6 Interpolación y ajuste polinómico

En Física y otras ciencias con frecuencia es necesario trabajar con conjuntos discretos de valores de alguna magnitud que depende de otra variable. Pueden proceder de muestreos, de experimentos o incluso de cálculos numéricos previos. En ocasiones, para utilizar estos valores en cálculos posteriores es preciso "darles forma" de función, es decir: es preciso disponer de una función dada por una expresión matemática que "coincida" con dichos valores. En Matemáticas también es necesario en ocasiones, sustituir o aproximar una función dada por una más simple o con alguna estructura especial, de forma que ambas tomen los mismos valores en un cierto conjunto de puntos.

En algunas ocasiones, lo anterior no es posible o el resultado no posee las propiedades deseadas. En estos casos se puede recurrir al **ajuste**

6.1 Interpolación global ó de Lagrange

Cuando se trata de encontrar un polinomio (de algún grado determinado) que tome unos valores dados en un conjunto discreto de puntos se habla de **interpolación polinómica global** o **interpolación de Lagrange**. El problema que se plantea es:

> Dados un soporte de n puntos distintos $S = \{x_1, x_2, \ldots, x_n\}$ con $a \leq x_1 < x_2 < \ldots x_n \leq b$, y n valores y_1, y_2, $\ldots y_n \in \mathbb{R}$, hallar un polinomio de grado $n-1$
>
> $$p(x) = c_1 x^{n-1} + c_2 x^{n-2} + \cdots + c_{n-1} x + c_n$$
>
> que verifique
>
> $$p(x_i) = y_i \quad \forall i = 1, 2, \ldots n$$

Para calcular los coeficientes de este polinomio con MATLAB se usa la orden

```
coef = polyfit(x,y,n-1)
```

`x` es un vector que contiene los puntos del soporte, x_i.

`y` es un vector que contiene los valores y_i

`n` es el número de puntos (i.e. $n-1$ es el orden del polinomio).

La orden `polyfit` calcula los coeficientes del polinomio de interpolación resolviendo el sistema de Vandermonde.

Una vez calculados los coeficientes, se puede evaluar el polinomio en un punto z mediante la orden

```
pz = polyval(coef,z)
```

`coef` es un vector que contiene los coeficientes de un polinomio en orden de mayor a menor potencia.

`z` es el punto en que se desea evaluar el polinomio (puede ser un vector).

`pz` la salida de esta función es el valor del polinomio en el punto `z` (si `z` es un vector, `pz` será un vector de la misma dimensión que `z`).

Ejemplo 6.1
Calcular el polinomio de interpolación de grado 2 que pasa por los puntos

$$(1,-3),\ (2,1),\ (3,3)$$

Representar su gráfica en el intervalo $[0,7]$, señalando con marcadores los puntos interpolados y dibujando también los ejes coordenados. (Escribir un *script* con las órdenes siguientes).

```
%
%  Calcular y representar graficamente el polinomio de interpolacion
%  global que pasa por tres puntos dados
%
  x = [1, 2, 3]
  y = [-3, 1, 3];
  coef = polyfit(x, y, 2);

  z = linspace(0,7);
  pz = polyval(coef, z);
  plot(z, pz, 'LineWidth',1.1)
  hold on
  plot([-1,7],[0,0], 'k','LineWidth',1.1)
  plot([0,0],[-10,6],'k','LineWidth',1.1)
  plot(x,y,'r.','MarkerSize',20)
  axis([-2, 8, -11, 7])
  hold off
```

Ejercicio 6.1 Este ejercicio pretende mostrar que el procedimiento de **interpolación global** es, en general inestable, ya que los polinomios tienden a hacerse oscilantes al aumentar su grado y eso puede producir grandes desviaciones sobre los datos (fenómeno de Runge).

Calcula el polinomio de grado 10 que interpola los valores:

```
x=( 0,  2,  3,   5,  6,  8,  9,   11, 12, 14, 15)
y=(10, 20, 30, -10, 10, 10, 10.5, 15, 50, 60, 85)
```

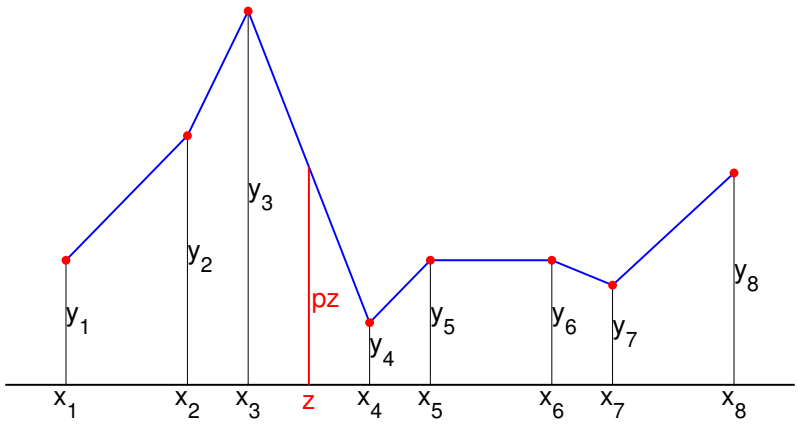
Dibuja su gráfica, así como los puntos con marcadores, y observa las inestabilidades cerca de los extremos. Escribe un *script* o M-función que lleve a cabo todo lo que se pide.

6.2 Interpolación lineal a trozos

Como se ha puesto de manifiesto en el ejercicio 6.1, la interpolación polinómica global es inestable cuando el número de puntos es elevado. En la práctica se usan con frecuencia procedimientos de interpolación polinómica a trozos. En particular, cuando se interpola mediante polinomios de grado 1 a trozos (poligonales), se habla de **interpolación lineal a trozos**.

Dados un soporte de n puntos distintos $S = \{x_1, x_2, \ldots, x_n\}$ con $a \leq x_1 < x_2 < \ldots x_n \leq b$, y n valores y_1, y_2, $\ldots y_n \in \mathbb{R}$, el interpolante lineal a trozos asociado es la poligonal representada en la Figura ??, que viene definida, en cada intervalo de la forma $[x_i, x_{i+1}]$ por

$$p_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i), \quad i = 1, 2, \ldots n-1$$



Para calcular el valor de la función interpolante en un punto z intermedio entre x_1 y x_n, MATLAB dispone de la función

```
pz = interp1(x,y,z)
```

`x` es un vector que contiene los puntos del soporte, x_i.

`y` es un vector que contiene los valores y_i

`z` es el punto en que se desea evaluar el interpolante (puede ser un vector).

`pz` la salida de esta función es el valor del polinomio en el punto `z` (si `z` es un vector, `pz` será un vector de la misma dimensión que `z`).

Ejemplo 6.2
El fichero de datos `censo.txt` contiene dos columnas que corresponden al censo de EEUU entre los años 1900 y 1990 (en millones de personas).

Representar gráficamente la evolución del censo en esos años y estimar la población que había en el año 1956. (Escribir un *script* con las órdenes siguientes).

```
%
%  Interpolacion lineal a trozos de un conjunto de datos
%  del censo de los EEUU durante el siglo XX
%

% -------- lectura de los datos
M = load('censo.txt');
anno = M(:,1);
pobl = M(:,2);

% -------- representacion grafica
plot(anno, pobl, 'LineWidth',1.1)
grid on
legend('Evolucion del censo de EEUU')

% -------- poblacion en 1956
p56 = interp1(anno, pobl, 1956);

% -------- inclusion del resultado en la gráfica
texto = [num2str(p56),' millones de personas'];
text(1910, 230, 'Poblacion en 1956:')
text(1910, 220, texto)
shg
```

Ejercicio 6.2 Se consideran los mismos valores que en el ejercicio 6.1.

Representa gráficamente (juntos) el polinomio de interpolación global y el interpolante lineal a trozos. Representa también los puntos del soporte de interpolación, mediante marcadores. Añade las leyendas adecuadas para que se pueda identificar cada curva adecuadamente.

Calcula y muestra el valor interpolado para $z = 1$ por cada uno de los procedimientos.

Escribe un *script* o M-función que lleve a cabo todo lo que se pide.

Ejercicio 6.3 (para valientes) Escribe tu propia función

```
function [yz] = interpola(x, y, z)
```

que haga lo mismo que `interp1(x, y, z)`.

6.3 Interpolación mediante *splines* cúbicos

La interpolación mediante *splines* busca interpolar un conjunto de datos mediante funciones polinómicas a trozos que no sean sólo continuas, sino que sean continuamente derivables hasta un cierto orden. Para ello es necesario, naturalmente, aumentar los grados de libertad de que disponemos, es decir, el número de coeficientes que podemos elegir, i.e. el grado de los polinomios.

Concretamente, mediante *splines* cúbicos, se pueden obtener funciones interpolantes de clase C^2.

El problema ahora es:

> Dados un soporte de n puntos distintos $S = \{x_1, x_2, \ldots, x_n\}$ con $a \leq x_1 < x_2 < \ldots x_n \leq b$, y n valores y_1, y_2, $\ldots y_n \in \mathbb{R}$, encontrar una función $s \in C^2([a, b])$ que verifique
>
> $$s(x_i) = y_i, \quad i = 1, 2, \ldots n$$
>
> y tal que, restringida a cada subintervalo $[x_i, x_{i+1}]$, sea un polinomio de grado tres.

Para calcular *splines* cúbicos, MATLAB dispone de la función `spline`, que se puede usar de dos formas distintas.

```
pz = spline(x, y, z)
```

`x` es un vector que contiene los puntos del soporte, x_i.

`y` es un vector que contiene los valores y_i

`z` es el punto en que se desea evaluar el *spline* (puede ser un vector).

`pz` la salida de esta función es el valor del *spline* en el punto `z` (si `z` es un vector, `pz` será un vector de la misma dimensión que `z`).

Otra forma de usar la función `spline` es:

```
coef = spline(x, y)
```

`x` es un vector que contiene los puntos del soporte, x_i.

`y` es un vector que contiene los valores y_i

`coef` es una estructura de datos que contiene los coeficientes que definen el *spline* (no es necesario, a los efectos de este curso, saber más sobre esta estructura)

Una vez calculados los coeficientes del *spline* con esta función, para evaluarlo en un punto z hay que usar la orden

```
pz = ppval(coef, z)
```

`coef` es la estructura de datos previamente calculada.

`z` es el punto en que se desea evaluar el *spline* (puede ser un vector).

`pz` la salida de esta función es el valor del *spline* en el punto `z` (si `z` es un vector, `pz` será un vector de la misma dimensión que `z`).

Ejemplo 6.3

El fichero `DatosSpline.dat` contiene una matriz con dos columnas, que corresponden a las abscisas y las ordenadas de una serie de datos.

Leer los datos del fichero y calcular y dibujar juntos el polinomio de interpolación global y el spline cúbico que interpolan dichos valores, en un intervalo que contenga todos los puntos del soporte. (Escribir un *script* con las órdenes siguientes).

```
% ------------- lectura de los datos
datos = load('DatosSpline.txt');
x     = datos(:,1);
y     = datos(:,2);

% ------------- calculo de los interpolantes
z = linspace(min(x),max(x));
c = polyfit(x,y,length(x)-1);
p = polyval(c,z);
s = spline(x,y,z);

% ------------- graficas
plot(z,p,'b--','LineWidth',1.1);
hold on
plot(z,s,'g',  'LineWidth',1.1);
plot(x,y,'r.', 'MarkerSize',15)
legend('Interpolante global', 'Spline cubico', 'Puntos del soporte')
hold off
shg
```

Ejemplo 6.4

En este ejemplo se muestra el uso de la función `spline` en la segunda de las formas que se ha explicado antes: en una primera etapa se calculan los coeficientes del *spline* y se almacenan en una variable, y en una segunda etapa se evalúa el spline en los puntos deseados, utilizando la función `ppval`. Esto permite no repetir el cálculo de los coeficientes (que siempre son los mismos) cada vez que se desea evaluar el *spline*.

Con los datos del mismo fichero `DatosSpline.dat` se pide:

a) Definir una función anónima que represente el *spline* cúbico $s(x)$ que interpola dichos valores, es decir, que calcule $s(x)$ para cualquier x.

b) Calcular

$$V = \int_0^{40} s(x)\,dx$$

(Escribir un *script* con las órdenes siguientes).

```
% ------------- lectura de los datos
datos = load('DatosSpline.txt');
x     = datos(:,1);
y     = datos(:,2);

% ------------- calculo de los coeficientes del spline
coef = spline(x,y);

% ------------- definicion de la funcion anonima
s = @(z) ppval(coef, z);

% ------------- calculo de la integral
V = integral(s, 0, 40);

% ------------- impresion del resultado
fprintf('\n')
fprintf('El valor de la integral es: %15.9f \n',V)
```

Ejercicio 6.4 Se consideran los mismos valores que en el ejercicio 6.1 y el ejercicio 6.2.

Representa gráficamente (juntos) el polinomio de interpolación global, el interpolante lineal a trozos y el spline cúbico. Representa también los puntos del soporte de interpolación, mediante marcadores. Añade las leyendas adecuadas para que se pueda identificar cada curva adecuadamente.

Calcula el valor interpolado para $z = 1$ por cada uno de los procedimientos.

Escribe un *script* o M-función que lleve a cabo todo lo que se pide.

Ejercicio 6.5 (Prescindible. Para ampliar conocimientos) Cuando se calcula un *spline* cúbico con la función `spline` es posible cambiar la forma en que éste se comporta en los extremos. Para ello hay que añadir al vector `y` dos valores extra, uno al principio y otro al final. Estos valores sirven para imponer el valor de la pendiente del spline en el primer punto y en el último. El spline así construido se denomina *sujeto*.

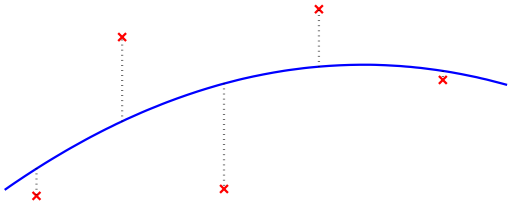
En este ejercicio se trata de calcular y dibujar una aproximación de la función $\text{sen}(x)$ en el intervalo $[0, 10]$ mediante la interpolación con dos tipos distintos de spline cúbico y comparar estos resultados con la propia función, utilizando para ello un soporte regular con 8 puntos. Hay por lo tanto que dibujar tres curvas en $[0, 10]$:

1. La curva $y = sen(x)$.
2. El spline que calcula MATLAB por defecto (denominado *not-a-knot*).
3. El spline *sujeto* con pendiente $= -1$ en $x = 0$ y pendiente $= 5$ en $x = 10$.

6.4 Mejor aproximación en el sentido de los mínimos cuadrados de un conjunto discreto de puntos mediante polinomios

La técnica de interpolación que hemos explicado antes requiere que la función que interpola los datos pase exactamente por los mismos. En ocasiones esto no da resultados muy satisfactorios, por ejemplo si se trata de muchos datos. También sucede con frecuencia que los datos vienen afectados de algún error, por ejemplo porque provienen de mediciones. No tiene mucho sentido, pues, obligar a la función que se quiere construir a «pasar» por unos puntos que ya de por sí no son exactos.

Otro enfoque diferente es construir una función que no toma exactamente los valores dados, sino que «se les parece» lo más posible, por ejemplo minimizando el error, medido éste de alguna manera.



Cuando lo que se minimiza es la suma de las distancias de los puntos a la curva (medidas como se muestra en la figura) hablamos de **ajuste por mínimos cuadrados**. Veremos solamente cómo se puede hacer esto con MATLAB en algunos casos sencillos.

La función `polyfit` usada ya para calcular el polinomio de interpolación global sirve también para ajustar unos datos por un polinomio de grado dado:

```
coef = polyfit(x, y, m)
```

`x y` son dos vectores de la misma dimensión que contienen respectivamente las abscisas y las ordenadas de los N puntos.

`m` es el grado del polinomio de ajuste deseado

`c` es el vector con los coeficientes del polinomio de ajuste

Si $m = 1$, el polinomio resultante es una recta, conocida con el nombre de **recta de regresión**, si $m = 2$ es una parábola, y así sucesivamente. Naturalmente, cuando $m = N - 1$ el polinomio calculado es el polinomio de interpolación global de grado $N - 1$ que pasa por todos los puntos.

Ejemplo 6.5
Calcula y dibuja (en el intervalo $[0.5, 10]$) los polinomios de ajuste de grado 1, 2, 3 y 6 para los siguientes datos:

```
x = [ 0.9, 1.5, 3,   4,   6,   8,   9.5]
y = [ 0.9, 1.5, 2.5, 5.1, 4.5, 4.9, 6.3]
```

¿Cuál es la ecuación de la recta de regresión?

```
% ------------ puntos del soporte
x = [0.9, 1.5, 3,   4,   6,   8,   9.5];
y = [0.9, 1.5, 2.5, 5.1, 4.5, 4.9, 6.3];

% ------------ polinomio de grado 1
coef1 = polyfit(x,y,1);
p1 = @(x) polyval(coef1, x);

% ------------ polinomio de grado 2
coef2 = polyfit(x,y,2);
p2 = @(x) polyval(coef2, x);

% ------------ polinomio de grado 3
coef3 = polyfit(x,y,3);
p3 = @(x) polyval(coef3, x);

% ------------ polinomio de grado 6 (pol. interpolacion global)
coef6 = polyfit(x,y,6);
p6 = @(x) polyval(coef6, x);

% ------------ graficas
z = linspace(0.5, 10);
plot(z, p1(z), z, p2(z), z, p3(z), z, p6(z))
hold on
plot(x, y, 'm.', 'MarkerSize', 15)
legend('Recta de regresion', 'Parábola de regresion', ...
       'Cubica de regresion','Pol. interpolacion global',...
       'Location', 'SouthEast')
hold off
shg
```

Se obtendrá `coef1 = [0.5688, 0.9982]` luego la recta de regresión es $y = 0.5688x + 0.9982$.

Ejercicio 6.6 En el fichero `Finanzas.txt` está recogido el precio de una determinada acción de la Bolsa española a lo largo de 88 días, tomado al cierre de cada día.

Queremos ajustar estos datos por una recta que nos permita predecir el precio de la acción para un corto intervalo de tiempo más allá de la última cotización.

Lee los datos del fichero y represéntalos mediante una linea poligonal, para observar las oscilaciones de las cotizaciones. Calcula y representa gráficamente la recta de regresión para estos datos.

Ejemplo 6.6
En el fichero `concentracion.txt` se dispone de unos datos, organizados en dos columnas, obtenidos por medición durante una reacción química. La primera columna representa el tiempo, en horas, y la segunda la concentración de cierta sustancia. Se sabe que la concentración, $c(t)$ se comporta como una exponencial: $c(t) = b\,e^{at}$, por lo que se desea ajustar los datos mediante una curva de ese tipo.

Una forma de encontrar esta función es hacer un cambio de variables que transforme la relación exponencial entre c y t en una relación lineal entre las nuevas variables. Denotamos, pues, $y = \ln(c)$, y se tiene

$$c = be^{at} \iff y = \ln(c) = \ln(be^{at}) = \ln(b) + \ln(e^{at}) = \ln(b) + at$$

Así, si ajustamos los datos $(t, y) = (t, \ln(c))$ mediante una recta $y = m + at$, tendremos la curva de ajuste exponencial $c = e^m\, e^{at}$ para los datos originales.

```
% -------- lectura de los datos
M = load('concentracion.txt');
t = M(:,1);
c = M(:,2);

% -------- calculo de la recta de regresion para los
%          datos t, log(c)
coef = polyfit(t, log(c), 1);

% ------- construccion de la exponencial
a = coef(1);
b = exp( coef(2) );
fc = @(t) b*exp(a*t);

% ------- graficas
plot(t,c,'r.')
hold on
plot(t, fc(t), 'LineWidth', 1.1)
hold off
shg
```

Ejercicio 6.7 Escribir una M-función

```
function FunInterp(f, a, b, n)
```

que reciba como argumentos de entrada
— `f` : un *handle* de una función
— `a, b` : un intervalo
— `n` : un número entero positivo $n > 2$

La M-función debe construir una partición regular del intervalo $[a, b]$ con n puntos, calcular los valores de la función f en dichos puntos, y calcular y dibujar juntas en el intervalo $[a, b]$ las gráficas

de
— los puntos del soporte, con marcadores
— el polinomio de interpolación global
— el interpolante lineal a trozos
— el *spline* cúbico
— la recta de regresión (ajuste mediante un polinomio de grado 1)
— la parábola de regresión (ajuste mediante un polinomio de grado 2)
debidamente identificadas.