Lección G

Interpolación y aproximación

El objetivo principal de la interpolación y aproximación de funciones es poder estimar valores funcionales a partir de cierto número de datos iniciales. Este objetivo es fundamental ya que de él se deducen los métodos de derivación e integración numéricas que tendremos ocasión de ver más adelante.

G.1. Aritmética de los polinomios

En este apartado vemos los comando más usuales de que dispone Matlab para trabajar con polinomios.

En Matlab un polinomio se representa mediante un vector fila que contiene los coeficientes de las potencias de la variable en orden decreciente. Por ejemplo, los polinomios $p = 2x^4 + 33x + 2$, $q = 16x^3 + 23x^2 - 24x + 1.2$ serán introducidos en el ordenador con los mandatos

```
p=[2 0 0 33 2], q=[16 23 -24 1.2]
```

Práctica a Realizamos las siguientes operaciones con los polinomios p, q anteriores:

`conv(p,q)`	Realizamos el producto de los polinomios p y q
`[c,r]=deconv(p,q)`	División con resto del polinomio p entre q
`s=roots(p)`	Cálculo de las raíces del polinomio p
`poly(s)`	Calcula el polinomio cuyas raíces son las componentes de s
`y=polyval(q,[0 1 2])`	Evalúa el polinomio q en los puntos $x = 0, 1, 2$.

G.2. Interpolación y aproximación polinomial

Práctica b Consideremos la siguiente tabla de datos:

$\mathbf{x_i}$	1.0	2.0	3.5	5.0
$\mathbf{f(x_i)}$	1.2375	8.7417	44.5976	127.5692

(G.1)

y nos planteamos el cálculo de un polinomio $f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ que verifique $f(x_i) = f_i$ lo cual equivale a resolver el siguiente sistema

$$\begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ x_3^3 & x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

La matriz de este sistema recibe el nombre de matriz de Vandermonde del vector $(x_0, x_1, x_2, x_3)^t$, como se puede observar su fila i-ésima está formada por las potencias sucesivas de x_{i+1}. Llevamos a Matlab el anterior sistema con el listado

```
A=[1 1 1 1;8 4 2 1;42.875 12.25 3.5 1;125 25 5 1] b=[1.2375
8.7417   44.5976  127.5692].' p=(A\b).'
```

Obtenemos como respuesta `p=[0.9776 0.2057 0.0441 0.0101]` con lo que el polinomio interpolador de la función f es $p(x) = 0.9776x^3 + 0.2057x^2 + 0.0441x - 0.0101$.

Existen dos formas alternativas de realizar los cálculos anteriores mediante Matlab:

1.º Método:

```
A=vander([1 2 3.5 5])          %Matriz de Vandermonde
b=[1.2375    8.7417   44.5976  127.5692].' p=(A\b).'
```

2.º Método:

```
x=[1 2 3.5 5], y=[1.2375    8.7417   44.5976  127.5692]
p=polyfit(x,y,3)
```

La función `polyfit` calcula directamente el polinomio interpolador que pasa por los puntos x_i, f_i. Además si en el listado anterior hubiéramos puesto en su segunda línea la orden

`polyfit(x,y,2)`

entonces Matlab nos hubiera calculado, por el método de los mínimos cuadrados, el polinomio de grado dos que más se aproxima a los datos (x_i, f_i). Veamos con mayor detalle esto que estamos diciendo:

Un polinomio de grado dos $C(x) = c_2x^2 + c_1x + c_0$ se desvía de cada uno de nuestros datos (x_i, f_i), $i = 1, \ldots, 4$, en la cantidad

$$r_i = f_i - C(x_i), \quad i = 1, \ldots, 4$$

La suma de los cuadrados de dichas desviaciones tiene la expresión

$$R = \sum_{i=1}^{4} r_i^2$$

de donde, el mínimo, en función de los coeficientes c_j se logrará cuando las derivadas parciales de R respecto a c_j son cero, es decir, cuando ocurre:

$$\begin{aligned} c_2(x_0^4+\cdots+x_3^4)+c_1(x_0^3+\cdots+x_3^3)+c_0(x_0^2+\cdots+x_3^2) &= y_0x_0^2+\cdots+y_3x_3^2 \\ c_2(x_0^3+\cdots+x_3^3)+c_1(x_0^2+\cdots+x_3^2)+c_0(x_0+\cdots+x_3) &= y_0x_0+\cdots+y_3x_3 \\ c_2(x_0^2+\cdots+x_3^2)+c_1(x_0+\cdots+x_3)+c_0(1+1+1+1) &= y_0+y_1+y_2+y_3 \end{aligned} \tag{G.2}$$

lo que equivale a resolver el sistema

$$\begin{bmatrix} \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i^1 \\ \sum x_i^2 & \sum x_i^1 & \sum x_i^0 \end{bmatrix} \begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \sum x_i^2 y_i \\ \sum x_i^1 y_i \\ \sum x_i^0 y_i \end{bmatrix}; \qquad H = \begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \tag{G.3}$$

sistema del que observamos que tiene la forma $H^tHc = H^tb$ para H la matriz anterior. Ahora, se verifica que resolver el sistema $H^tHc = H^tb$ es equivalente a resolver el sistema $Hc = b$ por el método de los mínimos cuadrados (ver la práctica ED) lo que podemos lograr con el listado :

```
x=[1 2 3.5 5], y=[1.2375 8.7417 44.5976 127.5692]
 A=vander(x); H=A(:,[2 3 4])
 c=H\y
```

o simplemente ejecutando

```
c=polyfit(x,y,2)
```

Práctica c Es importante tener en cuenta que las matrices de Vandermonde están en general mal condicionadas con lo cual el método explicado en la práctica anterior para la obtención del polinomio interpolador es muy sensible a cambios en las condiciones iniciales. Como ejemplo vamos a repetir dicho cálculo pero suponiendo que $b' = b - v_2$ para $v_2 = (0.0516, -0.0757, 0.0390, -0.0090)$ el vector de norma 0.1 obtenido en la práctica EJ. Ejecutamos el listado

```
A=vander([1 2 3.5 5])
 bprima=[1.1859    8.8174   44.5586   127.5781].'
 p=(A\bprima).'
```

y el resultado obtenido es el polinomio interpolador

$$P_{b'}(x) = 1.0070x^3 - 0.0670x^2 + 0.7835x - 0.5377$$

del que podemos observar que es bastante diferente del polinomio $p(x) = 0.9776x^3 + 0.2057x^2 + 0.0441x + 0.0101$ obtenido en la práctica anterior para b a pesar de que el error relativo cometido al sustituir b por b' es de $7.3840e-4$. La justificación de este hecho se visualiza si ejecutamos `cond(vander([1 2 3.5 5]))` ya que obtenemos 1300.9 como valor de la condición de la matriz del sistema.

Práctica d En esta práctica vamos a ajustar los datos

```
x=[0.10 0.40 0.50 0.70 0.71 0.90]; y=[0.61 0.92 0.99 1.47 1.52
2.03];
```

lineal y cuadráticamente y a compararlo, mediante una gráfica, con el polinomio interpolador de Lagrange. Para ello, ejecutamos el listado

```
Lag=polyfit(x,y,length(x)-1); Ajus1=polyfit(x,y,1);
Ajus2=polyfit(x,y,2);
 xx=0:0.005:1;          %Discretización de [0,1]
yLag=polyval(Lag,xx); yAjus1=polyval(Ajus1,xx);
yAjus2=polyval(Ajus2,xx);
plot(xx,yLag,'r',xx,yAjus1,':g',xx,yAjus2,'-.m',x,y,'+')
legend('Lagrange','Ajuste lin.','Ajuste cuadr.',0)
```

Mediante la gráfica obtenida podremos observar que no siempre conviene realizar el ajuste con un orden alto, ya que a veces se comporta mejor el de orden bajo.

Práctica e ([7],p. 289) Se nos da el conjunto (x_i, y_i) de datos reflejado en el siguiente listado del cual se sabe o se supone que pueden quedar ajustados mediante una función potencial $f(x) = \beta x^{\alpha}$. En esta práctica se nos pide realizar tal ajuste y representarlo para ver si es correcto. La simple observación de que la toma de logaritmos convierte la función potencial anterior en la función lineal $\log f(x) = c_1 \log x + c_2$, $c_1 = \alpha$, $c_2 = \log(\beta)$, nos dice que el ajuste potencial se convierte en un ajuste lineal a condición de tomar logaritmos. Todo esto lo realizamos con el guión siguiente:

```
x=[0.15 0.4 0.6 1.01 1.5 2.2 2.4 2.7 2.9 3.5 3.8 4.4 4.6 5.1 6.6 7.6];
y=[4.4964 5.1284 5.6931 6.2884 7.0989 7.5507 7.5106 ...
8.0756 7.8708 8.2403 8.5303 8.7394 8.9981 9.1450 9.5070 9.9115];
c=polyfit(log(x),log(y),1);
beta=exp(c(2)), alfa=c(1),
xx=0:0.05:8;
yy=exp(c(2))*(xx.^c(1));
plot(xx,yy,x,y,'+')
```

G.3. Extrapolación

Práctica f Consideremos los datos correspondientes a puntos de la gráfica de la función $y = x^3$:

```
x=[0 1 2  3  4] y=[0 1 8 27 64]
```

y supongamos que a partir de dichos datos deseamos realizar una interpolación para calcular el valor de $\sqrt[3]{20}$. Dicha interpolación deberá ser inversa ya que nosotros conocemos $y_0 = 20$ y nuestro deseo es conocer el x_0 que lo produce. La *interpolación inversa* o *extrapolación* consiste entonces en expresar la x en función de la y. En nuestro caso y usando la expresión de Lagrange del polinomio interpolador tendremos:

$$\begin{aligned} x =& y\left[\frac{(y-8)(y-27)(y-64)}{1\cdot(-7)(-26)(-63)}\cdot 1+\frac{(y-1)(y-27)(y-64)}{8\cdot 7\cdot(-19)(-56)}\cdot 2\right. \\ &\left.+\frac{(y-1)(y-8)(y-64)}{27\cdot 26\cdot 19\cdot(-37)}\cdot 3+\frac{(y-1)(y-8)(y-27)}{64\cdot 63\cdot 56\cdot 37}\cdot 4\right] \end{aligned} \tag{G.4}$$

y realizando dichas operaciones con $y_0 = 20$ obtenemos $x_0 = -1.3139$, en lugar del valor correcto 2.7144. Las operaciones anteriores se pueden realizar, en Matlab, simplemente con los mandatos

```
c=polyfit(y,x,length(x)-1), x0=polyval(c,20)
```

Si hubiéramos realizado el ajuste lineal de los datos anteriores hubiéramos obtenido

```
c_lin=polyfit(y,x,1),x0_lin=polyval(c_lin,20)
```

x0_lin=2.63

lo que supone una desviación de sólo el 3 %, con lo que queda patente una vez más que no siempre conviene un ajuste de orden alto.

G.4. Diferencias divididas. Polinomio interpolador de Newton

Práctica g Consideremos la tabla (G.1) como nuestra tabla de datos. Nos proponemos calcular el polinomio interpolador asociado a dichos datos en su forma de Newton

$$\begin{aligned} P(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) \\ P(x) &= a_0 + (x-x_0)(a_1 + (x-x_1)(a_2 + (x-x_2)a_3)) \end{aligned} \tag{G.5}$$

Si hacemos $P(x_i) = f_i$, llegamos a un sistema de ecuaciones triangular inferior de la forma

$$\begin{aligned} a_0 &= f_0 \\ a_0 + a_1(x_1-x_0) &= f_1 \\ a_0 + a_1(x_2-x_0) + a_2(x_2-x_0)(x_2-x_1) &= f_2 \\ a_0 + a_1(x_3-x_0) + a_2(x_3-x_0)(x_3-x_1) + a_3(x_3-x_0)(x_3-x_1)(x_3-x_2) &= f_3 \end{aligned} \tag{G.6}$$

Ahora, teniendo en cuenta la definición:

Se define la *diferencia dividida* de orden 1 entre dos puntos s y t como

$$f[x_s, x_t] = \frac{f_t - f_s}{x_t - x_s} \tag{G.7}$$

Las diferencias divididas de ordenes superiores se construyen tomando como base las diferencias divididas de orden inferior mediante la regla recursiva

$$f[x_0, \ldots, x_i] = \frac{f[x_1, \ldots, x_i] - f[x_0, \ldots, x_{i-1}]}{x_i - x_0} \tag{G.8}$$

Es fácil de resolver el sistema anterior utilizando sustitución progresiva, obteniéndose:

$$\begin{aligned} a_0 &= f_0 \\ a_1 &= f[x_0, x_1] \\ a_2 &= f[x_0, x_1, x_2] \\ a_3 &= f[x_0, x_1, x_2, x_3] \end{aligned} \tag{G.9}$$

Lo que nos dice que la expresión del polinomio de interpolación en su forma de Newton se escribe:

$$\begin{aligned} P(x) &= f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1)(x - x_2) \end{aligned} \tag{G.10}$$

y también

$$\begin{aligned} P(x) &= f(x_0) + \frac{(x - x_0)}{h}\Delta f(x_0) + \frac{(x - x_0)(x - x_1)}{2!h^2}\Delta^2 f(x_0) \\ &\quad + \frac{(x - x_0)(x - x_1)(x - x_2)}{3!h^3}\Delta^3 f(x_0) \end{aligned} \tag{G.11}$$

cuando la distancia entre las abscisas de nuestros datos es constante e igual a h y donde Δ es el *operador diferencia* definido por la igualdad $\Delta f(x_0) = f(x_1) - f(x_0)$, con lo que $\Delta^2 f(x_0) = \Delta(f(x_1) - f(x_0)) = f(x_2) - f(x_1) - f(x_1) + f(x_0) = f(x_2) - 2f(x_1) + f(x_0)$.

En consecuencia, para calcular los coeficientes del polinomio de interpolación en su forma de Newton basta obtener las sucesivas diferencias divididas lo que hacemos en la siguiente tabla

x_i	f_i	$f[x_i, x_i + 1]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_0, x_1, x_2, x_3]$
1.0	**1.2375**			
		7.5042		
2.0	8.7417		**6.5599**	
		23.9039		**0.9776**
3.5	44.5976		10.4702	
		55.3144		
5.0	127.5692			

(G.12)

de donde $P(x) = 1.2375 + (x - 1)(7.5042 + (x - 2)(6.5599 + (x - 3.5)0.9776))$, es decir, $P(x) = 0.9776x^3 + 0.2057x^2 + 0.0441x + 0.0101$

La ventaja de calcular el polinomio interpolador mediante las diferencias divididas es que permite utilizar la información obtenida al realizar una interpolación de n puntos a la hora de obtener la interpolación de $n + 1$ puntos. En particular, si nuestra tabla de datos hubiera sido

$\mathbf{x_i}$	1.0	2.0	3.5
$\mathbf{f(x_i)}$	1.2375	8.7417	44.5976

(G.13)

entonces nuestra tabla de diferencias hubiera sido

x_i	f_i	$f[x_i, x_i+1]$	$f[x_0, x_1, x_2]$
1.0	**1.2375**		
		7.5042	
2.0	8.7417		**6.5599**
		23.9039	
3.5	44.5976		

(G.14)

de donde observamos que las tres primeras diferencias divididas son las mismas que en (G.12) y por tanto para calcular el polinomio interpolado $P(x)$ a partir del polinomio interpolador de los tres datos anteriores, $p(x) = 1.2375 + (x-1)(7.5042 + (x-2)6.5599)$ sólo tendríamos que calcular la 4.ª diferencia dividida, lo que podemos hacer a partir de la tabla anterior hasta obtener la misma que en (G.12).

En el fichero `g_difdiv` ([6], pág. 244) encontramos la implementación del algoritmo de las diferencias dividas. En consecuencia, se propone listar dicho fichero y ejecutar

```
x=[1 2 3.5 5],
y=[1.2375 8.7417 44.5976 127.5692],
g_difdiv(x,y)
```

con el fin de obtener de un sólo golpe lo que hemos descrito en esta práctica. También se pide comprobar que todos los datos contenidos en la tabla (G.12) son correctos.

Práctica h Vamos a comprobar cuál de los métodos descritos para calcular el polinomio interpolador es más exacto y más eficiente, si los descritos en la práctica GB o el descrito en la práctica anterior. Para ello ejecutamos el listado:

```
x=[1 2 3.5 5];
y=[1.2375 8.7417 44.5976 127.5692];
flops(0);
P1=polyfit(x,y,length(x)-1), operaciones1=flops;

yaprox1=polyval(P1,x); normaresidual1=norm(y-yaprox1);
A=vander(x); flops(0); P2=(A\y.').', operaciones2=flops;
yaprox2=polyval(P2,x); normaresidual2=norm(y-yaprox2);
flops(0); P3=g_difdiv(x,y), operaciones3=flops;
yaprox3=polyval(P3,x); normaresidual3=norm(y-yaprox3);
resultados=[operaciones1,normaresidual1,operaciones2,...
normaresidual2,operaciones3,normaresidual3];
fprintf('oper 1\t ...resid 1\t oper 2\t ...resid 2\t oper 3\t ...resid3\n')
fprintf('%i\t %e\t %i\t %e\t %i\t %e\n',resultados.')
```

Con ello comprobamos que la exactitud en todos ellos es semejante y que el número de operaciones es menor en el caso de las diferencias divididas.

G.5. Interpolación polinomial fragmentaria

En este tipo de interpolación no se intenta aproximar todos los datos o la función mediante un único polinomio interpolante sino que se intenta aproximar mediante fragmentos o trozos polinomiales conectados con mayor o menor regularidad, con ello intentamos conseguir comportamientos más suaves y menos oscilantes de las funciones interpolantes.

Práctica i Matlab cuenta con un comando que le permite trabajar con funciones polinomiales definidas a trozos. Lo vemos con un ejemplo:

Consideramos la *función polinomial fragmentaria* siguiente

$$f(x) = \begin{cases} x+1 & si \quad x \in [0,1) \\ 2(x-1)^2+1 & si \quad x \in [1,2) \\ -2(x-2)^3+3 & si \quad x \in [2,3) \end{cases} \tag{G.15}$$

de la que observamos:

- f es una función que está definida en $[0,3]$ mediante 3 fragmentos polinomiales, un fragmento por cada uno de los intervalos $[0,1)$, $[1,2)$ y $[2,3)$ lo que puede resumirse diciendo que f es una *función polinomial fragmentaria* definida sobre los *nodos* $x_0 = 0 < x_1 = 1 < x_2 = 2 < x_3 = 3$.
- La expresión de f en cada tramo $[x_i, x_{i+1})$ está dado como una serie de potencias respecto al nodo x_i.
- El 'grado' de f es tres, en el sentido de que el mayor de los grados de las funciones polinomiales que intervienen en la definición de f es de grado tres.

Pues bien, en dichas condiciones, la función f se puede introducir en el ordenador mediante el comando `mkpp` de la siguiente manera:

```
x=[0 1 2 3];          %introducimos los nodos
Coef=[0 0 1 1         %introducimos el primer tramo polinomial, x+1=[0 0 1 1]
      0 2 0 1         %segundo tramo polinomial
     -2 0 0 3];       %tercer y último tramo polinomial
f=mkpp(x,Coef);       %creación de f
[nodos,coeficientes,numero_trozos,grado_mas_1]=unmkpp(f)  %Operación

                      % inversa a la anterior
```

Ahora la representación gráfica de f la realizamos añadiendo los siguientes comandos al listado anterior

```
xp=-.5:0.01:3.2;      %discretización del intervalo [-0.5,3.2]
yp=ppval(f,xp);       %valores de f en xp
x1=[1 2 3]; y1=[2 3 1];          %(x1,y1)puntos finales
x2=[0  1  2]; y2=[1  1  3];      %(x2,y2)puntos iniciales
```

```
plot(xp,yp,'.',x,0*x,'*',x2,y2,'o',x1,y1,'x',xp,0*xp)
axis([-1 3.5 -2 4.5])
legend('f','nodos','inicio','final',0)
```

De la representación gráfica de f observamos que no sólo está definida en $[0,3]$ sino en todo $\mathbb{R}$. Para valores menores que 0 coincide con el polinomio $x+1$ y para valores mayores que 3 con el polinomio $-2(x-2)^3+3$

Práctica j El método más sencillo de interpolación polinomial fragmentaria consistiría en unir nuestros datos mediante un polinomio de primer orden, en cuyo caso no podemos esperar que la función global obtenida sea derivable. Matlab cuenta con el comando `interp1` que en su opción `'linear'` realiza este tipo de interpolación que se denomina *interpolación lineal*. Veamos con un ejemplo concreto como funciona dicho comando

```
x=[0 1.6 3.2 4.8 6.2832],                  %datos
y=[0  0.9996  -0.0584  -0.9962  0.0000],   %datos
x1=[0:0.05:2*pi];                 %discretización de [0,2pi]
y1=sin(x1);                       %la función que interpolamos es el seno
ylin=interp1(x,y,x1,'linear');
plot(x1,y1,'r',x1,ylin,':g',x,y,'*')
legend('y=sen(x)','interpl
lin.','datos',0)
```

Otras opciones del comando `interp1` son: `'spline'` y `'cubic'` que serán explicados seguidamente.

Una de las interpolaciones fragmentarias más utilizada es la del *trazador cúbico*, también denominada *c-spline*. En ella lo que hacemos es interpolar en cada uno de los intervalos $[x_i, x_{i+1})$ mediante un polinomio de grado tres de modo que la función global tenga derivada segunda continua. En realidad, para esto no existe un método único ya que dados $n+1$ puntos $x_0, x_1, \ldots, x_n$ el conjunto de cúbicas que podemos definir sobre ellos,

$$C_3 = \{ax^3 + bx^2 + cx + d, \text{ tales que } a,b,c,d \in \mathbb{R}\} \tag{G.16}$$

es un espacio vectorial de dimensión $4n$ y ahora el número de condiciones que representa que sea de clase 2 y que pase por los datos es igual a $4n-2$ ya que: que pase por los datos representa $2n$ condiciones, dos por fragmento; que además sea de clase 1 y 2 representa $2\cdot(n-1)$ condiciones adicionales, tantas como puntos interiores donde exigir que las derivadas primera y segunda coincidan.

En consecuencia, para obtener el trazador cúbico de unos datos tenemos 2 grados de libertad que podemos fijar de las siguientes formas, obteniendo en cada caso un tipo distinto de trazador

completo: En este tipo exigimos que la derivada primera en el primer nodo y el último sean iguales a cero.

natural: La derivada segunda en el primer y último nodos son iguales a cero. Eso significa terminar la gráfica de la función con curvatura cero.

extrapolado: En este tipo fijamos la derivada primera en el primer nodo como la obtenida al extrapolarla con la derivada de los dos siguientes nodos y la derivada primera en el último nodo como la obtenida al extrapolarla con sus dos anteriores nodos. Este es el método que usa Matlab cuando ejecutamos el comando `interp1` en su opción `'spline'`. Los detalles de la extrapolación realizada se pueden encontrar en [7], pág. 303.

cíclico: Se exige que el valor de la función y de sus derivadas en su último nodo coincidan con las del primer nodo.

sujetos: En este caso se fijan los valores de la derivada en el nodo inicial y en el final. Los cálculos se puede hacer de forma automática utilizando la función `g_sujeto` definida en el libro [6], página 268.

Pasamos ahora a poner ejemplos de los trazadores cúbicos más usados:

Práctica k ([6], p. 268) Vamos a calcular el trazador cúbico con condiciones sobre la derivada que coincide con la función $f(x) = xe^x$ en los puntos $0, 1/2$ y 1. La tabla de datos, en consecuencia, es

x	$f(x)$	$f'(x)$
0	0	1
0.5	$\sqrt{e}/2$	
1	e	$2e$

(G.17)

Y el listado que nos calcula y representa dicho trazador cúbico es

```
x=[0 0.5 1]; y=[0 0.5*exp(0.5) exp(1)];
[Coef,P]=g_sujeto(x,y,1,2*exp(1));  %Calculamos  los coeficientes del
                            %polinomio fragmentario P sujeto a pasar
                            %por los datos (x,y) y con derivada inicial
                            %  1 y final 2*exp(1)

xp=0:(1/500):1;             %discretización del intervalo [0,1]
yp=ppval(P,xp);             %(xp,yp) puntos del trazador
fp=xp.*exp(xp);             %(xp,fp) puntos de la función
plot(xp,fp,xp,yp,':') legend('función f','trazador',0)
error=max(abs(yp-fp)); trozos=[0 0.5;0.5 1];
resultados=[trozos,Coef];
fprintf('[x_i.....\t .....x_i+1)\t coef. x^3\t coef. x^2\t ...
coef. x\t coef. x^0\n')
fprintf('%f\t %f\t %f\t %f\t %f\t %f\n',resultados.');
disp('máximo error cometido con el trazador'), error
```

Práctica l En el siguiente guión representamos el trazador cúbico extrapolado que pasa por los datos que también damos en el guión

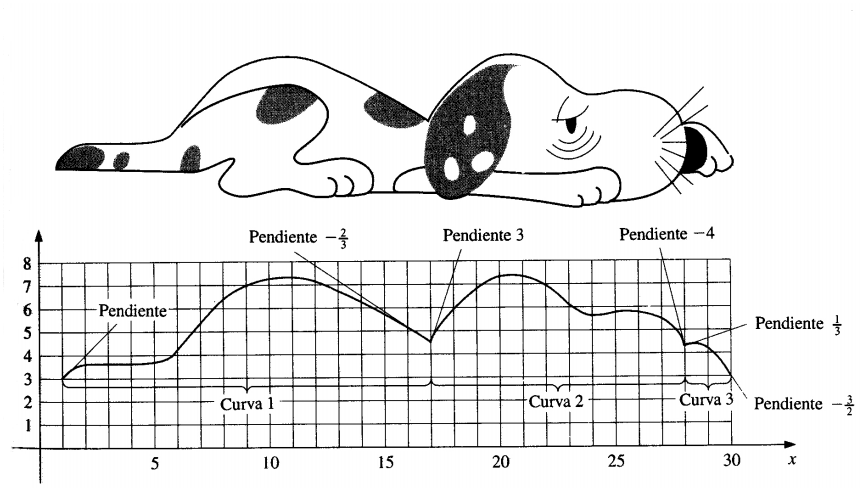
```
x=[0  1  3  5  5.8  6  5  4  3.3  2  1  1.5  3  4.15  5  6];
y=[3.3  3.7  5  8.5  11  13  13.66  13.05  12  8  4.5...
  1.5  0  0.5  1.1  2.8];

s=1:length(x);              %s es el parámetro que usamos
                            % para ajustar x e y por separado
sp=1:(length(x)/500):length(x);   %discretización de s
xp=spline(s,x,sp);
yp=spline(s,y,sp);

plot(xp,yp,x,y,'o')
%plot(xp,yp)
axis([-5 10 -5 18])
axis equal
xlabel('eje x'), ylabel('eje y')
```

¿Podrías decir por qué necesitamos parametrizar la x y la y independientemente?.

Práctica m ([1], pág. 157) Mediante los trazadores cúbicos sujetos vamos a aproximar la parte superior del perro que se muestra en la siguiente figura . En ella también aparece la curva a aproximar en una cuadrícula a partir de la cual se construyeron los datos del listado que sigue a la citada figura




```
x1=[1 2   5   6   7   8   10  13  17 ];
y1=[3 3.7 3.9 4.2 5.7 6.6 7.1 6.7 4.5];   %(x1,y1) puntos de la Curva 1
rr1=length(x1);
[CoefP1,P1]=g_sujeto(x1,y1,1,-(2/3));     %P1 polinomio fragmentario de C.1

x2=[17    20.5 22   23  24  25  27  27.7];
y2=[4.5   7.45 6.9  6.1 5.6 5.8 5.4 4.1];       %(x2,y2) puntos de C.2
rr2=length(x2);
[CoefP2,P2]=g_sujeto(x2,y2,2,-4);         %P2 polinomio fragmentario de C.2
```

```
x3=[27.7  28  29  30];
y3=[4.1   4.3 4.1 3];                      %(x3,y3) puntos de C.3
[CoefP3,P3]=g_sujeto(x3,y3,(1/3),-(3/2)); %P3 polinomio fragmentario de C.3

x=[x1(1:(rr1-1)), x2(1:(rr2-1)), x3];     %nodos
y=[y1(1:(rr1-1)), y2(1:(rr2-1)), y3];
Coef=[CoefP1; CoefP2; CoefP3];
P=mkpp(x,Coef);                            %Polinomio fragmentario total

xp=1:(1/20):30;                            %discretización de [1,30]
yp=ppval(P,xp);                            %(xp,yp) puntos del trazador
clf
plot(xp,yp)
axis equal
```

obsérvese como en la definición del polinomio fragmentario de la segunda curva ponemos el mandato

```
[CoefP2,P2]=g_sujeto(x2,y2,2,-4)
```

en lugar de `[CoefP2,P2]=g_sujeto(x2,y2,3,-4)` y es que creemos que se ajusta mejor al dibujo una pendiente inicial de magnitud 2 que no de 3.

Ejecutamos ahora las ordenes

```
Q=polyfit(x,y,length(x)-1);
yp_lag=polyval(Q,xp);
figure(2),
plot(xp,yp_lag)
```

con el fin de observar las diferencias de realizar una interpolación fragmentaria y una interpolación total.

G.6. Interpolación bidimensional

Práctica n Realizamos un ejemplo de interpolación bidimensional. Se considera la siguiente tabla de valores medidos para la variable z en los valores de las variables x e y que aparecen en la siguiente tabla:

x\y	-3	-2	-1	0	1	2	3
-3	37.0000	23.0000	11.0000	1.0001	-7.0000	-13.0000	-17.0000
-2	22.0000	13.0003	6.0067	1.0183	-1.9933	-2.9997	-2.0000
-1	13.0000	7.0067	3.1353	1.3679	1.1353	3.0067	7.0000
0	10.0001	5.0183	2.3679	2.0000	2.3679	5.0183	10.0001
1	13.0000	7.0067	3.1353	1.3679	1.1353	3.0067	7.0000
2	22.0000	13.0003	6.0067	1.0183	-1.9933	-2.9997	-2.0000
3	37.0000	23.0000	11.0000	1.0001	-7.0000	-13.0000	-17.0000

Queremos calcular y representar, utilizando interpolación lineal y cúbica bidimensional, el valor estimado de la variable z sobre un mallado $[-3,3]\times[-3,3]$ de incremento 0.4. Para ello ejecutamos el listado

```
x=[-3:1:3]; y=x;     %datos
[X,Y]=meshgrid(x,y);
%Datos para z
%Z=F(X,Y) para F desconocida. La calculamos por interpolación
% Lineal y Cúbica
Z=[
 37.0000   23.0000   11.0000    1.0001   -7.0000  -13.0000  -17.0000
 22.0000   13.0003    6.0067    1.0183   -1.9933   -2.9997   -2.0000
 13.0000    7.0067    3.1353    1.3679    1.1353    3.0067    7.0000
 10.0001    5.0183    2.3679    2.0000    2.3679    5.0183   10.0001
 13.0000    7.0067    3.1353    1.3679    1.1353    3.0067    7.0000
 22.0000   13.0003    6.0067    1.0183   -1.9933   -2.9997   -2.0000
 37.0000   23.0000   11.0000    1.0001   -7.0000  -13.0000  -17.0000
];
%Realizamos el mallado donde queremos calcular F
  [XI,YI]=meshgrid(-3:0.4:3,-3:0.4:3);
subplot(1,2,1)
%Interpolación Lineal
ZI=interp2(X,Y,Z,XI,YI); mesh(XI,YI,ZI),
xlabel('x'), ylabel('y'), zlabel('z'),
title('Interpolación Lineal')
%Interpolación Cúbica
subplot(1,2,2)
ZI=interp2(X,Y,Z,XI,YI,'cubic'); mesh(XI,YI,ZI),
xlabel('X'), ylabel('Y'), zlabel('Z'),
title('Interpolación Cúbica')
```

G.7. Cálculo de ceros de funciones

Como aplicación de las fórmulas de interpolación damos aquí los algoritmos más empleados en el cálculo de ceros de funciones.

Consideramos el problema de calcular un cero $r \in \mathbb{R}$ de una función continua f de la que se sabe que lo posee en el intervalo $[a, b]$ porque signo $f(a) \neq$ signo $f(b)$.

Método de la regula falsi Como una primera aproximación al cero de f consideramos el cero c de la aproximación lineal de f que pasa por los datos $(a, f(a))$, $(b, f(b))$, fácilmente se obtiene

$$c = b - f(b)\frac{b-a}{f(b)-f(a)} \tag{G.18}$$

si ahora signo $f(a) =$ signo $f(c)$ entonces f tiene un cero en $[c, b]$ y en caso contrario f lo tiene en $[a, c]$. En cualquier caso para a_1, b_1 los números reales definidos por las igualdades

$$a_1 = \begin{cases} c & \text{si signo } f(a) = \text{signo } f(c) \\ a & \text{en caso contrario} \end{cases} \qquad b_1 = \begin{cases} b & \text{si signo } f(a) = \text{signo } f(c) \\ c & \text{en caso contrario} \end{cases}$$

se verifica $r \in [a_1, b_1] \supset [a_0 = a, b_0 = b]$ y que, por tanto, r se puede obtener como límite de las sucesiones monótonas $\{a_n\}$ y $\{b_n\}$ construidas tal y como hemos señalado en la construcción de a_1, b_1 a partir de $a_0 = a$, $b_0 = b$.

Método de Newton En el método de Newton se obtiene la aproximación al cero de f calculando el cero de la recta tangente en b a la gráfica de la función f, eso implica tomar como primera aproximación de r el valor

$$c = b - \frac{f(b)}{f'(b)} \tag{G.19}$$

y como sucesivas aproximaciones las obtenidas por la recurrencia

$$b_0 = b,\ b_1 = b_0 - \frac{f(b_0)}{f'(b_0)},\ \ldots, b_{n+1} = b_n - \frac{f(b_n)}{f'(b_n)},\ \ldots \tag{G.20}$$

Método de Brent En este método necesitamos una primera aproximación $c_0 \in [a, b]$ a la raíz r que podemos tomar como el valor medio o el valor obtenido por la regula falsi y a partir de él construimos una nueva aproximación c_1 de la raíz utilizando interpolación cuadrática que claramente habrá de ser inversa ya que de $r \cong c_1$ sabemos que su imagen por f es cero.

De aquí obtenemos

$$\begin{aligned} c_1 &= \frac{f_a f_b}{(f_{c_0}-f_a)(f_b-f_a)}c_0 + \frac{f_a f_{c_0}}{(f_b-f_a)(f_b-f_{c_0})}b + \frac{f_b f_{c_0}}{(f_a-f_b)(f_a-f_{c_0})}a \\ &= c_0 + \Delta = c_0 + \frac{G[H(F-H)(b-c_0)-(1-F)(b-a)]}{(F-1)(G-1)(H-1)} \end{aligned} \tag{G.21}$$

para $F = \frac{f_{c_0}}{f_b}$, $G = \frac{f_{c_0}}{f_a}$ y $H = \frac{f_a}{f_b}$.

A partir de la terna $a_0 = a > c_0 > b_0 = b$ hemos definido la terna $a_1 > c_1 > b_1$ por las igualdades

$$a_1 = \begin{cases} a & \text{si } c_1 \in [a, c] \\ c & \text{en caso contrario} \end{cases} \qquad b_1 = \begin{cases} c & \text{si } c_1 \in [a, c] \\ b & \text{en caso contrario} \end{cases} \tag{G.22}$$

y ahora r se obtiene como límite de cualesquiera de las sucesiones $\{a_n\}$, $\{b_n\}$ y $\{c_n\}$ definidas por recurrencia tal y como definimos $a_1 > c_1 > b_1$ a partir de $a_0 = a > c_0 > b_0 = b$.

Práctica o Una implementación de los algoritmos de Newton y de Brent los podemos encontrar en los ficheros `g_newton` ([7], pág. 277)y `g_brent`([6], pág 257), respectivamente. La implementación de la regula falsi la dejamos como ejercicio al lector (ver práctica GV) y ahora lo que vamos a hacer es comprobar la efectividad de las dos anteriores funciones y el de la función `fzero` en el cálculo de un cero de la función $f(x) = xe^x - 1$ en el intervalo $[0, 1]$, para ello ejecutamos:

```
flops(0), r_Newton=g_newton('g11',0,1e-7);
   oper_Newton=flops;
flops(0), r_Brent=g_brent('g11',0,1,1e-7);
   oper_Brent=flops;
flops(0), r_fzero=fzero('g11',0,1e-7);
   oper_fzero=flops;
resultados=[oper_Newton,r_Newton,oper_Brent,r_Brent,oper_fzero,r_fzero];
fprintf('o_New.\t r_Newton\t o_Br.\t r_Brent\t o_fz.\t r_fzero\n')
fprintf('%i\t %f \t %i\t %f\t %i\t %f\n', resultados.');
```

y donde previamente hemos introducido el listado

```
function f=g11(x) f=(x.*exp(x))-1;
```

para definir en Matlab la función f.

Del resultado podemos comprobar que los tres métodos nos dan el mismo cero, $r = 0.567143$, siendo distinto el número de operaciones necesario para obtenerlo, siendo estas 79 por el método de Newton, 141 por el método de Brent y 207 por el método de `fzero`. Por último mencionar el hecho de que la función `fzero` utiliza también el algoritmo de Brent en el cálculo de los ceros sólo que no precisa introducir el intervalo de búsqueda sino tan sólo un valor aproximado de éste.

G.8. Aplicación de la interpolación a la modelización

La modelización es la disciplina de la Matemática que se dedica a la elaboración de modelos matemáticos que expliquen determinados problemas. En esta sección vemos, con un ejemplo concreto, como se aplican las técnicas de interpolación y aproximación a la modelización de un problema.

ENUNCIADO DEL PROBLEMA: **Práctica puntuable** (0.2 puntos)

En la tabla (G.24) aparecen los periodos de revolución de los planetas del sistema solar y sus distancias medias al sol, suponiendo una relación de la forma $T = Cr^a$, $C, a \in \mathbb{R}$, para T el periodo de revolución alrededor del Sol de un planeta y r la distancia media al Sol. Encontrar los valores de C y a e intenta formular la ley que relacionan T y r. Esta ley es conocida como la *tercera ley de Kepler*. De hecho se verifica

$$T^2 = \frac{4\pi^2}{G \cdot M} r^3 \tag{G.23}$$

para M la masa del Sol y G la *constante gravitacional* de Newton, ver I.3.2.

Planeta	Periodo en días	Distancia (Km$\times 10^{-6}$)
Mercurio	88	57.6
Venus	225	108.2
Tierra	365	149.6
Marte	687	227.9
Jupiter	4329	778.3
Saturno	10753	1427.0
Urano	30660	2870.0
Neptuno	60150	4497.0
Plutón	90670	5907.0

(G.24)

INDICACIÓN: La solución es muy sencilla, basta ver la práctica GE para deducir los detalles de su resolución

G.9. Ejercicios

Práctica p Decidir si es posible encontrar una función $f(x)$ que pase por los datos

```
x=[0   0.4    0.8    1.2]
y=[1   1.491  2.225  3.32]
```

y que además verifique $f^{4)}(0.6) = 1.8222$ y $f(0.2) = 50$. Misma pregunta con las mismas condiciones salvo $f(0.2) = 100$.

Práctica q Consideramos la función $f(x) = \cos x$ en el intervalo [0.2]. Se pide:

1. Calcular el polinomio interpolador de f con seis puntos equiespaciados y representar la función error de la interpolación, es decir la función diferencia entre f y su polinomio interpolador.
2. Repetir lo anterior con 12 puntos equiespaciados.
3. Repetir el punto 1 para el trazador cúbico extrapolado.

Práctica r De una función f se tiene la siguiente información

x	$f(x)$	$f'(x)$
0	-0.232343	1.234934
1	0.730456	
2	2.224353	-0.93245

Con dichos datos se pide calcular la interpolación que se considere más conveniente.

Práctica s Considérese el siguiente conjunto de datos

x	$f(x)$
0.1	0.6029196177260
0.2	0.6598675103186
0.3	0.7283500958774
0.4	0.7861317056154
0.5	0.8514267017599
0.6	0.9174968894121
0.7	0.9811441017109
0.8	0.1040720867913

1. Construir la aproximación lineal por mínimos cuadrados y dibujarla frente al conjunto de datos.
2. Construir las aproximaciones por mínimos cuadrados mediante polinomios de segundo y de tercer grado. Representar gráficamente las curvas y compararlas con el conjunto de puntos.

Práctica puntuable t (0.3 puntos) Encontrar, con una tolerancia de $1e-5$, todas las soluciones positivas de las siguientes ecuaciones empleando para ello los comandos `fzero`, `g_newton` y `g_brent`:

(1) $\mathrm{tg}(x) - x + 1 = 0,\ 0 < x < 3\pi$ (2) $\mathrm{sen}(x) - 0.3e^x = 0,\ x > 0$

(3) $0.1x^3 - 5x^2 - x + 4 + e^{-x} = 0$ (4) $\log(x) - 0.2x^2 + 1 = 0$

(5) $x + x^2 + 3x^{-1} - 40 = 0$ (6) $0.5\exp(x/3) - \mathrm{sen}(x) = 0$

(7) $\log(1+x) - x^2 = 0,\ x \in [0;2]$ (8) $\exp(x) - 5x^2,\ x \in [0;5]$

(9) $x^3 + 2x - 1 = 0$ (10) $\sqrt{x+2} - x = 0$

(11) $e^x - \frac{1}{\mathrm{sen}\, x},\ x \in [0.1, 3.2]$ (12) $x^{1.4} - \sqrt{x} + 1/x - 100 = 0, \quad x > 0$

Práctica puntuable u (0.3 puntos) La función definida por la igualdad

$$T_n(x) = \cos(n \arccos x), \quad n \in \mathbb{N} \tag{G.25}$$

recibe el nombre de *polinomio de Chebychev* de orden n. Se pide:

1. De la igualdad general $\cos a \cos b = \frac{1}{2}[\cos(a+b) + \cos(a-b)]$ deducir la siguiente ecuación en recurrencia que verifican las funciones de Chebychev

$$2xT_n(x) = T_{n+1}(x) + T_{n-1}(x) \tag{G.26}$$

deduciéndose de ello que $T_n(x)$ es una función polinómica para todo $n \in \mathbb{N}$ cuyo primer coeficiente es 2^{n-1} y que $T_0 = 1$, $T_1(x) = x - 1$, $T_2(x) = 2x^2 - 1, \ldots$.
(Indicación: En la fórmula inicial tomar $a = n \arccos x$ y $b = \arccos x$)

2. Definir una función propia, `Tn=g_cheby(n)`, de modo que calcule para cada $n \in \mathbb{N}$ el polinomio de Chebychev de orden n.
3. Todas las raíces de $T_n(x)$ son reales, pertenecen al intervalo $[-1,1]$ y además coinciden con

$$r_k = \cos(\frac{2k+1}{2n}\pi), \quad k = 0, 1, \ldots, n-1 \tag{G.27}$$

Los puntos $\{r_k\}$, $k = 0, 1, \ldots, n-1$, reciben el nombre de *nodos de Chebychev* del intervalo $[-1,1]$ y por semejanza obtenemos los puntos:

$$\overline{r}_k = \frac{a+b}{2} + \frac{b-a}{2}\cos(\frac{2k+1}{2n}\pi), \quad k = 0, 1, \ldots, n-1 \tag{G.28}$$

que son los nodos de Chebychev en el intervalo $[a,b]$.

4. Denotemos por $L_n = \prod_{i=1}^{n}(x - s_i)$ donde $s_k = -1 + \frac{2}{n-1}k$, $k = 0, 1, \ldots, n-1$, son n puntos equiespaciados en el intervalo $[-1,1]$. Se pide representar conjuntamente en $[-1,1]$ las funciones polinomiales $\frac{1}{2^{n-1}}T_n$ y L_n para $n = 5, 7, 10$.
5. De las representaciones anteriores comprobar que $\max|\frac{1}{2^{n-1}}T_n(x)| \leq \max|L_n(x)|$, $x \in [-1,1]$, para $n = 5, 7, 10$. De hecho se verifica que el polinomio mónico $\frac{1}{2^{n-1}}T_n$ está caracterizado por ser el que menor máximo tiene en $[-1,1]$ de entre todos los polinomios mónicos de orden n. Esto tiene como consecuencia (ver [6], sección 7.2.7) que la interpolación de una función f mediante un polinomio interpolador de grado n sea óptima precisamente cuando tomamos como nodos los nodos de Chebychev.

Práctica puntuable v (0.3 puntos) Realizar la implementación del método de la regula falsi para el cálculo de ceros de una función tal y como se describe en el apartado G.7.