PRÁCTICA 8: Interpolación a Trozos. Mínimos Cuadrados

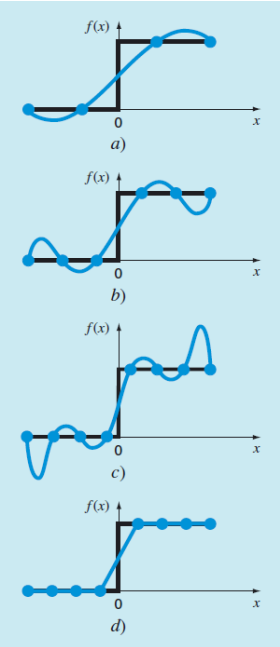
8.1 Resumen de Teoría

8.1.1 Interpolación a Trozos: Splines

En la práctica anterior, se usaron polinomios de *n*-ésimo grado para interpolar entre *n* + 1 puntos que se tenían como datos. Por ejemplo, para ocho puntos se puede obtener un perfecto polinomio de séptimo grado. No obstante, hay casos donde estas funciones llevarían a resultados erróneos a causa de los errores de redondeo y los puntos lejanos.

Un procedimiento alternativo consiste en colocar polinomios de grado inferior en subconjuntos de los datos. Tales polinomios conectores se denominan *trazadores* o *splines*. Por ejemplo, las curvas de tercer grado empleadas para unir cada par de datos se llaman *splines cúbicos*. Esas funciones se pueden construir de tal forma que las conexiones entre ecuaciones cúbicas adyacentes resulten visualmente suaves.

Podría parecer que la aproximación de tercer grado de los splines sería inferior a la expresión de séptimo grado. La figura muestra una situación donde un spline se comporta mejor que un polinomio de grado superior. Éste es el caso donde una función en general es suave, pero presenta un cambio abrupto en algún punto.



Los casos *a*) a *c*) indican que el cambio abrupto induce oscilaciones en los polinomios de interpolación. En contraste, un spline lineal *d*) ofrece una aproximación mucho más aceptable.

8.1.2 Spline Lineal

La unión más simple entre dos puntos es una recta. Las interpolaciones a trozos de primer grado, para un grupo de datos ordenados, se definen como un conjunto de funciones lineales:

$$f(x) = f(x_0) + m_0(x - x_0),\ con\ \ x_0 \le x \le x_1,$$

$$f(x) = f(x_1) + m_1(x - x_1),\ con\ \ x_1 \le x \le x_2,$$

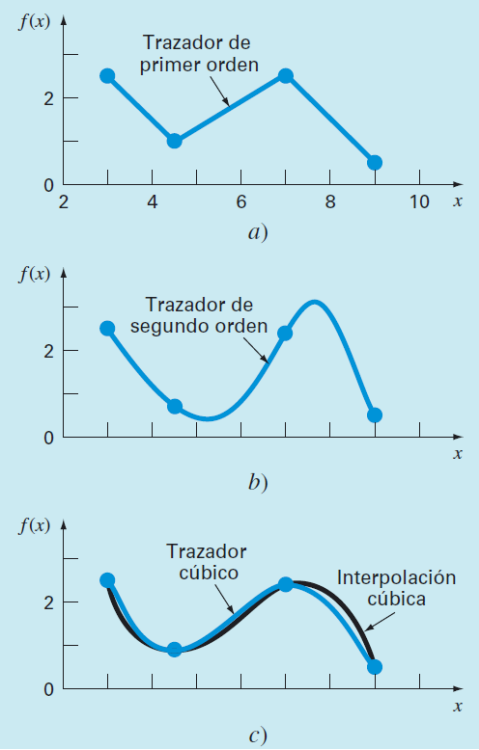
:

$$f(x) = f(x_{n-1}) + m_{n-1}(x - x_{n-1}),\ con\ \ x_{n-1} \le x \le x_n,$$

donde la pendiente m_i de la línea recta que une los puntos es:

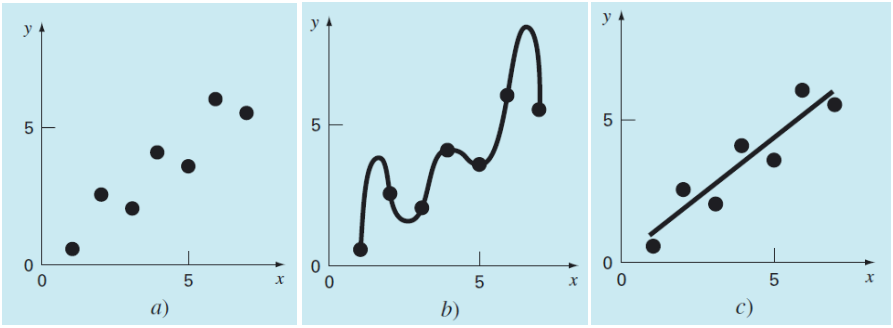
$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Estas relaciones se usan para evaluar la función en cualquier punto intermedio entre nodos.



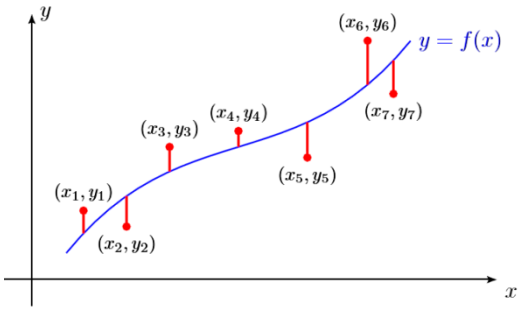
8.1.3 Regresión por Mínimos Cuadrados

Cuando los datos tienen errores sustanciales, la interpolación polinomial es inapropiada y puede dar resultados poco satisfactorios cuando se utiliza para predecir valores intermedios. Con frecuencia los datos experimentales en ingeniería son de este tipo.



Supongamos que queremos determinar una función $y = f(x)$ que modelice de manera adecuada algún fenómeno, del que disponemos de n observaciones. En el método de los mínimos cuadrados la función $f(x)$ se halla de manera que sea mínimo el error cuadrático:

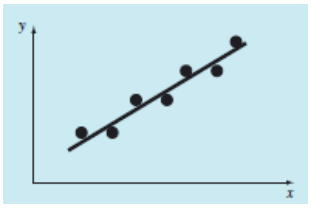
$$\sum_{i=1}^{n} |y_i - f(x_i)|^2$$



Regresión Lineal

El ejemplo más simple de una aproximación por mínimos cuadrados es ajustar una línea recta a un conjunto de observaciones definidas por puntos:

$$f(x) = a_0 + a_1 x$$



Para determinar los valores de a_0 y a_1, la ecuación:

$$\sum_{i=1}^{n} |e_i|^2 = \sum_{i=1}^{n} |y_i - f(x_i)|^2 = \sum_{i=1}^{n} |y_i - a_0 - a_1(x_i)|^2$$

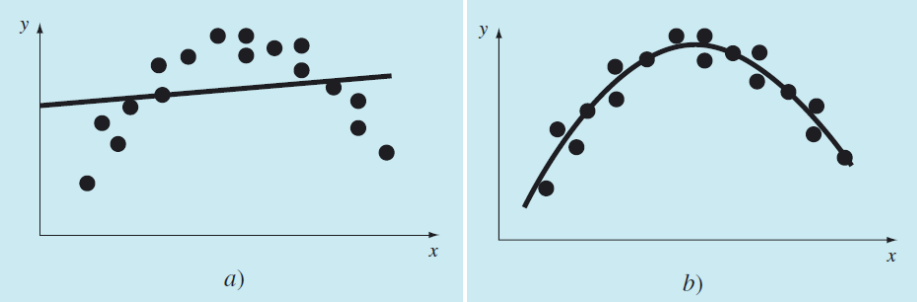
se deriva con respecto a cada uno de los coeficientes. Al igualar estas derivadas a cero, dará como resultado un *error cuadrático* mínimo. Si se hace esto, las ecuaciones se expresan como:

$$\left.\begin{aligned} n a_0 + \left(\sum x_i\right) a_1 &= \sum y_i \\ \left(\sum x_i\right) a_0 + \left(\sum {x_i}^2\right) a_1 &= \sum x_i\, y_i \end{aligned}\right\}$$

donde todas los sumatorios van desde i = 1 hasta n. Este sistema de dos ecuaciones lineales, con dos incógnitas a_0 y a_1, lo podemos resolver con los métodos conocidos.

Regresión Cuadrática

En ingeniería, algunos datos exhiben un patrón marcado, como el que se advierte en la siguiente figura. Estos datos son pobremente representados por una línea recta. En este caso parece más adecuado ajustar los datos mediante *regresión polinomial*.



El procedimiento de mínimos cuadrados se puede extender fácilmente al ajuste de datos con un polinomio de grado superior. Por ejemplo, suponga que ajustamos un polinomio de segundo grado o cuadrático:

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

Al seguir el procedimiento de la sección anterior, obtenemos la derivada de la ecuación del error con respecto a cada uno de los coeficientes desconocidos del polinomio. Estas ecuaciones se igualan a cero y se reordenan para desarrollar el siguiente conjunto de ecuaciones normales:

$$\left.\begin{aligned} n a_0 + \left(\sum x_i\right) a_1 + \left(\sum {x_i}^2\right) a_2 &= \sum y_i \\ \left(\sum x_i\right) a_0 + \left(\sum {x_i}^2\right) a_1 + \left(\sum {x_i}^3\right) a_2 &= \sum x_i\, y_i \\ \left(\sum {x_i}^2\right) a_0 + \left(\sum {x_i}^3\right) a_1 + \left(\sum {x_i}^4\right) a_2 &= \sum {x_i}^2\, y_i \end{aligned}\right\}$$

donde todas los sumatorios van desde i = 1 hasta n. Observe que las tres ecuaciones anteriores son lineales y tienen tres incógnitas: a_0, a_1 y a_2 . Los coeficientes de las incógnitas se evalúan de manera directa, a partir de los datos observados.

En este caso, observamos que el problema de determinar un polinomio de segundo grado por mínimos cuadrados es equivalente a resolver un sistema de tres ecuaciones lineales simultáneas.

El caso bidimensional se extiende con facilidad a un polinomio de m-ésimo grado.

8.2 Splines

En la práctica las aproximaciones más habituales son la interpolación lineal a trozos, y el spline cúbico. Antes de ver los comandos que implementa Maxima para su cálculo, veamos sobre un ejemplo como aplicar lo visto en teoría.

8.2.1 Spline Lineal

Consideremos como ejemplo la función:

$$f(x) = x + \cos 3x$$

Vamos a construir 10 puntos x_i igualmente espaciados en el intervalo [0, 4] y vamos a obtener la aproximación obtenida por interpolación lineal a trozos correspondiente a los datos:

$$x = (x_i),\ y = f(x_i)$$

```
(%i1)    kill(all)$
(%i1)    f(z):=z+cos(3*z)$
(%i4)    x:makelist(i,i,0,4,4/(10-1));
         y:makelist(i+cos(3*i),i,0,4,4/(10-1)),numer$
         transpose(y);
```

$$(x)\quad [0, \frac{4}{9}, \frac{8}{9}, \frac{4}{3}, \frac{16}{9}, \frac{20}{9}, \frac{8}{3}, \frac{28}{9}, \frac{32}{9}, 4]$$

$$(\%o4)\quad \begin{bmatrix} 1.0 \\ 0.6796820177474339 \\ -4.376793241525068\,10^{-4} \\ 0.6796897124697213 \\ 2.359581267636949 \\ 3.149589925273197 \\ 2.521166632858052 \\ 2.115289258322854 \\ 3.23254615718018 \\ 4.843853958732495 \end{bmatrix}$$

Aunque existen diversas formas de hacerlo, hemos optado por trabajar con listas. Es sencillo y fácil de programar. Tan sólo debemos tener cuidado, como siempre con el contador.

Por ejemplo si queremos tomar n nodos en el intervalo $[a, b]$, con el comando **makelist** hacemos variar la i entre a y b, con un paso de $(a - b)/(n - 1)$, pues, como bien sabemos, n nodos producen $(n - 1)$ intervalos. Esta forma de trabajar ya la utilizamos en la Práctica 7 y es típica del problema donde en vez de darnos los puntos, nos dan una función $f(x)$ y nosotros seleccionamos los nodos.

Luego, cuando usemos las listas x e y, debemos recordar que el primer elemento es $x_1 = x[1]$, $y_1 = y[1]$ y no $x_0 = x[0]$, $y_0 = y[0]$ como suele denotarse en teoría. El último será $x_n = x[n]$, $y_n = y[n]$, pues tenemos n nodos.

Vamos ahora a construir las rectas que constituyen la interpolación lineal. Habrá $(n-1)$, tantas como intervalos. En este caso serán 9. Comenzamos calculando las pendientes y luego las rectas.

```
(%i5)    m:makelist((y[i+1]-y[i])/(x[i+1]-x[i]),i,1,9),numer$

(%i8)    splin1in:makelist(y[i]+m[i]*(z-x[i]),i,1,9),numer$
         splin1in(z):=expand(splin1in)$
         msp:transpose(splin1in(z))$
```

Para hacernos una idea sobre el intervalo de validez de cada recta, añadimos sobre la matriz **msp** anterior dos columnas que así nos lo indican.

```
(%i9)    addcol(msp,delete(x[10],x),delete(x[1],x));
```

(%o9)

$1.0-0.7207154600682738\,z$	0	$\frac{4}{9}$
$1.35980171481902-1.530269318411069\,z$	$\frac{4}{9}$	$\frac{8}{9}$
$1.530286631536216\,z-1.3606924629119$	$\frac{8}{9}$	$\frac{4}{3}$
$3.779755999126261\,z-4.35998495303196$	$\frac{4}{3}$	$\frac{16}{9}$
$1.777519479681559\,z-0.800453362908045$	$\frac{16}{9}$	$\frac{20}{9}$
$6.291706387348928-1.413952407934078\,z$	$\frac{20}{9}$	$\frac{8}{3}$
$4.95643088006924-0.9132240927041956\,z$	$\frac{8}{3}$	$\frac{28}{9}$
$2.513828022428985\,z-5.705509033678434$	$\frac{28}{9}$	$\frac{32}{9}$
$3.625442553492709\,z-9.657916255238339$	$\frac{32}{9}$	4

Ahora vamos a definir una función a trozos mediante un bloque usando los comandos **for** e **if** la cual representa el conjunto de todos los splines lineales.

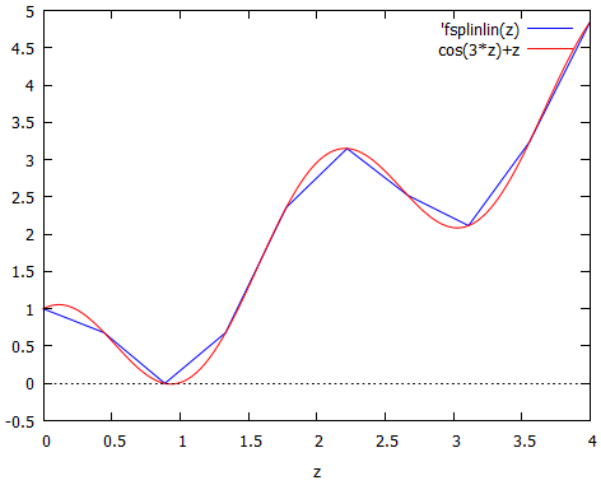
```
(%i10)   fsplinlin(z):=block([v],for i:1 thru 9 do (
                  if(x[i]<=z and z<=x[i+1]) then
                 v:ev(splin1in(z)[i])),v)$
```

A continuación, vamos a hallar los errores absolutos que se cometen con la aproximación anterior en $x = 1$, $x = 2$, y $x = 3$. Evidentemente en los nodos el error es cero.

```
(%i14)   abs(f(1)-fsplinlin(1)),numer;
         abs(f(2)-fsplinlin(2)),numer;
         abs(f(3)-fsplinlin(3)),numer;
         abs(f(4/9)-fsplinlin(4/9)),numer;
(%o11)   0.1595866652247615
 (%o12)   0.2055846901952929
 (%o13)   0.12788886384133
 (%o14)   0.0
```

Finalmente representamos, en dicho intervalo, la función *f*(*z*) junto con la aproximación obteniendo:

```
(%i15)   plot2d(['fsplinlin(z),f(z)],[z,0,4])$
```



En la gráfica anterior hemos usado el operador **comilla** ‘. Este operador aplicado sobre una función impide la evaluación de la función aunque los argumentos de la función sí se evalúan. De esta forma somos capaces de realizar su gráfica.

8.3 El paquete interpol

Maxima dispone del paquete **interpol** que permite interpolar un conjunto de datos por una función lineal a trozos es decir por la poligonal que pasa por ellos, por un polinomio o por splines cúbicos, para ello se comienza cargando dicho paquete mediante la orden:

load(interpol)

Los datos a interpolar, pueden venir en forma de pares, o como una matriz de dos columnas:

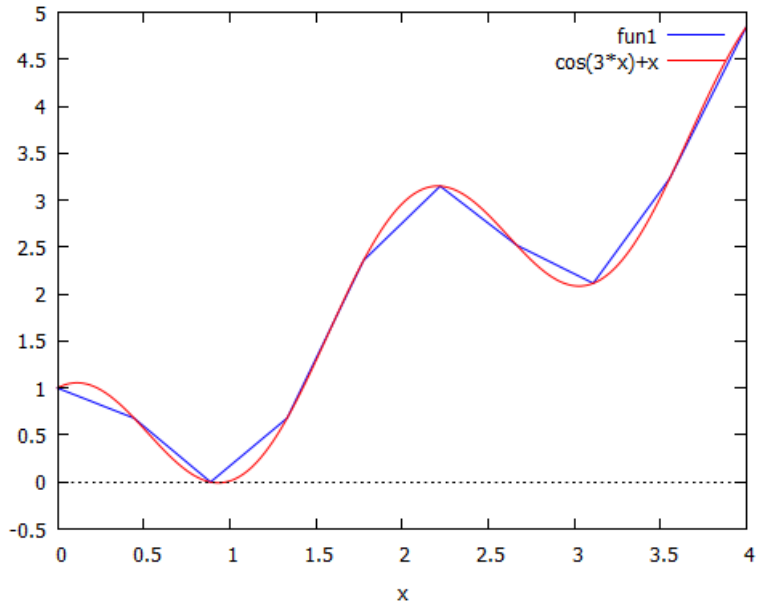
datos: [[x_1, y_1], [x_2, y_2],…]
datos: matrix([x_1, y_1], [x_2, y_2],…])

Los comandos disponibles son:

linearinterpol(datos) = si se requiere la función poligonal que pasa por los puntos dados.
lagrange(datos) = si se requiere el polinomio de Lagrange que pasa por esos puntos.
cspline(datos) = si se requiere el splin cúbico natural con esos datos.

Vamos a aplicar estos comandos para obtener los splines lineales y cúbicos para la función anterior, y así comprobar los resultados. Comenzamos con los lineales (10 nodos).

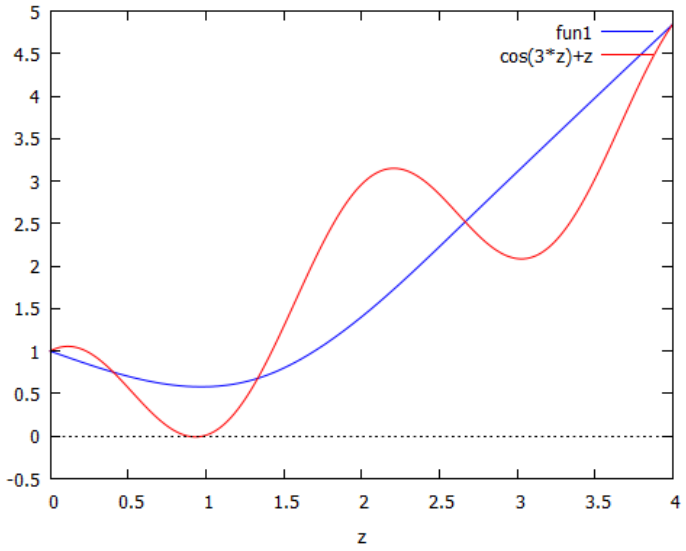
```
(%i1)    kill(all)$
(%i1)    load(interpol)$
(%i2)    f(z):=z+cos(3*z)$
(%i4)    x:makelist(i,i,0,4,4/(10-1))$
         y:makelist(i+cos(3*i),i,0,4,4/(10-1)),numer$
(%i5)    datos:makelist([x[i],y[i]],i,1,10)$
(%i6)    sol:linearinterpol(datos)$
(%i7)    plot2d([ev(sol,x=z),f(z)],[z,0,4])$
```



La solución devuelta por Maxima mediante el comando **linearinterpol** viene, por defecto, sobre la variable *x*, por eso debemos evaluarla sobre *z* que es el nombre de nuestra variable.

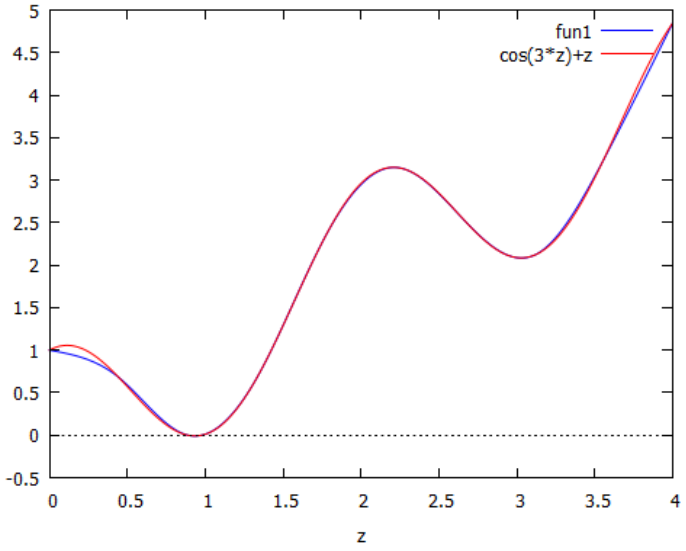
Ahora para los splines cúbicos (4 nodos).

```
(%i9)    x:makelist(i,i,0,4,4/(4-1))$
         y:makelist(i+cos(3*i),i,0,4,4/(4-1)),numer$
(%i10)   datos:makelist([x[i],y[i]],i,1,4)$
(%i11)   sol:cspline(datos)$
(%i12)   plot2d([ev(sol,x=z),f(z)],[z,0,4])$
```



Ahora aumentamos el número de nodos para los splines cúbicos (10 nodos).

```
(%i14)   x:makelist(i,i,0,4,4/(10-1))$
         y:makelist(i+cos(3*i),i,0,4,4/(10-1)),numer$
(%i15)   datos:makelist([x[i],y[i]],i,1,10)$
(%i16)   sol:cspline(datos)$
(%i17)   plot2d([ev(sol,x=z),f(z)],[z,0,4])$
```



Obteniendo un magnífico resultado, como era de esperar.

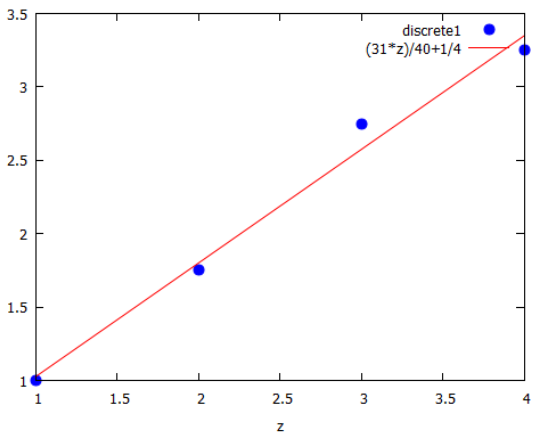
8.4 Regresión por Mínimos Cuadrados

Para finalizar esta práctica veamos la otra técnica de aproximación: la regresión por mínimos cuadrados. Comenzaremos, como siempre, por implementar las fórmulas que vimos en teoría, para posteriormente, comprobar la solución con los comandos Maxima. Consideremos un primer ejemplo de regresión lineal: el caso más simple de una aproximación por mínimos cuadrados, ajustando a una línea recta: $f(x) = a_0 + a_1 x$ un conjunto de puntos.
Para minimizar el *error cuadrático* debe cumplirse:

$$\left.\begin{aligned} n a_0 + \left(\sum x_i\right) a_1 &= \sum y_i \\ \left(\sum x_i\right) a_0 + \left(\sum {x_i}^2\right) a_1 &= \sum x_i\, y_i \end{aligned}\right\}$$

Este sistema de dos ecuaciones lineales, con dos incógnitas a_0 y a_1, lo podemos resolver con el comando **linsolve**.

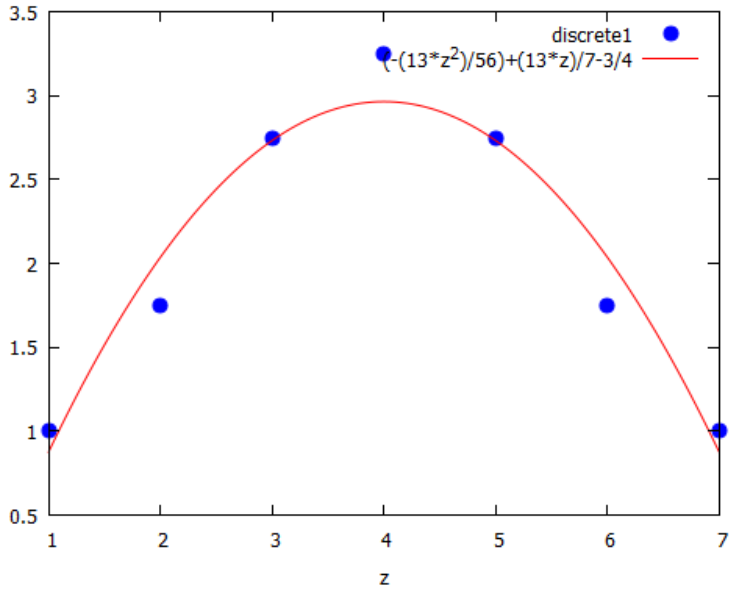
```
(%i1)   kill(all)$;
(%i3)   x:[1,2,3,4]$
        y:[1,7/4,11/4,13/4]$
        n:length(x)$
(%i4)   sis:[
        n*a0+sum(x[i],i,1,n)*a1=sum(y[i],i,1,n),
        sum(x[i],i,1,n)*a0+sum(x[i]^2,i,1,n)*a1
        =sum(x[i]*y[i],i,1,n)];
(sis)   [10 a1+4 a0= 35/4 , 30 a1+10 a0= 103/4 ]
(%i5)   sol:linsolve(sis,[a0,a1]);
(sol)   [a0= 1/4 , a1= 31/40 ]
(%i6)   pol:ev(a0+a1*z,sol);
(pol)   31 z/40 + 1/4
(%i7)   plot2d([[discrete,x,y],pol],[z,1,4],
            [style,points,lines])$
```



Para el caso cuadrático: $f(x) = a_0 + a_1 x + a_2 x^2$ el procedimiento es idéntico. Sólo cambia el sistema a resolver que ahora es:

$$\left.\begin{aligned} n a_0 + \left(\sum x_i\right) a_1 + \left(\sum {x_i}^2\right) a_2 &= \sum y_i \\ \left(\sum x_i\right) a_0 + \left(\sum {x_i}^2\right) a_1 + \left(\sum {x_i}^3\right) a_2 &= \sum x_i\, y_i \\ \left(\sum {x_i}^2\right) a_0 + \left(\sum {x_i}^3\right) a_1 + \left(\sum {x_i}^4\right) a_2 &= \sum {x_i}^2\, y_i \end{aligned}\right\}$$

```
(%i1)   kill(all)$
(%i3)   x:[1,2,3,4,5,6,7]$
        y:[1,7/4,11/4,13/4,11/4,7/4,1]$
        n:length(x)$

(%i4)   sis:[n*a0+sum(x[i],i,1,n)*a1+
            sum(x[i]^2,i,1,n)*a2=sum(y[i],i,1,n),
        sum(x[i],i,1,n)*a0+sum(x[i]^2,i,1,n)*a1+
            sum(x[i]^3,i,1,n)*a2=sum(x[i]*y[i],i,1,n),
        sum(x[i]^2,i,1,n)*a0+sum(x[i]^3,i,1,n)*a1+
            sum(x[i]^4,i,1,n)*a2=sum(x[i]^2*y[i],i,1,n)]$

(%i5)   sol:linsolve(sis,[a0,a1,a2]);
(sol)   [a0=-3/4,a1=13/7,a2=-13/56]
(%i6)   pol:ev(a0+a1*z+a2*z^2,sol);
(pol)   -13*z^2/56+13*z/7-3/4

(%i7)   plot2d([[discrete,x,y],pol],[z,1,7],
            [style,points,lines])$
```



8.5 El paquete lsquares

Para resolver el problema de mínimos cuadrados, Maxima incorpora el paquete **lsquares**. Comenzaremos por cargarlo con la orden:

load(lsquares)

Entre otros comandos que podemos usar destacamos:

lsquares_estimates (dat, var, ecu, par)
lsquares_mse (dat, var, ecu)
lsquares_residual_mse (dat, var, ecu, par)

El primer comando **lsquares_estimates** estima los parámetros **par** que mejor se ajusten a la ecuación **ecu** de variables **var** y a los datos **dat**, por el método de los mínimos cuadrados. La función busca primero una solución exacta, y si no la encuentra, buscará una aproximada. El resultado es una lista de listas de ecuaciones con los parámetros. Los datos deben darse en formato matricial. Cada fila es un dato y las columnas contienen los valores para cada una de las variables.

Una vez obtenidos los valores de los parámetros con el comando anterior, el comando **lsquares_residual_mse** devuelve el error medio cuadrático, definido por **lsquares_mse** como:

$$\frac{1}{n}\sum_{i=1}^{n}|e_i|^2=\frac{1}{n}\sum_{i=1}^{n}|y_i-f(x_i)|^2$$

Resolvamos de nuevo los problemas anteriores.

```
(%i1)   kill(all)$
(%i1)   M : matrix ([1, 1], [2, 7/4], [3, 11/4], [4, 13/4])$
(%i2)   load (lsquares)$
(%i3)   sol:lsquares_estimates(
              M, [x,y], y=a0+a1*x, [a0,a1]);
```

$$(sol)\quad [[a0=\frac{1}{4},a1=\frac{31}{40}]]$$

```
(%i4)   mse:lsquares_mse(
              M, [x,y], y=a0+a1*x);
```

$$(mse)\quad \frac{\sum_{i=1}^{4}\left(M_{i,2}-a1\,M_{i,1}-a0\right)^2}{4}$$

Y usando los valores ya obtenidos de los parámetros y que guardamos en *sol* obtenemos el error:

```
(%i5)   mse:lsquares_residual_mse (
            M, [x,y], y=a0+a1*x,sol[1]);
```

$$(mse)\quad \frac{7}{640}$$

Valor que podemos fácilmente comprobar sin más que aplicar la definición:

```
(%i6)  1/4*ev(sum(abs(M[i,2]-(a0+a1*M[i,1]))^2,i,1,4),sol);
```

$$(\%o6)\quad \frac{7}{640}$$

Para el segundo ejemplo de regresión cuadrática tenemos:

```
(%i7)  M : matrix ([1, 1], [2, 7/4], [3, 11/4], [4, 13/4],
       [5,11/4],[6,7/4],[7,1])$
(%i8)  sol:lsquares_estimates(
             M, [x,y], y=a0+a1*x+a2*x^2, [a0,a1,a2]);
```

$$(sol)\quad [[a0=-\frac{3}{4}, a1=\frac{13}{7}, a2=-\frac{13}{56}]]$$

```
(%i9)  mse:lsquares_residual_mse (
           M, [x,y], y=a0+a1*x+a2*x^2,sol[1]);
```

$$(mse)\quad \frac{31}{784}$$

Nota: Si queremos aproximar por un polinomio de mayor grado y no queremos deducir la fórmula general podemos utilizar Maxima para hallarlo. Por ejemplo, para aproximar los puntos por un polinomio de grado 3:

$$y = a_0 + a_1x + a_2x^2 + a_3x^3$$

la función a minimizar será:

$$\sum_{i=1}^{n} |e_i|^2 = \sum_{i=1}^{n} |y_i - a_0 - a_1(x_i) - a_2(x_i)^2 - a_3(x_i)^3|^2$$

Así pues resolvemos con Maxima el sistema que resulta de imponer las tres derivadas parciales iguales a cero. Veamos un ejemplo donde vamos a aproximar los valores dados en 6 puntos mediante un polinomio de grado 3.

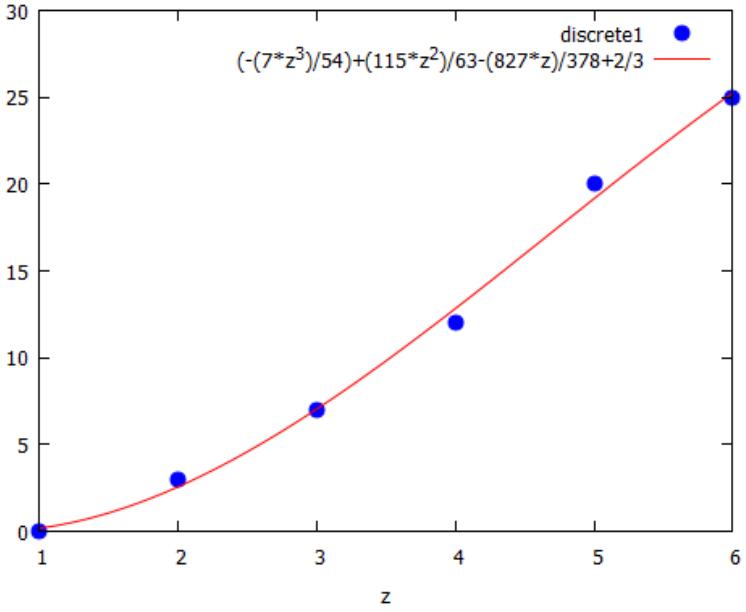
```
(%i1)  kill(all)$
(%i3)  x:[1,2,3,4,5,6]$
       y:[0,3,7,12,20,25]$
       n:length(x)$

(%i4)  err:sum((y[i]-(a0+a1*x[i]+a2*x[i]^2+a3*x[i]^3))^2
                    ,i,1,n)$

(%i5)  sol:linsolve([diff(err,a0)=0,diff(err,a1)=0,
              diff(err,a2)=0,diff(err,a3)=0]
          ,[a0,a1,a2,a3]);
```

$$(sol)\quad [a0=\frac{2}{3}, a1=-\frac{827}{378}, a2=\frac{115}{63}, a3=-\frac{7}{54}]$$

```
(%i6)    pol:ev(a0+a1*z+a2*z^2+a3*z^3,sol);
```

$$(pol)\quad -\frac{7\,z^3}{54}+\frac{115\,z^2}{63}-\frac{827\,z}{378}+\frac{2}{3}$$

```
(%i7)    plot2d([[discrete,x,y],pol],[z,1,6],
             [style,points,lines])$
```



Y lo comprobamos con la ayuda del paquete **lsquares**.

```
(%i1)    kill(all)$
(%i1)    load (lsquares)$
(%i2)    dat:matrix([1,0],[2,3],[3,7],[4,12],[5,20],[6,25])$

(%i3)    sol:lsquares_estimates(
               dat, [x,y], y=a0+a1*x+a2*x^2+a3*x^3,
              [a0,a1,a2,a3]);
```

$$(sol)\quad [[a0=\frac{2}{3},a1=-\frac{827}{378},a2=\frac{115}{63},a3=-\frac{7}{54}]]$$

```
(%i4)    mse:lsquares_residual_mse (
             dat, [x,y], y=a0+a1*x+a2*x^2+a3*x^3,sol[1]);
```

$$(mse)\quad \frac{53}{189}$$

Nota: Con este método que consiste en minimizar el error, resolviendo el sistema de derivadas parciales iguales a cero, podemos ajustar por mínimos cuadrados, utilizando un sistema de funciones (siempre linealmente independientes), que no sean polinomios (por ejemplo, funciones trigonométricas, exponenciales, etc.). En los ejercicios propuestos veremos algún caso.

Nota: Si se desea ajustar, utilizando mínimos cuadrados, un conjunto de $n + 1$ puntos por un polinomio de grado n, obtendremos el polinomio de interpolación, ya que, como éste pasa por todos los puntos, la suma de errores al cuadrado será nula y, evidentemente, ése es el valor mínimo posible.

8.6 Ejercicios Propuestos

Ejercicio 1. Consideremos 21 puntos igualmente distribuidos en el intervalo [-1,1] de la función:

$$f(x) = \frac{1}{1+x^2}$$

Calcular el polinomio de Lagrange de interpolación. Hacer también splines lineales con nuestro método y con el paquete **interpol**. Dibujar e interpretar los resultados. Comparar la función y las dos aproximaciones obtenidas en el punto 0.06.

Ejercicio 2. Repetir el problema anterior en el intervalo [-5,5]. Comentar los resultados.

Ejercicio 3. Usando el paquete **interpol** repite el ejercicio anterior pero ahora con splines cúbicos. Compara la precisión alcanzada en el punto 0.06.

Ejercicio 4. Ajustar, mediante una recta de regresión, los datos:

x: 0, 1, 2, 2.5, 3
y: 2.9, 3.7, 4.1, 4.4, 5

Utilizar las ecuaciones vistas en teoría y comprobar el resultado con el paquete **lsquares**.

Ejercicio 5. La densidad relativa *d* del aire depende de la altura *h*, habiéndose obtenido las siguientes mediciones:

h (km): 0, 1.525, 3.05, 4.575, 6.1, 7.625, 9.15
d (kg/m3): 1, 0.8617, 0.7385, 0.6292, 0.5328, 0.4481, 0.3741

Ajustar los datos anteriores por una parábola de regresión y estimar la densidad del aire a 5 km de altitud. Utilizar las ecuaciones vistas en teoría y comprobar el resultado con el paquete **lsquares**.

Ejercicio 6. Comentamos antes que con el método de minimizar el error, resolviendo el sistema de derivadas parciales iguales a cero, podemos ajustar por mínimos cuadrados, utilizando un sistema de funciones (linealmente independientes), que no sean polinomios. Resolver de nuevo el ejercicio:

x: [1, 2, 3, 4, 5, 6]
y: [0, 3, 7, 12, 20, 25]

usando:

$$f(x) = a_0 + a_1 e^x$$

Comprobar el resultado con el paquete **lsquares**.

Ejercicio 7. Consideremos la función:

$$f(x) = x + \cos 3x$$

realizar la interpolación con splines cúbicos para 4 nodos y para 5 nodos, usando el paquete **interpol**.

Ejercicio 8. Comentamos antes que al ajustar por mínimos cuadrados, un conjunto de $n + 1$ puntos por un polinomio de grado n, obtendremos el polinomio de interpolación. Comprobarlo con los siguientes datos:

x: [1, 2, 3]
y: [0, 3, 7]

usando una parábola de regresión.