos 1 y 2 o los puntos 9 y 10!



Interpolación por Intervalos

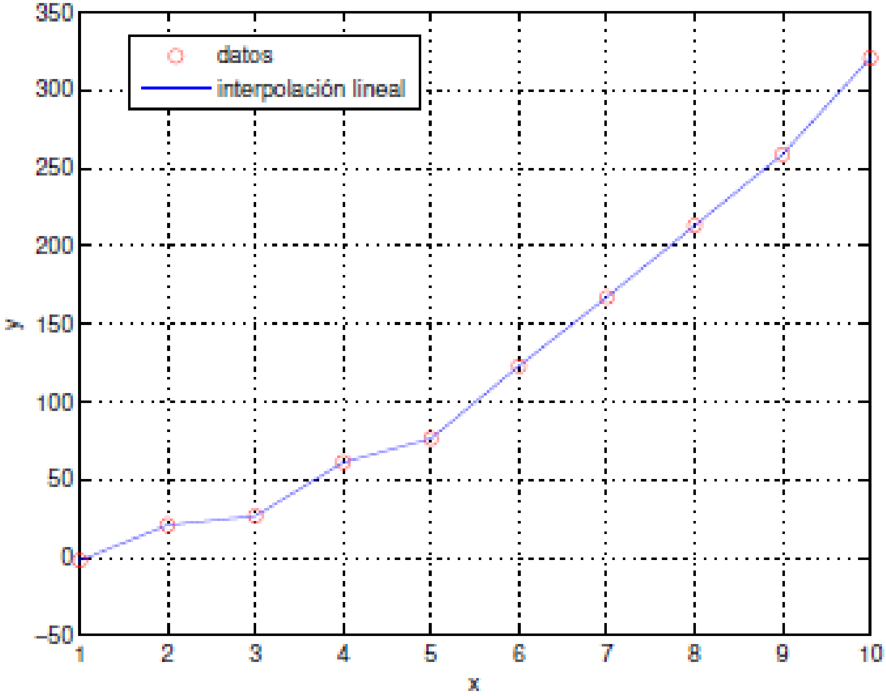
En muchos casos es preferible no emplear todos los datos disponibles para obtener un único polinomio de interpolación. En su lugar, lo que se hace es **dividir el conjunto de datos en varios grupos** (normalmente se agrupan formando intervalos de datos consecutivos) y obtener varios polinomios de menor grado, de modo que cada uno interpole los datos de un grupo distinto. El grado de los polinomios empleados deberá estar, en principio, relacionado con los datos contenidos en cada tramo.

Interpolación de orden cero: si hacemos que **cada intervalo contenga un solo dato**, obtendremos polinomios de interpolación de grado cero, $a_{0i} = y_i$. El resultado, es un **conjunto de escalones** cuya valor varía de un intervalo a otro de acuerdo con el dato representativo contenido en cada tramo.



Interpolación por Intervalos

Interpolación lineal: En este caso, se dividen los datos en **grupos de dos**. Cada par de datos consecutivos se interpola calculando **la recta** que pasa por ellos. La interpolación lineal se emplea en muchas aplicaciones debido a su *sencillez de calculo*.



Interpolación por Splines Cúbicos

Hemos visto como el polinomio interpolador de orden n para un conjunto de n+1 datos puede presentar el inconveniente de complicar excesivamente la forma de la curva entre los puntos interpolados. La interpolación a tramos **simplifica la forma de la curva** entre los puntos pero presenta el **problema de la continuidad** en las uniones entre tramos sucesivos.

Sería deseable encontrar métodos de interpolación que fueran capaces de solucionar ambos problemas simultáneamente. Una buena aproximación a dicha solución la proporcionan los **splines**.

Una **función spline** está formada por varios polinomios, cada uno definido en un intervalo que se unen entre sí obedeciendo a ciertas condiciones de continuidad.

Supongamos que tenemos la siguiente tabla de valores:

x	x0	x1	...	xn
f	f0	f1	...	fn

…y que se debe construir un *spline*

$$\begin{cases} S_0(x) & x \in [x_0, x_1] \\ \vdots & \\ S_{n-1}(x) & x \in [x_{n-1}, x_n] \end{cases}$$

Interpolación por Splines Cúbicos

Los polinomios $S_{i-1}(x)$ y $S_i(x)$ interpolan el mismo valor en el punto x_i para que $S(x)$ sea **continua**. Además $S'(x)$ y $S''(x)$ son también funciones continuas.

$$S_{i-1}(x_i) = f_i = S_i(x_i), (1 \le i \le n-1)$$

$$S'_{i-1}(x_i) = S'_i(x_i), (1 \le i \le n-1)$$

$$S''_{i-1}(x_i) = S''_i(x_i), (1 \le i \le n-1)$$

Grado del polinomio interpolador

Es decir, dos polinomios consecutivos del *spline* y sus m- 1 primeras derivadas, deben tomar los mismos valores en el extremo común.

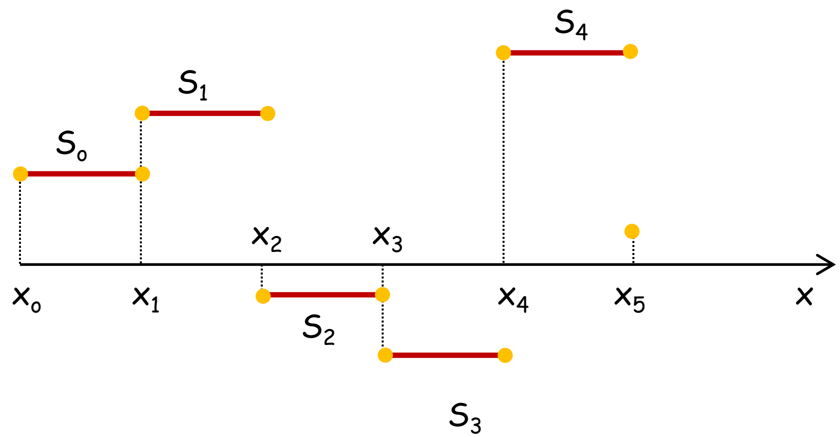
Las **condiciones de continuidad** suministran **(n-1)·m** ecuaciones que, unidas a las **n+1 condiciones de interpolación**, suministran un **total de n·(m+1)-(m-1) ecuaciones**. Este numero es insuficientes para determinar los **(m+1)·n** parámetros

adicionales.

Interpolación por Splines Cúbicos

Los *splines de* ***grado 0*** son funciones constantes por zonas. Una forma explícita de representar un spline de grado 0 es la siguiente:

$$S(x)=\begin{cases} S_o(x)=c_o & x\in[x_o,x_1] \\ S_1(x)=c_1 & x\in[x_1,x_2] \\ \vdots & \vdots \\ S_{n-1}(x)=c_{n-1} & x\in[x_{n-1},x_n] \end{cases}$$



Los intervalos $[x_{i-1},x_i]$ no se intersecan entre si, por lo que no hay ambigüedad en la definición de la función en los nudos. Un *spline de* ***grado 1*** se puede definir por:

$$S(x)=\begin{cases} S_o(x)=a_o x+b_o & x\in[x_o,x_1] \\ S_1(x)=a_1 x+b_1 & x\in[x_1,x_2] \end{cases}$$

UCM

Interpolación por Splines Cúbicos

Vamos a deducir la ecuación para $S_i(x)$ en el intervalo $[x_i, x_{i+1}]$. Definimos $S_i(x)$:

$$S_i(x_i) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad en \quad [x_i, x_{i+1}] \quad (i = 0, \ldots, n-1)$$

Impongamos las condiciones anteriores (definiendo la constante $h_i = x_{i+1} - x_i$) y consideramos que hay que determinar **4·n coeficientes** [(m+1)·n] para los <u>n</u> <u>polinomios cúbicos</u>:

- En cada sub-intervalo $[x_i, x_{i+1}]$ tenemos 2 condiciones de interpolación por un total de **2·n condiciones**.

$$S(x_i) = f_i \;\; \text{y} \;\; S(x_{i+1}) = f_{i+1}$$

- Debemos exigir la continuidad de S(x)': 1 condición por nudo → **n-1 condiciones**

$$S'_{i-1}(x_i) = S'_i(x_i)$$

- Lo mismo vale para la continuidad de S(x)'': 1 condición por nudo → **n-1 condiciones**

$$S''_{i-1}(x_i) = S''_i(x_i)$$

$$S_{n-1}(x_n) = a_{n-1} + b_{n-1}h_{n-1} + c_{n-1}h_{n-1}{}^2 + d_{n-1}h_{n-1}{}^3 = f_{n-1}$$

$$S_i(x_{i+1}) = a_i + b_i \cdot h_i + c_i \cdot h_i^2 + d_i \cdot h_i^3 = S_{i+1}(x_{i+1}) = a_{i+1}$$

$$S'_i(x_{i+1}) = b_i + 2 \cdot c_i \cdot h_i + 3 \cdot d_i \cdot h_i^2 = S'_{i+1}(x_{i+1}) = b_{i+1}$$

Por tanto tenemos:

Estas dos ecuaciones que faltan las podemos obtener **imponiendo condiciones a los polinomios en los bordes**, ya que aquí no existen las condiciones de empalme.

Interpolación por Splines Cúbicos

Dependiendo del tipo de condiciones de contorno que impongamos se obtienen distintos tipos de splines:

Splines con valor conocido en la primera derivada de los extremos $S'(x_o)=f'_o$ y $S'(x_n)=f'_n$.

Splines naturales, cuando no se dispone de condiciones en las fronteras, se suele exigir que la derivada segunda de los polinomios se anule en los extremos, de esta forma se consigue que el comienzo y el fin sean muy suaves.

Es posible despejar y reordenar $[c_i=S''(x_i)]$ las ecuaciones anteriores hasta obtener un sistema de ecuaciones **tridiagonal con diagonal dominante**.
Para el caso del **Spline Natural** el resultado es el siguiente:

$$c_0 = c_{n-1} = 0; \qquad Condiciones\,naturales$$

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & \\ h_1 & 2(h_1+h_2) & h_2 & \\ & \ddots & \ddots & \ddots \\ & & h_{n-3} & 2(h_{n-3}+h_{n-2}) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-2} \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \end{bmatrix}$$

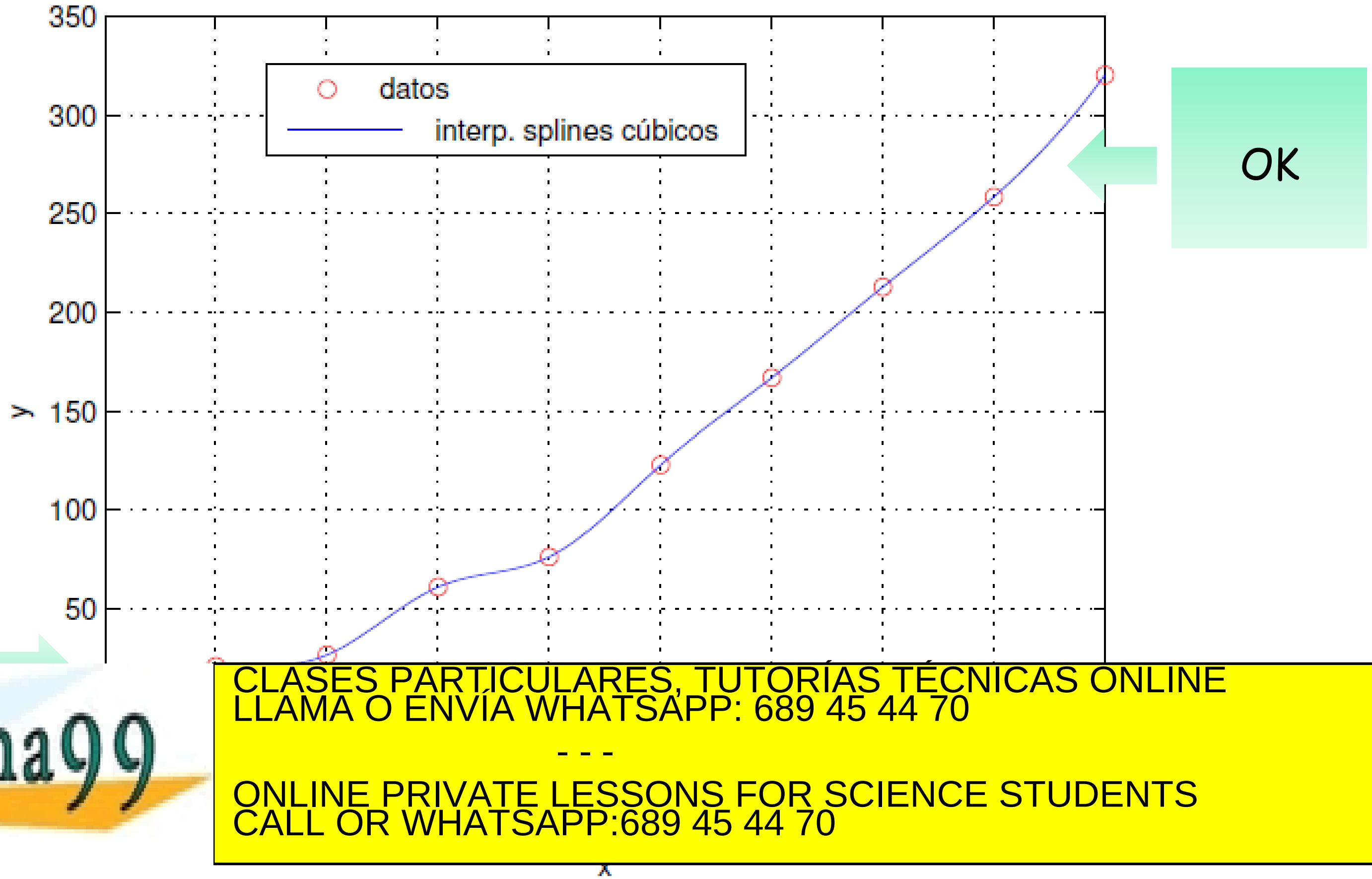
$$\boxed{\begin{aligned} c_0 &= S''(x_0) = 0 \\ c_{n-1} &= S''(x_{n-1}) = 0 \end{aligned}}$$

$$g_i = \frac{3}{h_i}(f_{i+1}-f_i) - \frac{3}{h_{i-1}}(f_i - f_{i-1})$$

$$S_i(x) = \frac{c_i}{6h_i}(x_{i+1}-x)^3 + \frac{c_{i+1}}{6h_i}(x-x_i)^3 + \left[\frac{f_{i+1}}{h_i} - \frac{c_{i+1}h_i}{6}\right](x-x_i) + \left[\frac{f_i}{h_i} - \frac{c_i h_i}{6}\right](x_{i+1}-x)$$

Interpolación por Splines Cúbicos

La figura a continuación muestra el resultado de interpolar mediante un **spline cubico**. Es fácil observar como ahora los polinomios de interpolación dan como resultado una **curva suave** en los datos interpolados y en la que además las curvas son también suaves, **sin presentar variaciones extrañas**, para los puntos contenidos en cada intervalo entre los datos.

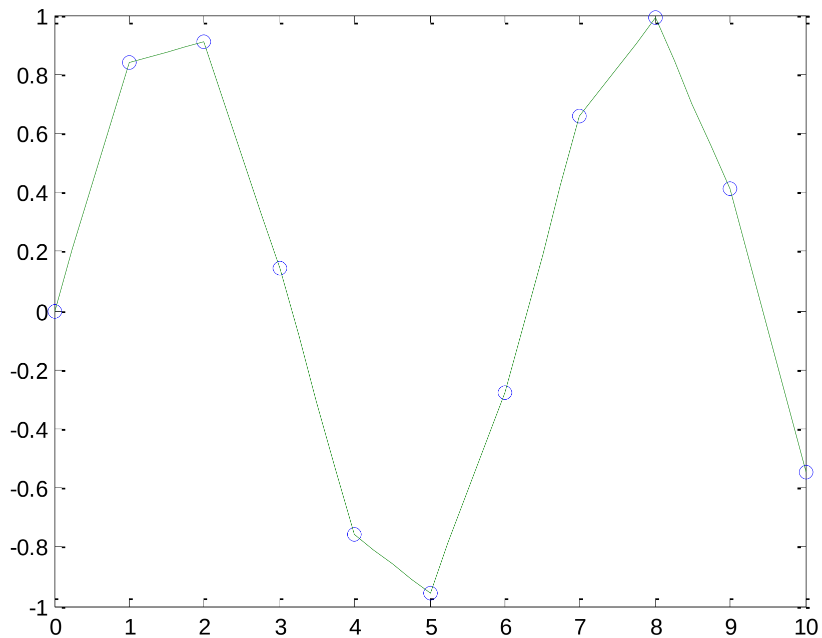


Funciones de interpolación del sistema MATLAB

- MATLAB aporta la función y1=**interp1**[x,y,x1,'tipo'] que produce una interpolación lineal (tipo=linear), cúbica (tipo=cubic) o por splines cúbicos (tipo=spline). Esta función tiene una serie de restricciones (véase *help* interp1). Hay una función llamada spline que además de interpolar valores por medio del método de los splines cúbicos puede devolver, como parámetro de salida, la función spline.

- **Ejemplo:**

```
>> x = 0:10;
>> y = sin(x);
>> xi = 0:.25:10;
>> yi = interp1(x,y,xi);
>> plot(x,y,'o',xi,yi)
```



Opcional:

$\int [S''(x)]^2 dx = \int [f''(x)]^2 dx$ Curvature of a curve

Aproximación por Mínimos Cuadrados

- El problema de la aproximación por mínimos cuadrados es diferente a la búsqueda del polinomio de interpolación estudiada en la sección anterior. **La aproximación por mínimos cuadrados consiste en encontrar un polinomio de un orden dado que se aproxime más a un conjunto de datos experimentales.**

- Ahora supongamos que *p(x)* es un polinomio de orden *n* y queremos determinar sus coeficientes. Si los datos se han tomado experimentalmente **pueden tener errores** y por ello lo normal es tomar más de n+1 datos.

 Tenemos **m puntos $x_1,...,x_m$, donde $m \geq n+1$** y al menos n+1 de los puntos son distintos. Sean $f_1,...,f_m$ los valores aproximados de la función *f(x)*, no necesariamente un polinomio, en esos puntos. Entonces, queremos encontrar un polinomio de orden *n*, $p(x)=a_0+a_1x+...+a_nx_n$ tal que la expresión:

$$\sum_{i=1}^{m} w_i\,(f_i - p(x_i))^2$$

Esta cantidad tiene la ventaja de que su valor es **siempre positivo** con independencia de que la diferencia sea positiva o negativa.

es mínima para todos los polinomios de orden *n*. Es decir,

queremos encontrar los coeficientes a_i del polinomio tal que la suma con pesos de los errores (distancias) f_i-$p(x_i)$ es mínima.

Aproximación por mínimos cuadrados

Ajuste a un polinomio de orden 0 o constante

Por ejemplo, supongamos que hemos tomado **varias medidas de la longitud** de un objeto $l_1,\ldots,l_m$. En este caso el **polinomio** que queremos obtener es **constante** $p(x)=a_0$. El objetivo es minimizar lo siguiente:

$$g(a_0)=\sum_{i=1}^{m} w_i (l_i - a_0)^2$$

Sabemos que **el mínimo de esta función tiene** $g'(a_0)=0$ y $g''(a_0)\geq 0$.

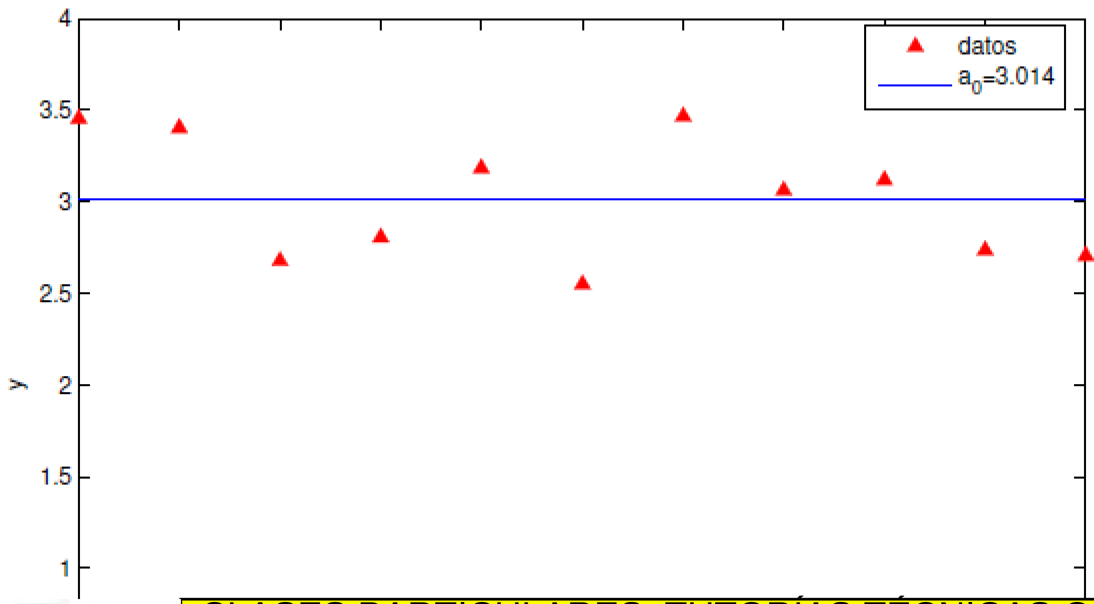
$$g'(a_0)=-2\sum_{i=1}^{m} w_i (l_i - a_0);\, g'(a_0)=-2\sum_{i=1}^{m} w_i (l_i - a_0)=0 \Rightarrow a_0=\frac{\sum_{i=1}^{m} w_i l_i}{\sum_{i=1}^{m} w_i}$$

$$g''(a_0)=2\sum_{i=1}^{m} w_i \Rightarrow g''(a_0)\geq 0$$

Aproximación por mínimos cuadrados

Aproximar un conjunto de valores por un polinomio de **grado cero**, es tanto como suponer que la variable y permanece constante para cualquier valor de x.

Las diferencias observadas deberán deberse entonces a errores aleatorios experimentales, y la mejor estima del valor de y será precisamente el **valor medio** de los valores disponibles. La figura muestra el resultado de calcular el polinomio de mínimos cuadrados de grado cero para un conjunto de datos.



UCM

Aproximación por mínimos cuadrados

Ajuste a un polinomio de orden 1 o lineal

Si tomamos un conjunto de medidas experimentales y suponemos que mantienen una relación lineal podemos obtener el polinomio lineal que los relaciona aproximando por mínimos cuadrados los datos a una **recta** ***$p(x)=a_0+a_1x$***. En este caso la función que debemos minimizar respecto a los coeficientes a_0 y a_1 es:

$$g(a_0,a_1)=\sum_{i=1}^{m} w_i(f_i-a_0-a_1x_i)^2$$

f_i

Sabemos que **el mínimo** de esta función se obtiene cuando las derivadas parciales respecto a ambos coeficientes son iguales a cero:

$$\frac{\partial g}{\partial a_0}=-2\sum_{i=1}^{m} w_i(l_i-a_0-a_1x_i)=0$$

$$\frac{\partial g}{\partial a_1}=-2\sum_{i=1}^{m} w_ix_i(l_i-a_0-a_1x_i)=0$$

Reordenando estas expresiones obtenemos el siguientes **sistema**

$$(\sum_{i=1}^{m} w_i)a_0+(\sum_{i=1}^{m} w_ix_i)a_1=\sum_{i=1}^{m} w_if_i$$

Aproximación por mínimos cuadrados

Ajuste a un polinomio de orden n (generalización)

Si tomamos un conjunto de medidas experimentales y suponemos que mantienen una relación polinomial de orden *n*, podemos aproximar por mínimos cuadrados los datos a un polinomio $p(x) = a_0+a_1x+...+a_nx_n$. En este caso la función que debemos minimizar es:

$$g(a_0, a_1, \ldots, a_n) = \sum_{i=1}^{m} w_i (a_0 + a_1 x_i + \cdots a_n x_i^n - f_i)^2$$

Sabemos que el mínimo de esta función se obtiene cuando las <u>derivadas parciales respecto a los coeficientes son iguales a cero</u>:

$$\frac{\partial g(a_0, a_1, \ldots, a_n)}{\partial a_j} = \sum_{i=1}^{m} w_i x_i^j (a_0 + a_1 x_i + \cdots a_n x_i^n - f_i) \quad = 0; j = 0,1\ldots,n$$

Reordenando estas expresiones obtenemos el siguiente **sistema de *n*+1 ecuaciones** que nos

$$\begin{bmatrix} s_0 & s_1 & \cdots & s_n \\ s_1 & s_2 & & s_{n+1} \\ \vdots & & \ddots & \vdots \\ s_n & s_{n+1} & \cdots & s_{2n} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

$$c_j = \sum_{i=1} w_i x_i f_i$$

Aproximación por mínimos cuadrados

Escribe una función MATLAB a=**mc**(x,f,n,w) que tenga como entrada los datos (*xi, fi, wi*) y el orden n del polinomio que se debe ajustar por mínimos cuadrados. La salida de la función es un vector con los coeficientes del polinomio desde el menor al mayor. Si no se introduce vector de pesos se toma por defecto w_i=1.

```
function a = mc(x,f,n,w)
%  a = mc(x,f,n,w) Calcula los
   coeficientes de un polinomio
   de orden n por minimos
   cuadrados partiendo de los
   datos que se introducen en x y
   f, con peso w.
%  Variables de entrada:
%       x,f,w: datos y pesos
%       n: orden del polinomio
   que se debe ajustar
%  Variables de salida:
%       a: coeficientes del
   polinomio
%
if (nargin<4)
```

```
if (m>=n+1) %Si hay suficientes datos
 for j=1:2*n+1, % crear el vector s
   s(j) = 0;
   for i=1:m
        s(j) = s(j)+w(i)*x(i)^(j-1);
   end
 end
 for j=1:n+1, % crear el vector c
   c(j) = 0;
   for i=1:m
        c(j) = c(j)+w(i)*f(i)*x(i)^(j-1);
   end
 end
 for i=1:n+1, %crear la matriz del sistema
   for j=1:n+1
   size(M), a = M\c', %resolver el sistema
 end
```

Mínimos cuadrados en Matlab

MATLAB aporta la función p=**polyfit**(x,y,n) para ajustar los puntos (x_i,y_i) a un polinomio de grado n. El vector p se carga con los coeficientes del polinomio desde el más significativo al menos $p(x) = a(n+1)+a(n)x+...a(1)x_n$.

Ejercicio: Obtener en los puntos [10.0:1:50.0] los valores de la función f(x)=x^2.

A continuación sumarle un ruido y obtener el ajuste de los datos resultantes a un polinomio de orden 2.

Usaremos la función **polyfit**:

```
>> x = [10.0:1.0:50.0]; f = x.^2; f = f+rand(size(f));
>> p2 = polyfit(x,f,2);
```

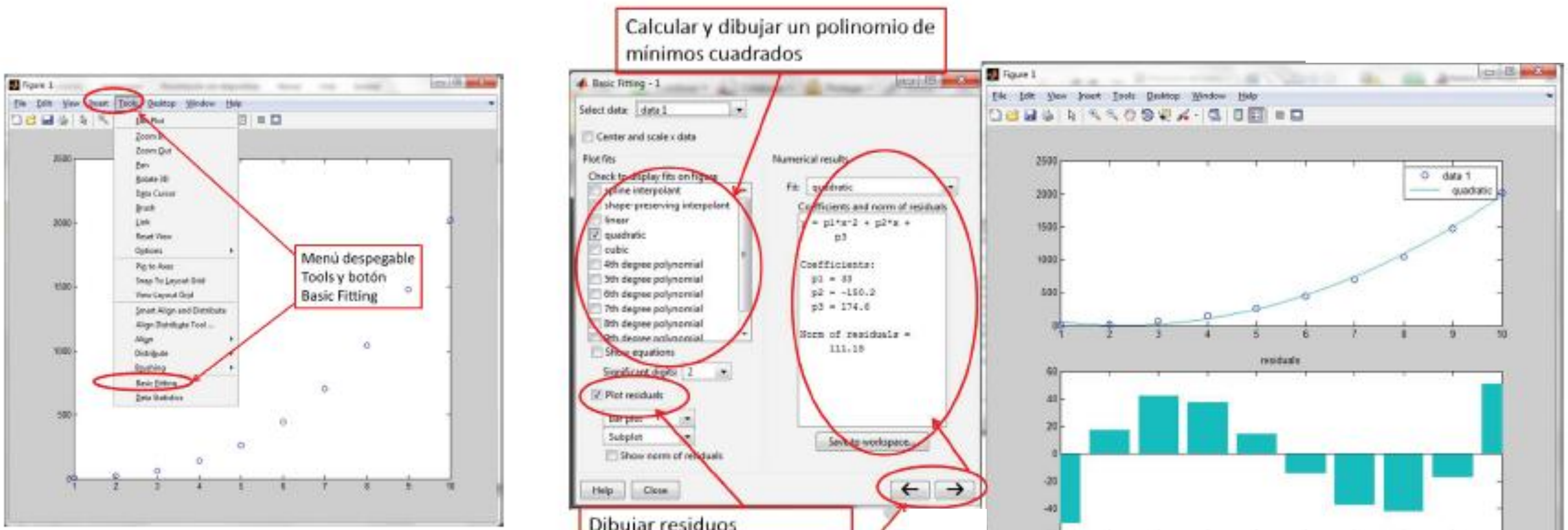
A continuación, podemos emplear el comando **polyval**, para obtener en valor del polinomio de mínimos cuadrados obtenido en cualquier punto. En particular, si lo aplicamos a los datos x:

```
>> yhat=polyval(p2,x)
```

Por ultimo, podemos calcular el error cometido por el polinomio $e_i=|p(x_i)- f_i|$:

Mínimos cuadrados en Matlab

Además del comando **polyfit** Matlab permite ajustar un polinomio por mínimos cuadrados a un conjunto de datos a través de la **ventana grafica** de Matlab. Para ello, es suficiente representar los datos con el comando **plot(x,y)**. Una vez que Matlab muestra la ventana grafica con los datos representados, se selecciona en el menú desplegable **tools** la opción **Basic Fitting**. Matlab abre una segunda ventana que permite seleccionar el polinomio de mínimos cuadrados que se desea ajustar a los datos:



UCM