Matemáticas II - Grado en Ingeniería Química

Práctica 0. Introducción.

Una manera de seguir las prácticas consiste en ejecutarlas como una demo. En esta introducción mostramos los pasos para ejecutar una demo y veremos algunos ejemplos.

Contents

Uso de una demo

Una demo contiene textos y grupos de instrucciones que se van ejecutando paso a paso en la ventana de 'Comandos' (Command Window). Por ejemplo la siguiente instrucción calcula el valor de la función seno tomando como argumento 3*pi/2 radianes. Puede observar la instrucción y el resultado.

```
sin(3*pi/2)
```

```
ans =

    -1
```

Tras cada grupo de instrucciones se produce una pausa, y aparece (en la parte superior de la ventana de 'Comandos') la opción de continuar con el siguiente grupo de instrucciones 'Next', de finalizar la demo 'Stop' o de abrir en el editor la demo pinchando en el nombre de la demo. Si quisiera ver un grupo de instrucciones ejecutadas con anterioridad, podría desplazar hacia arriba la ventana de 'Comandos', o parar la demo y volver a ejecutarla desde el principio. -> Pulse ahora 'Next' para ejecutar el siguiente grupo de instrucciones.

Ejecutar una demo

Para ejecutar una demo se puede introducir el comando 'echodemo nombredelademo.m' suponiendo que tenga el archivo descargado en el 'directorio actual' (Current Directory). Puede cambiar el directorio actual pinchando con el ratón a la derecha de la barra de 'directorio actual'.

A partir de ahora no se le pedirá que pulse en 'Next' para ejecutar el siguiente grupo de instrucciones.

Al acabar cada grupo de instrucciones aparece en la ventana de 'Comandos' el símbolo >> que indica que puede introducir comandos antes de seguir (reproduciendo) la demo.

Tras algunos grupos de instrucciones aparecerán ejercicios que debe realizar, tecleando las instrucciones necesarias, antes de continuar con la demo.

Ej 1: Teclee en la ventana 'Command Window' la instrucción 2+2 y pulse la tecla Enter. Compruebe que ha obtenido el resultado 4 antes de continuar con el siguiente grupo de instrucciones de esta demo. (Recuerde qué debe pulsar al acabar el ejercicio para ejecutar las siguientes instrucciones de la demo)

Uso de la ayuda

En cualquier momento puede pulsar la tecla F1 para abrir la ventana de ayuda, o pedir ayuda en la ventana de comandos con la instrucción help, como por ejemplo.

```
help echodemo
```

```
  ECHODEMO Run a cell script as an echo-and-pause command line demo.
     ECHODEMO FILENAME displays a cell script FILENAME.  Cell scripts can be
     created using Cell Mode in the MATLAB Editor. The demo is advanced by
     clicking on HTML hypertext links embedded in the command window.

     ECHODEMO('FILENAME',CELLINDEX) evaluates the cell number given by
     CELLINDEX.

     Reference page in Help browser
        doc echodemo
```

Observe que se le ha mostrado, en inglés, ayuda sobre el comando echodemo, y que aunque en la ayuda los comandos aparezcan en mayúscula 'ECHODEMO' debe teclearlos en minúsculas.

Otra forma de obtener ayuda consiste, como le ha indicado la ayuda que acabamos de ver, en ejecutar el comando doc.

Ej 2: Introduzca el comando 'doc echodemo' y cierre la ventana de ayuda para seguir con esta demo.

Orden de las operaciones

Al evaluar una expresión, Matlab efectúa las potencias antes que los productos y divisiones. Y los productos y las divisiones las efectúa antes que las sumas y las restas. Observe como las siguientes parejas de instrucciones dan resultados distintos.

```
1+1/2
(1+1)/2
```

```
ans =

    1.5000


ans =

     1
```

```
2^3+1
2^(3+1)
```

```
ans =

     9


ans =

    16
```

Errores de aproximación. Error absoluto.

Si x es el valor exacto de una cantidad y x0 es una aproximación de x, llamaremos error absoluto al resultado de restar el valor exacto menos el valor aproximado. Por ejemplo, si el valor exacto es x=1.257 y consideramos como aproximación el valor x0=1.2, el error absoluto sería

```
x=1.257
x0=1.2
errorabsoluto=x-x0
```

```
x =

    1.2570


x0 =

    1.2000


errorabsoluto =

    0.0570
```

Errores de aproximación. Error relativo.

Si x es el valor exacto de una cantidad (distinta de cero) y x0 es una aproximación de x, llamamos error relativo al error absoluto dividido entre el valor exacto, es decir al resultado de (x-x0)/x. En el ejemplo anterior el error relativo sería

```
errorrelativo=(x-x0)/x
```

```
errorrelativo =

    0.0453
```

Si multiplicamos este error por 100 obtenemos que al considerar 1.2 como una aproximación de 1.257 se está cometiendo un error cercano al 4.5 por ciento.

```
errorporcentual=100*errorrelativo
```

```
errorporcentual =

    4.5346
```

Ej 3: Considere que la solución exacta de un problema es 123.456 y por determinado método se ha obtenido la aproximación 123.412. Halle el error absoluto y relativo que se comete al considerar la aproximación. Compruebe si ha obtenido los resultados que siguen. (Sol. errorabsoluto=0.044, errorrelativo=3.5640e-004)

Gráficas.

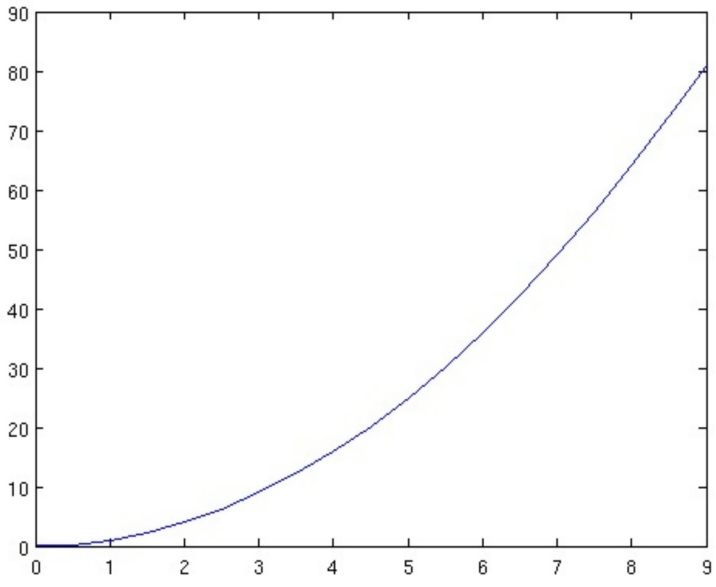
Una gráfica en Matlab puede representarse de forma sencilla usando el comando plot.

Con la instrucción plot(x,y) podemos unir un grupo de puntos para formar una gráfica. El vector x contiene las coordenadas horizontales (abscisas) de los puntos. El vector y contiene las coordenadas verticales (ordenadas).

Como ejemplo definimos un vector x con los valores 0, 0.5, 1, 1.5, ..., 8, 8.5 y 9. También definimos un vector y con los cuadrados de los elementos de x.

```
x=[0:0.5:9]
y=x.^2
plot(x,y)
```

```
x =

  Columns 1 through 7

         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

  Columns 8 through 14

    3.5000    4.0000    4.5000    5.0000    5.5000    6.0000    6.5000

  Columns 15 through 19

    7.0000    7.5000    8.0000    8.5000    9.0000


y =

  Columns 1 through 7

         0    0.2500    1.0000    2.2500    4.0000    6.2500    9.0000

  Columns 8 through 14

   12.2500   16.0000   20.2500   25.0000   30.2500   36.0000   42.2500

  Columns 15 through 19

   49.0000   56.2500   64.0000   72.2500   81.0000
```

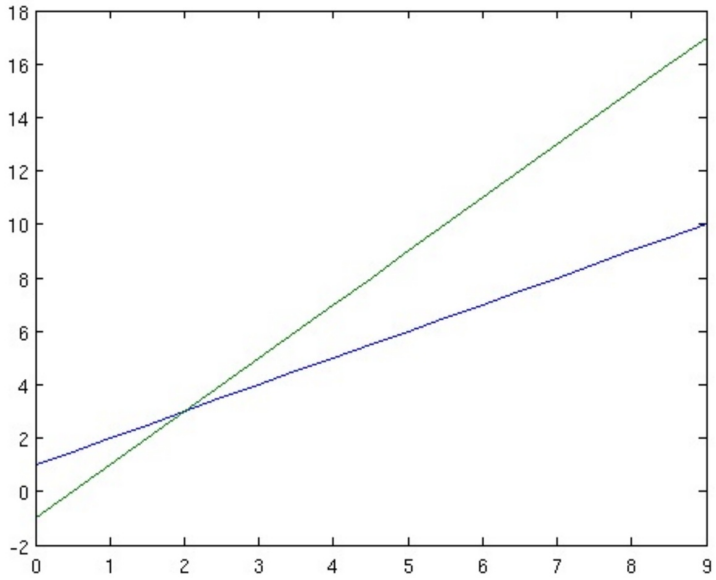


Observe que al definir el vector y estamos aplicando la operación 'elevar al cuadrado' a todas las componentes del vector x, por eso ponemos un punto antes del símbolo de potencia.

Para representar simultáneamente dos gráficas podemos emplear la instrucción plot(x1,y1,x2,y2) donde los vectores x1 e y1 contienen las coordenadas de una gráfica, y los vectores x2 e y2 contienen las coordenadas de otra gráfica.

En el siguiente ejemplo representamos conjuntamente las rectas y=x+1 e y=2x-1.

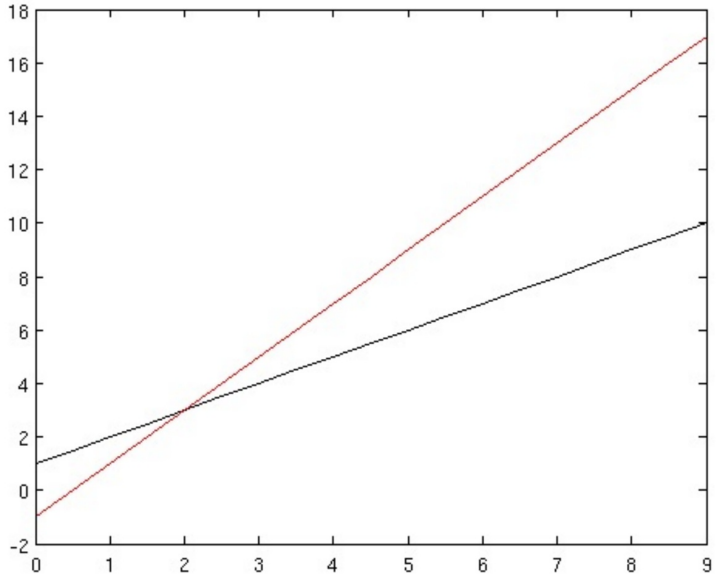
```
plot(x, x+1, x, 2*x-1)
```



Aunque en la representación anterior es fácil distinguir cuál es cada una de las rectas representadas (hazlo), sería conveniente elegir un color concreto para dibujar cada una de ellas (el comando 'help plot' proporciona información sobre las diferentes opciones disponibles).

A continuación se representan conjuntamente la recta y=x+1 (en color negro) y la recta y=2x-1 (en color rojo).

```
plot(x, x+1,'k', x, 2*x-1,'r')
```



Ejercicios finales.

Al final de cada práctica le aparecerán algunos ejercicios que deberá realizar. Si escribe los ejercicios en un script podrá usarlos más tarde, modificarlos o repasarlos. Al grabar el script use la extensión .m y no incluya en el nombre del script ni acentos ni espacios. Se recomienda poner comentarios (con el símbolo de tanto por ciento) en los archivos script, para facilitar su modificación o repaso.

Ej 4. Represente conjuntamente la gráfica del seno (en color verde) y del polinomio p(x)=x-x^3/6+x^5/120 (en color rojo) en el intervalo [-3.15,3.15]. Puede obtener la gráfica escribiendo las siguientes cuatro instrucciones.

- Defina un vector x formado por los elementos -3.15, -3.14, -3.13, ..., -0.01, 0, 0.01, 0.02, ..., 3.14, 3.15.

- Obtenga un vector f con el seno de cada uno de los valores del vector x.

- Obtenga un vector g aplicando el polinomio p(x)=x-x^3/6+x^5/120 a los elementos del vector x.

- Represente conjuntamente las gráficas de la función seno (en verde) y del polinomio p(x) (en rojo) empleando los vectores f y g.

Puede ahora observar como el polinomio aproxima a la función seno en valores más lejanos al cero representando ambas funciones en el intervalo [-7.15, 7.15].

Ej 5. Represente conjuntamente la gráfica del coseno (en color rojo) y del polinomio q(x)=1-x^2/2+x^4/24 (en color azul) en el intervalo [-3.15,3.15].

Ej 6. Obtenga a=sen(0.3) y b=p(0.3), donde p es el polinomio definido en el ejercicio 4. Halle el error absoluto que se comete al considerar b como una aproximación de a. (Solución : a=0.2955, b=0.2955, error absoluto= -4.3339e-008)

(Recuerde que el error absoluto se define como el valor exacto menos el valor aproximado.)

Observe que si se muestran a y b con cuatro cifras decimales (representación por defecto al usar 'format short') ambos valores tienen la misma representación. Al hallar la resta o emplear una representación con más decimales (como con 'format long') se puede apreciar la diferencia entre los dos valores.

Ej 7. Obtenga a=cos(0.4) y b=q(0.4), donde q es el polinomio definido en el ejercicio 5. Halle el error absoluto que se comete al considerar b como una aproximación de a. (Solución : a=0.9211, b=0.9211, error absoluto= -5.6727e-006).

Ej 8. Ejecute de nuevo esta demo con el comando 'echodemo', y tras el primer grupo de instrucciones, finalice la ejecución de la demo.

Al acabar la clase se recomienda que almacene o se envíe los script que haya creado. A continuación puede seguir una de estas dos opciones dependiendo de si hay un turno de prácticas después del suyo:

a) Mantener Matlab ejecutándose. (Con la instrucción clear puede borrar todas las variables del espacio de trabajo, y con la instrucción clc puede borrar el contenido de la ventana de 'comandos')

b) Apagar el ordenador desde el menú de Windows 'Inicio' y 'Apagar'.

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
02-Feb-2018
```

Published with MATLAB® R2013a

Práctica 1. Errores de cancelación y redondeo. Acumuladores. Vectorización.

Veremos que la manera en que Matlab almacena los números puede dar lugar a errores. También usaremos bucles para implementar acumuladores de suma y de producto, y veremos como obtener estos acumuladores sin necesidad de usar bucles, sino comandos de Matlab para manejar vectores.

Contents

Errores de redondeo

Es sabido que en la recta real, entre dos puntos dados hay infinitos números reales. También es sabido que no existe el 'menor número que es estrictamente positivo'. Esto no ocurre en Matlab, ya que sólo puede representar un número finito de números.

Por defecto Matlab representa los números con precisión finita, es decir con cierta cantidad de cifras significativas y un exponente. (Como este exponente permite colocar el punto decimal en distintas posiciones se dice que la representación es en 'punto flotante'.)

Debido a esta precisión finita se pueden producir errores de redondeo al almacenar números y efectuar operaciones.

La función eps de Matlab permite hallar la distancia entre un número y el siguiente número en la representación interna en punto flotante. Con esta precisión finita, entre estos dos números no se puede representar ningún otro.

```
dis=eps(7)
b=7+dis
```

```
dis =

   8.8818e-16


b =

    7.0000
```

Veamos que el número b es distinto del 7, pero entre el 7 y el b Matlab no puede representar ningún número. A continuación vemos que b es distinto de 7, pero c es igual a 7, aunque el número dis sea distinto de cero.

```
c=7+dis/2
b-7
c-7
```

```
c =

     7


ans =

   8.8818e-16


ans =

     0
```

Errores de cancelación

Al restar cantidades muy próximas se pueden producir errores de cancelación que suponen la pérdida de cifras significativas correctas en los resultados.

Por defecto Matlab muestra sólo unos pocos decimales de los resultados. Para poder apreciar los errores de cancelación, usaremos el comando format long, que hace que muestre más decimales. (Para volver al formato usual con menos decimales usaremos format short)

```
pi
format long
pi
```

```
ans =

    3.1416


ans =

   3.141592653589793
```

Veamos en un ejemplo este fenómeno. Supongamos que los valores exactos de a1 y a2 son los que se pretenden asignar

```
a1=0.3333333333333298
a2=0.3333333333333187
```

```
a1 =

   0.333333333333330


a2 =

   0.333333333333319
```

Podemos ver que Matlab no nos muestra los valores que hemos introducido. En los valores mostrados las cifras 298 del número a1 han sido redondeadas a 300 lo cual es un error en la cifra decimocuarta. Las cifras 187 del número a2 han sido redondeadas a 190 lo cual es un error en la cifra decimoquinta.

```
a1
a2
```

```
a1 =

   0.333333333333330


a2 =

   0.333333333333319
```

Si restamos 'a mano' los valores exactos de a1 y a2, el resultado es 1.11e-014, sin embargo Matlab proporciona un valor cuya quinta cifra significativa es erronea. Observe como hemos pasado de errores en la cifra decimocuarta a errores en la quinta cifra.

```
a1-a2
```

```
ans =

     1.110223024625157e-14
```

Veamos ahora otro ejemplo en el que los errores de cancelación se ven afectados por la multiplicación por números 'grandes'. Dado un valor x, consideraremos dos expresiones v1 y v2 que representan el mismo número, pero al calcular la primera se produce un error de cancelación. (Puede comprobar que, para x positivo, v1 y v2 representan al mismo número real si multiplica numerador y denominador de v2 por sqrt(x+1)-sqrt(x) y simplifica.)

```
x=15;
v1=x*(sqrt(x+1)-sqrt(x))
v2=x/(sqrt(x+1)+sqrt(x))
```

```
v1 =
```

```
    1.905249806888745

v2 =

    1.905249806888747
```

Hallamos la diferencia entre v1 y v2. Observe que aunque ambas expresiones representan el mismo número real, el valor numérico de v1 es distinto del de v2.

```
v1-v2
```

```
ans =

    -1.998401444325282e-15
```

Ej 1. Descargue de la web de la asignatura el script cancelacion.m para comprobar el error de cancelación que se produce al evaluar las expresiones v1 y v2. Ejecute el script para los valores x=19, 10^15, 10^20. (Sol. Errores = -3.9968e-015, 2.8151e+006, -5.0000e+009).

Usamos la instrucción format short para que los siguientes resultados se muestren con menos decimales.

```
format short
```

Acumulador (suma)

Ahora vamos a implementar un algoritmo sencillo en el que usamos un acumulador para calcular la suma de N números. (Estas instrucciones se encuentran en el archivo acumulador_suma.m)

```
% Definimos un vector que contiene los números que vamos a sumar
x=[2 4 9 3 6 0 3 2 1 8 6 7 5]
% Hallamos el número de elementos del vector
N=length(x)
% Inicializamos el acumulador
s=0;
% Sumamos al acumulador cada uno de los elementos del vector x
for i=1:N
    s=s+x(i);
end
% Mostramos el resultado
disp('La suma de las componentes es:')
disp(s)
```

```
x =

     2     4     9     3     6     0     3     2     1     8     6     7     5


N =

    13

La suma de las componentes es:
    56
```

Vectorización (suma)

En este caso se puede hallar la suma de las componentes del vector x de una forma más eficiente sin emplear un bucle for, usando la función sum.

```
sum(x)
```

```
ans =

    56
```

Ej 2. Defina un vector llamado y que contenga las cifras de su DNI o pasaporte. Use un acumulador para hallar la suma de las componentes del vector y. Obtenga el mismo resultado usando la función sum.

Acumulador (producto)

De forma similar, implementamos el uso de un acumulador para calcular el producto de N números, y además dejaremos sin multiplicar los que sean nulos.

En este caso, el acumulador se inicializa con el valor 1, y dentro del bucle comprobaremos que las componentes a multiplicar sean distintas de cero.

```
% Definimos un vector que contiene los números que vamos a multiplicar
v=[2 4 9 3 6 0 3 2 1 0 6 7 5]
% Hallamos el número de elementos del vector
N=length(v)
% Inicializamos el acumulador
p=1
% Multiplicamos el acumulador por cada uno de los elementos no nulos del vector v
for i=1:N
    if v(i)~=0
        p=p*v(i);
    end
end
% Mostramos el resultado
disp('El producto de las componentes no nulas es:')
disp(p)
```

```
v =

     2     4     9     3     6     0     3     2     1     0     6     7     5


N =

    13


p =

     1

El producto de las componentes no nulas es:
     1632960
```

Si empleamos el comando prod obtendremos el producto de todos los elementos, incluyendo los nulos.

```
prod(v)
```

```
ans =

     0
```

Si queremos evitar los elementos nulos, debemos usar la función find que nos da los índices (las posiciones) de aquellos elementos no nulos.

Si observamos el vector v tenemos que son nulos los elementos sexto y décimo, y todos los demás son distintos de cero.

```
v
```

```
v =

     2     4     9     3     6     0     3     2     1     0     6     7     5
```

Vectorización (producto)

Con esta instrucción obtenemos un vector con los índices de los elementos no nulos de v. Vemos que aparecen los números del 1 al 13, salvo el 6 y el 10.

```
find(v)
```

```
ans =

     1     2     3     4     5     7     8     9    11    12    13
```

La siguiente instrucción halla los elementos de v no nulos.

```
vd=v(find(v))
```

```
vd =

     2     4     9     3     6     3     2     1     6     7     5
```

Ahora podemos hallar el producto de los elementos no nulos de v.

```
prod(vd)
```

```
ans =

     1632960
```

Si quisiéramos calcular el producto de los elementos no nulos de v con una sola instrucción, podríamos haber empleado la instrucción siguiente.

```
prod(v(find(v)))
```

```
ans =

     1632960
```

La función find también sirve para encontrar los índices de los elementos que cumplan cierta condición. Por ejemplo, la siguiente instrucción nos da los índices de los elementos de v que son mayores estrictamente que 4.

```
find(v>4)
```

```
ans =

     3     5    11    12    13
```

Ej 3. Halle el producto de todas las cifras de su DNI o pasaporte que sean menores o iguales que 5. Haga este ejercicio en primer lugar usando un bucle, y luego sin usar bucles. (Evidentemente, si en las cifras usadas hay algún 0 debe obtener un resultado nulo.)

Ej 4. Defina un vector con los números naturales del 10 al 60. Halle la suma de los elementos que sean mayores o iguales que 35. Haga este ejercicio en primer lugar usando un bucle, y luego sin usar bucles. (El resultado es 1235.)

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
15-Mar-2017
```

Published with MATLAB® R2013a

Práctica 2. Estabilidad. Condicionamiento.

Contents

Estabilidad

Se dice que un algoritmo es estable (numéricamente) cuando es poco sensible a los errores de redondeo.

Para ilustrar el concepto de estabilidad de un algoritmo veremos dos maneras de calcular la sucesión (1/2)^i, una estable y otra que no lo es.

Sucesión (algoritmo estable)

Vemos un ejemplo en el que se hallan por recurrencia los términos de la sucesión s(1)=1/2, s(i)=s(i-1)/2 para i>=2.

A continuación para cada valor de i desde 2 hasta 50 se muestra el propio i, el término de la sucesión s(i), el valor de (1/2)^i y la diferencia entre estos dos últimos números. (Puede encontrar estas instrucciones en el programa sucesion.m)

```
% Sucesión de números (estable)
N=50;
% La siguiente instrucción genera un vector-columna con ceros. En este
% vector guardaremos los términos de la sucesión.
s=zeros(N,1);
s(1)=1/2;
% s(2)=0.5*s(1);
% s(3)=0.5*s(2);
    disp('    i         s(i)     (1/2)^i          s(i)-(1/2)^i  ');
    disp('-------------------------------------------------------');
% Encabezamiento para format short.
% Con format long el encabezamiento queda peor.
for i=2:N
    s(i)=s(i-1)/2;
    disp([i, s(i), (1/2)^i, s(i)-(1/2)^i]);
end
```

```
    i         s(i)     (1/2)^i          s(i)-(1/2)^i
-------------------------------------------------------
    2.0000    0.2500    0.2500         0

    3.0000    0.1250    0.1250         0

    4.0000    0.0625    0.0625         0

    5.0000    0.0313    0.0313         0

    6.0000    0.0156    0.0156         0

    7.0000    0.0078    0.0078         0

    8.0000    0.0039    0.0039         0

    9.0000    0.0020    0.0020         0

   10.0000    0.0010    0.0010         0

   11.0000    0.0005    0.0005         0

   12.0000    0.0002    0.0002         0

   13.0000    0.0001    0.0001         0

   14.0000    0.0001    0.0001         0
```

```
15.0000    0.0000    0.0000         0
16.0000    0.0000    0.0000         0
17.0000    0.0000    0.0000         0
18.0000    0.0000    0.0000         0
19.0000    0.0000    0.0000         0
20.0000    0.0000    0.0000         0
21.0000    0.0000    0.0000         0
22.0000    0.0000    0.0000         0
23.0000    0.0000    0.0000         0
24.0000    0.0000    0.0000         0
25.0000    0.0000    0.0000         0
26.0000    0.0000    0.0000         0
27.0000    0.0000    0.0000         0
28.0000    0.0000    0.0000         0
29.0000    0.0000    0.0000         0
30.0000    0.0000    0.0000         0
31.0000    0.0000    0.0000         0
32.0000    0.0000    0.0000         0
33.0000    0.0000    0.0000         0
34.0000    0.0000    0.0000         0
35.0000    0.0000    0.0000         0
36.0000    0.0000    0.0000         0
37.0000    0.0000    0.0000         0
38.0000    0.0000    0.0000         0
39.0000    0.0000    0.0000         0
40.0000    0.0000    0.0000         0
41.0000    0.0000    0.0000         0
42.0000    0.0000    0.0000         0
43.0000    0.0000    0.0000         0
44.0000    0.0000    0.0000         0
45.0000    0.0000    0.0000         0
46.0000    0.0000    0.0000         0
47.0000    0.0000    0.0000         0
48.0000    0.0000    0.0000         0
49.0000    0.0000    0.0000         0
50.0000    0.0000    0.0000         0
```

Ej 1. Ejecute el script sucesion.m para N=1700 usando antes la instrucción format long. (Si al acabar el ejercicio no aparece la opción 'Next' para continuar con la práctica, puede volver a ejecutarla con 'echodemo practica2').

Sucesión (algoritmo inestable)

En este ejemplo emplearemos la siguiente sucesión s(1)=1/2, s(2)=1/4, s(i)=(23/2)*s(i-1)-11/2*s(i-2) para i>=3. Puede probarse (por inducción) que para todo valor de i

natural se tiene que s(i)=(1/2)^i, aunque veremos que numéricamente se producen errores.

Las siguientes instrucciones (que puede encontrar en el programa sucesion2.m) hallan los primeros términos de esta sucesión.

```
% Sucesión de números (inestable)
format compact % Elimina lineas en blanco en el resultado
N=1700;
s=zeros(N,1);
s(1)=1/2;
s(2)=1/4;
    disp('    i         s(i)     (1/2)^i          s(i)-(1/2)^i  ');
    disp('------------------------------------------------------------');
% Encabezamiento para format short.
% Con format long el encabezamiento queda peor.
for i=3:N
    s(i)=(23/2)*s(i-1)-11/2*s(i-2);
    disp([i, s(i), (1/2)^i, s(i)-(1/2)^i]);
end
format loose
% Volvemos al formato usual en el que se incluyen más lineas en blanco.
```

```
    i         s(i)     (1/2)^i          s(i)-(1/2)^i
------------------------------------------------------------
    3.0000    0.1250    0.1250         0
    4.0000    0.0625    0.0625         0
    5.0000    0.0313    0.0313         0
    6.0000    0.0156    0.0156         0
    7.0000    0.0078    0.0078         0
    8.0000    0.0039    0.0039         0
    9.0000    0.0020    0.0020         0
   10.0000    0.0010    0.0010         0
   11.0000    0.0005    0.0005         0
   12.0000    0.0002    0.0002         0
   13.0000    0.0001    0.0001         0
   14.0000    0.0001    0.0001         0
   15.0000    0.0000    0.0000         0
   16.0000    0.0000    0.0000         0
   17.0000    0.0000    0.0000         0
   18.0000    0.0000    0.0000         0
   19.0000    0.0000    0.0000         0
   20.0000    0.0000    0.0000         0
   21.0000    0.0000    0.0000         0
   22.0000    0.0000    0.0000         0
   23.0000    0.0000    0.0000         0
   24.0000    0.0000    0.0000         0
   25.0000    0.0000    0.0000         0
   26.0000    0.0000    0.0000         0
   27.0000    0.0000    0.0000         0
   28.0000    0.0000    0.0000         0
   29.0000    0.0000    0.0000         0
   30.0000    0.0000    0.0000         0
   31.0000    0.0000    0.0000         0
   32.0000    0.0000    0.0000         0
   33.0000    0.0000    0.0000         0
   34.0000    0.0000    0.0000         0
   35.0000    0.0000    0.0000         0
   36.0000    0.0000    0.0000         0
   37.0000    0.0000    0.0000         0
   38.0000    0.0000    0.0000         0
   39.0000    0.0000    0.0000         0
   40.0000    0.0000    0.0000         0
   41.0000    0.0000    0.0000         0
   42.0000    0.0000    0.0000         0
   43.0000    0.0000    0.0000         0
   44.0000    0.0000    0.0000         0
   45.0000    0.0000    0.0000         0
   46.0000    0.0000    0.0000         0
   47.0000    0.0000    0.0000         0
   48.0000    0.0000    0.0000         0
   49.0000    0.0000    0.0000         0
   50.0000    0.0000    0.0000         0
   51.0000    0.0000    0.0000         0
   52.0000    0.0000    0.0000         0
   53.0000    0.0000    0.0000         0
   54.0000    0.0000    0.0000         0
   55.0000    0.0000    0.0000         0
   56.0000    0.0000    0.0000         0
```

57.0000	0.0000	0.0000	0
58.0000	0.0000	0.0000	0
59.0000	0.0000	0.0000	0
60.0000	0.0000	0.0000	0
61.0000	0.0000	0.0000	0
62.0000	0.0000	0.0000	0
63.0000	0.0000	0.0000	0
64.0000	0.0000	0.0000	0
65.0000	0.0000	0.0000	0
66.0000	0.0000	0.0000	0
67.0000	0.0000	0.0000	0
68.0000	0.0000	0.0000	0
69.0000	0.0000	0.0000	0
70.0000	0.0000	0.0000	0
71.0000	0.0000	0.0000	0
72.0000	0.0000	0.0000	0
73.0000	0.0000	0.0000	0
74.0000	0.0000	0.0000	0
75.0000	0.0000	0.0000	0
76.0000	0.0000	0.0000	0
77.0000	0.0000	0.0000	0
78.0000	0.0000	0.0000	0
79.0000	0.0000	0.0000	0
80.0000	0.0000	0.0000	0
81.0000	0.0000	0.0000	0
82.0000	0.0000	0.0000	0
83.0000	0.0000	0.0000	0
84.0000	0.0000	0.0000	0
85.0000	0.0000	0.0000	0
86.0000	0.0000	0.0000	0
87.0000	0.0000	0.0000	0
88.0000	0.0000	0.0000	0
89.0000	0.0000	0.0000	0
90.0000	0.0000	0.0000	0
91.0000	0.0000	0.0000	0
92.0000	0.0000	0.0000	0
93.0000	0.0000	0.0000	0
94.0000	0.0000	0.0000	0
95.0000	0.0000	0.0000	0
96.0000	0.0000	0.0000	0
97.0000	0.0000	0.0000	0
98.0000	0.0000	0.0000	0
99.0000	0.0000	0.0000	0
100.0000	0.0000	0.0000	0
101.0000	0.0000	0.0000	0
102.0000	0.0000	0.0000	0
103.0000	0.0000	0.0000	0
104.0000	0.0000	0.0000	0
105.0000	0.0000	0.0000	0
106.0000	0.0000	0.0000	0
107.0000	0.0000	0.0000	0
108.0000	0.0000	0.0000	0
109.0000	0.0000	0.0000	0
110.0000	0.0000	0.0000	0
111.0000	0.0000	0.0000	0
112.0000	0.0000	0.0000	0
113.0000	0.0000	0.0000	0
114.0000	0.0000	0.0000	0
115.0000	0.0000	0.0000	0
116.0000	0.0000	0.0000	0
117.0000	0.0000	0.0000	0
118.0000	0.0000	0.0000	0
119.0000	0.0000	0.0000	0
120.0000	0.0000	0.0000	0
121.0000	0.0000	0.0000	0
122.0000	0.0000	0.0000	0
123.0000	0.0000	0.0000	0
124.0000	0.0000	0.0000	0
125.0000	0.0000	0.0000	0
126.0000	0.0000	0.0000	0
127.0000	0.0000	0.0000	0
128.0000	0.0000	0.0000	0
129.0000	0.0000	0.0000	0
130.0000	0.0000	0.0000	0
131.0000	0.0000	0.0000	0
132.0000	0.0000	0.0000	0
133.0000	0.0000	0.0000	0
134.0000	0.0000	0.0000	0

(...)

```
  0.0000    2.9603         0    2.9603
1.0e+232 *
  0.0000    3.2563         0    3.2563
1.0e+233 *
  0.0000    3.5819         0    3.5819
1.0e+234 *
  0.0000    3.9401         0    3.9401
1.0e+235 *
  0.0000    4.3341         0    4.3341
1.0e+236 *
  0.0000    4.7675         0    4.7675
1.0e+237 *
  0.0000    5.2443         0    5.2443
1.0e+238 *
  0.0000    5.7687         0    5.7687
1.0e+239 *
  0.0000    6.3456         0    6.3456
1.0e+240 *
  0.0000    6.9801         0    6.9801
1.0e+241 *
  0.0000    7.6781         0    7.6781
1.0e+242 *
  0.0000    8.4459         0    8.4459
1.0e+243 *
  0.0000    9.2905         0    9.2905
1.0e+245 *
  0.0000    1.0220         0    1.0220
1.0e+246 *
  0.0000    1.1242         0    1.1242
1.0e+247 *
  0.0000    1.2366         0    1.2366
1.0e+248 *
  0.0000    1.3602         0    1.3602
1.0e+249 *
  0.0000    1.4963         0    1.4963
1.0e+250 *
  0.0000    1.6459         0    1.6459
1.0e+251 *
  0.0000    1.8105         0    1.8105
1.0e+252 *
  0.0000    1.9915         0    1.9915
1.0e+253 *
  0.0000    2.1907         0    2.1907
1.0e+254 *
  0.0000    2.4097         0    2.4097
1.0e+255 *
  0.0000    2.6507         0    2.6507
1.0e+256 *
  0.0000    2.9158         0    2.9158
1.0e+257 *
  0.0000    3.2073         0    3.2073
1.0e+258 *
  0.0000    3.5281         0    3.5281
1.0e+259 *
  0.0000    3.8809         0    3.8809
1.0e+260 *
  0.0000    4.2690         0    4.2690
1.0e+261 *
  0.0000    4.6959         0    4.6959
1.0e+262 *
  0.0000    5.1655         0    5.1655
1.0e+263 *
  0.0000    5.6820         0    5.6820
1.0e+264 *
  0.0000    6.2502         0    6.2502
1.0e+265 *
  0.0000    6.8752         0    6.8752
1.0e+266 *
  0.0000    7.5628         0    7.5628
1.0e+267 *
  0.0000    8.3190         0    8.3190
1.0e+268 *
  0.0000    9.1509         0    9.1509
1.0e+270 *
  0.0000    1.0066         0    1.0066
1.0e+271 *
  0.0000    1.1073         0    1.1073
1.0e+272 *
  0.0000    1.2180         0    1.2180
1.0e+273 *
```

```
  0.0000    1.3398         0    1.3398
1.0e+274 *
  0.0000    1.4738         0    1.4738
1.0e+275 *
  0.0000    1.6211         0    1.6211
1.0e+276 *
  0.0000    1.7833         0    1.7833
1.0e+277 *
  0.0000    1.9616         0    1.9616
1.0e+278 *
  0.0000    2.1577         0    2.1577
1.0e+279 *
  0.0000    2.3735         0    2.3735
1.0e+280 *
  0.0000    2.6109         0    2.6109
1.0e+281 *
  0.0000    2.8720         0    2.8720
1.0e+282 *
  0.0000    3.1592         0    3.1592
1.0e+283 *
  0.0000    3.4751         0    3.4751
1.0e+284 *
  0.0000    3.8226         0    3.8226
1.0e+285 *
  0.0000    4.2048         0    4.2048
1.0e+286 *
  0.0000    4.6253         0    4.6253
1.0e+287 *
  0.0000    5.0878         0    5.0878
1.0e+288 *
  0.0000    5.5966         0    5.5966
1.0e+289 *
  0.0000    6.1563         0    6.1563
1.0e+290 *
  0.0000    6.7719         0    6.7719
1.0e+291 *
  0.0000    7.4491         0    7.4491
1.0e+292 *
  0.0000    8.1940         0    8.1940
1.0e+293 *
  0.0000    9.0134         0    9.0134
1.0e+294 *
  0.0000    9.9148         0    9.9148
1.0e+296 *
  0.0000    1.0906         0    1.0906
1.0e+297 *
  0.0000    1.1997         0    1.1997
1.0e+298 *
  0.0000    1.3197         0    1.3197
1.0e+299 *
  0.0000    1.4516         0    1.4516
1.0e+300 *
  0.0000    1.5968         0    1.5968
1.0e+301 *
  0.0000    1.7565         0    1.7565
1.0e+302 *
  0.0000    1.9321         0    1.9321
1.0e+303 *
  0.0000    2.1253         0    2.1253
1.0e+304 *
  0.0000    2.3379         0    2.3379
1.0e+305 *
  0.0000    2.5716         0    2.5716
1.0e+306 *
  0.0000    2.8288         0    2.8288
1.0e+307 *
  0.0000    3.1117         0    3.1117
     1682         Inf          0         Inf
     1683         Inf          0         Inf
     1684         NaN          0         NaN
     1685         NaN          0         NaN
     1686         NaN          0         NaN
     1687         NaN          0         NaN
     1688         NaN          0         NaN
     1689         NaN          0         NaN
     1690         NaN          0         NaN
     1691         NaN          0         NaN
     1692         NaN          0         NaN
     1693         NaN          0         NaN
     1694         NaN          0         NaN
```

```
        1695         NaN           0         NaN
        1696         NaN           0         NaN
        1697         NaN           0         NaN
        1698         NaN           0         NaN
        1699         NaN           0         NaN
        1700         NaN           0         NaN
```

Inf y NaN

En la salida del programa han aparecido los resultados Inf y NaN.

Inf es la forma en que Matlab representa el infinito, y se produce al obtener resultados muy grandes que desbordan la capacidad de representación, y también al efectuar operaciones como dividir un número positivo entre 0.

En este ejemplo se tiene que s(1681)=3.1117e+307 y al multiplicar este valor por 23/2 se obtiene un valor que no se puede representar, y por eso aparece s(1682)=Inf.

NaN significa Not-a-Number y se obtiene al efectuar operaciones que no están definidas matemáticamente, en este caso al restar infinito menos infinito al calcular s(1684).

Ej 2. Tras ejecutar el script sucesion2.m halle el error absoluto que se comete al considerar s(1300) como aproximación de (1/2)^1300. Idem para s(1390). Ejecute de nuevo el script sucesion.m y halle los errores que se cometen en los mismos términos de la sucesión. (Sol. Usando sucesion2.m el error absoluto para s(1300) es 5.8046e-090 y para s(1390) es 3.0840e+004 . Usando sucesion.m el error absoluto para s(1300) es 0 y para s(1390) es 0.)

Ej 3. Escriba un programa sucesion3.m que permita hallar los términos de la sucesión dada por s(1)=1/3, s(2)=1/9, s(i)=(10/3)*s(i-1)-s(i-2) para i>=3, y halle el error absoluto y relativo que se comete en los términos s(15) y s(45) sabiendo que operando de forma exacta se tiene que s(i)=(1/3)^i.

El programa debe producir un resultado como el siguiente:

Error absoluto al considerar s(15): -1.8392e-013

Error relativo al considerar s(15): -2.6390e-006

Error absoluto al considerar s(45): -37.8665

Error relativo al considerar s(45): -1.1187e+023

(Observe cómo este método para calcular la sucesión es inestable)

Condicionamiento

Un problema se dice mal condicionado si una pequeña variación en los datos del problema causa grandes variaciones en su solución. En estos problemas, los errores de redondeo pueden provocar grandes variaciones en los resultados.

Por ejemplo, el problema de resolver un sistema de ecuaciones lineales compatible determinado puede ser, a veces, un problema mal condicionado. Así, veremos cómo afecta a la solución de un cierto sistema de ecuaciones lineales un pequeño cambio en el vector de términos independientes.

Estudiaremos primero la manera de plantear un sistema de ecuaciones lineales en forma matricial y luego veremos ejemplos de sistemas de ecuaciones lineales mal condicionados.

Forma matricial de un sistema de ecuaciones lineales.

Consideramos el siguiente sistema de dos ecuaciones lineales y dos incógnitas

2x+3y= 8,

x- y= -1.

(Este ejemplo se encuentra en el script sistema.m)

Para plantearlo en la forma A*x=b, comenzamos definiendo la matriz de coeficientes del sistema

```
A=[2 3;
   1 -1]
% Definimos ahora el vector de términos independientes del sistema
b=[8;
  -1]
% Como ejemplo, definiremos dos vectores columna x1, x2 y comprobaremos si
% son solución del sistema.
% Definimos el vector columna x1
x1=[1;
   2]
% Comprobamos si x1 es solución del sistema. (Como se obtiene una
% columna de ceros, se cumplen todas las ecuaciones del sistema)
A*x1-b
```

```
A =

     2     3
     1    -1


b =

     8
    -1


x1 =

     1
     2


ans =

     0
     0
```

Definimos el vector columna x2

```
x2=[2 ; 1] ;
% Comprobamos si x2 es solución del sistema
A*x2-b
```

```
ans =

    -1
     2
```

Debemos notar que al plantear un sistema de ecuaciones lineales usando matrices, el producto de la matriz de coeficientes del sistema por el vector de incógnitas se efectúa en Matlab con un asterisco A*x1 mientras que por escrito se denota con un punto A·x1.

Ej 4. Defina A la matriz de coeficientes del sistema

x+2y+ 3z=-6,

3x -z= 6,

  y+(5/3)z=-4,

así como b el vector de términos independientes del sistema.

Compruebe si los siguientes valores de (x, y , z) son solución del sistema a) x=1, y= 1, z=-3. b) x=2, y= 0, z= 3. c) x=2, y=-4, z= 0. (Solución: Los valores de los apartados a y c son solución, los del b no lo son)

Solución de un sistema de ecuaciones lineales.

Supongamos que tenemos un sistema planteado en la forma A*x=b. La 'división izquierda' A\b nos da la solución del sistema cuando A es una matriz regular. (En la práctica 9 usaremos la división izquierda con A una matriz no cuadrada y A*x=b incompatible; en tal caso, el vector que se obtiene al hacer A\b se dice que es la solución 'en el sentido de los mínimos cuadrados' del sistema A*x=b. De hecho, veremos que ese vector es el de los coeficientes del polinomio de un cierto grado (recta, parábola,...) que ajusta por mínimos cuadrados un conjunto de puntos del plano.) (Como regla mnemotécnica, en esta 'división izquierda' la matriz está en la izquierda, y siguiendo la barra en sentido ascendente, apunta hacia la izquierda. La tecla de la 'división izquierda' está a la izquierda del teclado.)

Hallamos la solución del sistema de ecuaciones lineales -x + y = 1, x + y = 3. (Este ejemplo se encuentra en el script solsistema.m)

```
% Definimos la matriz de coeficientes del sistema
A=[-1 1;
    1 1];
% Definimos el vector de términos independientes del sistema
b=[1;
   3];
% Como el sistema tiene el mismo número de ecuaciones que de incógnitas, la
% matriz A de coeficientes del sistema es cuadrada. Hallamos el determinante
% de A para ver si A es regular y, por tanto, comprobar si el sistema tiene
```

```
% una única solución.
det(A)
% Usamos la 'división izquierda' para obtener la solución del sistema.
x=A\b
% Comprobamos que el vector x obtenido es solución del sistema.
A*x-b
```

```
ans =

    -2


x =

     1
     2


ans =

     0
     0
```

Ej 5. Halle la solución del siguiente sistema de ecuaciones lineales x+2y+3z= -2, 4x+5y+6z= -2, -x+ y+ z= -2. Halle el determinante de la matriz de coeficientes del sistema. (Sol. x=1, y=0, z=-1, determinante=6).

Ejemplo de sistema mal condicionado

Consideramos el siguiente sistema de 2 ecuaciones lineales y 2 incógnitas.

2.3x+1.2y=0.99,

4.4x+2.3y=1.89.

```
% Resolvemos el sistema
A=[2.3 1.2 ; 4.4 2.3];
b=[0.99; 1.89];
x=A\b
```

```
x =

    0.9000
   -0.9000
```

Veamos de qué manera un pequeño cambio en el vector de términos independientes afecta a la solución de este sistema.

```
% Consideramos el sistema
% 2.3x+1.2y=1,
% 4.4x+2.3y=1.9.

% Resolvemos el sistema (la matriz de coeficientes es la misma)
b1=[1; 1.9];
x1=A\b1

% Vemos que un pequeño cambio en el vector de términos independientes resulta
% en un cambio mayor en la solución.
```

```
x1 =

    2.0000
   -3.0000
```

Este último sistema se dice que está 'mal condicionado'. En la siguiente sección de esta práctica se estudiará la norma de una matriz y el significado del número de condición de una matriz, y aprenderemos a calcularlos con Matlab.

Ej 6. Halle la solución del siguiente sistema de ecuaciones lineales

```
      x+      2y+3z=13,

     2x+       y   = 8,

(7/10)x+(29/10)y+5z=18.
```

(Sol. x=2, y=4, z=1).

Ej 7. Considere el siguiente sistema de ecuaciones lineales

```
      x+      2y+3z=13,

     2x+       y   = 8,

(8/10)x+(29/10)y+5z=18.
```

a) Defina la matriz de coeficientes y el vector de términos independientes del sistema. b) Halle x0=A\b. (Compruebe como una variación de una décima en un coeficiente, respecto a la matriz del ejericio anterior, da lugar a una 'solución' muy diferente) c) Halle el determinante de la matriz A. (Sol. -2.2204e-015, aunque el valor exacto del determinante es 0, por lo que el sistema no tiene solución.) d) Halle A*x0-b. ¿Representa el vector x0 una solución del sistema?

Condicionamiento de una matriz

El condicionamiento (o número de condición) de una matriz cuadrada es igual a la norma de la matriz por la norma de su inversa. Aunque se pueden definir diferentes normas para una matriz, los números de condición para cada una de ellas no varían mucho de unas a otras. Cuando para alguna norma ocurre que el número de condición de la matriz está muy próximo a 1 se dice que la matriz está 'bien condicionada'; en cambio, cuando para alguna norma ocurre que el número de condición de la matriz es mucho mayor que uno entonces se dice que la matriz está 'mal condicionada'.

La función norm de Matlab permite calcular cuatro normas diferentes de una matriz y, por tanto, con MatLab podemos hallar cuatro números de condición para una matriz (uno por cada una de las cuatro normas).

```
norm(A,2)
norm(A,1)
norm(A,inf)
norm(A,'fro')
```

```
ans =

    5.6018


ans =

    6.7000


ans =

    6.7000


ans =

    5.6018
```

Por ejemplo si usamos la norma 1, podemos calcular el número de condición de la matriz A para esa norma con estas instrucciones

```
n=norm(A,1)
ni=norm(inv(A),1)
ncondicion=n*ni

% Observamos que el número de condición obtenido es mucho mayor que uno y,
% por tanto, la matriz A está mal condicionada.
```

```
n =

    6.7000


ni =

  670.0000
```

```
ncondicion =

   4.4890e+03
```

También podría haberse hallado el condicionamiento para la norma 1 con la función cond.

```
cond(A,1)
```

```
ans =

   4.4890e+03
```

Ej 8. Considere la matriz A=[1 3 -1; 0 2 -2; 3 2 4]. a) Halle el número de condición de la matriz A usando la norma 1. b) Halle el número de condición de la matriz A usando la norma 2. (Sol. 6.3050e+016, 4.1041e+016 multiplicando las normas.)

Observe que en ambos casos antes de calcular la inversa el propio Matlab nos avisa de que los resultados (los elementos de la inversa) pueden ser poco precisos. Además nos muestra (sin pedirlo) el valor de rcond(A) que es una estimación de 1/cond(A,1).

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
15-Mar-2017
```

Published with MATLAB® R2013a

Práctica 3. Resolución de ecuaciones (I)

Muchos problemas de las ciencias requieren hallar una solución de una ecuación, o equivalentemente, una raíz (o cero) de una función. Por ejemplo, para hallar el volumen en la ecuación de Van der Waals, hay que hallar una raíz de una función polinómica de tercer grado.

Así pues, dada una función f, nos plantearemos encontrar valores x que hagan f(x)=0. En esta práctica y en la siguiente veremos métodos para hallar con Matlab las raíces de una función. En concreto, en esta práctica trataremos los métodos de bisección y de regula falsi.

Contents

Antes de estudiar estos métodos, veremos dos maneras de definir una función, mediante un archivo .m o como una función anónima y también veremos maneras de representar conjuntamente la gráfica de una función y el eje OX.

Funciones en archivos .m

Pueden definirse funciones en Matlab mediante archivos .m cuya primera linea comienza con `function`. Lo que aparece en la siguiente linea serviría para definir una función llamada `nombrefuncion` con dos argumentos:

```
function valorsalida=nombrefuncion (argumento1,argumento2)
```

dentro del código del archivo se debe dar a `valorsalida` el valor que toma la función. Dentro del archivo de la función deben incluirse sólo las instrucciones necesarias para calcular el valor que toma la función, todas las instrucciones que usen la función deben ir fuera del archivo.

Una función puede tener varias salidas:

```
function [salida1 salida2]=nombrefuncion (argumento1,argumento2)
```

Observe la función f.m, que tiene sólo una entrada (la variable x) y sólo una salida (el valor de f). El comando `type f.m` muestra el contenido del archivo f.m . Tenga en cuenta que la función se usa mediante el nombre del archivo y no mediante el `nombrefuncion` que se da en la primera linea.

```
type f.m
```

```
function f=f(x)
% Ejemplo de función f.
% Al emplear .^ la función puede aplicarse a vectores
f=1/2-x.^2;
```

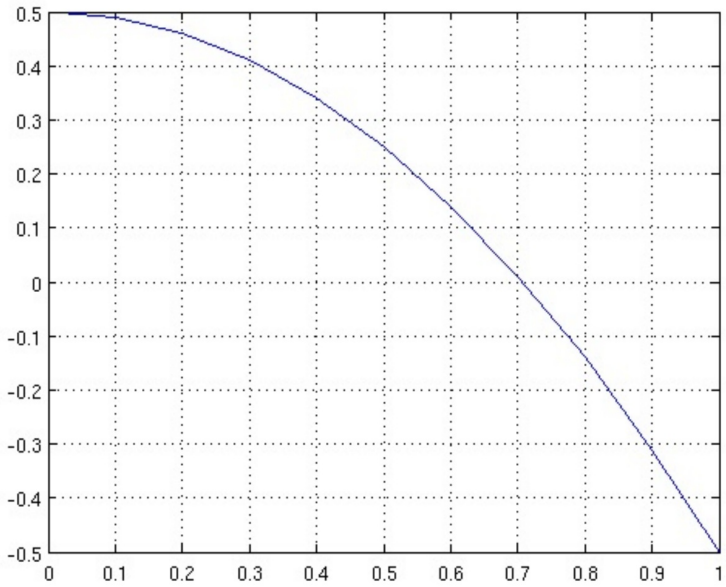
Ej 1. Defina un vector x con valores desde -1 a 1 con paso 0.1. Compruebe que la función f.m puede aplicarse al vector x. (Sol. f(x)=[-0.5, -0.31, -0.14, 0.01, ...])

Ej 2. Represente la función f en el intervalo [-1,1].

Representación del eje OX

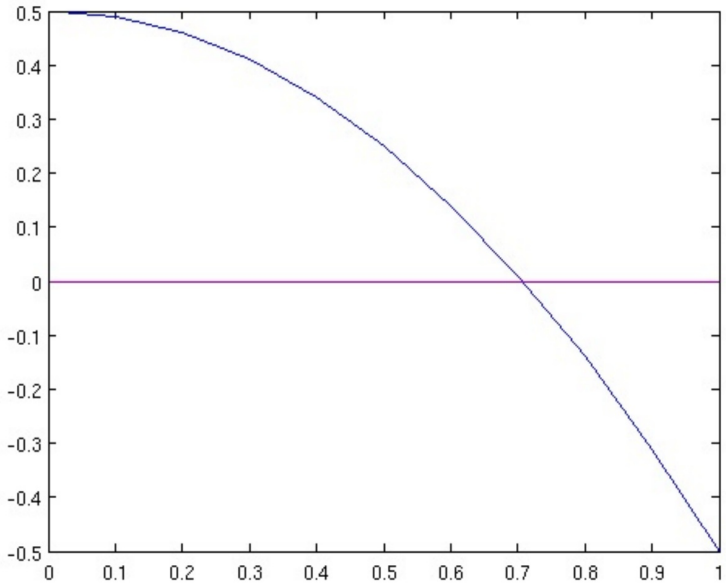
Una forma de representar el eje OX es utilizar, después del comando plot, el comando grid on, que representa una malla o cuadrícula en la gráfica de la función.

```
x=0:0.1:1;
plot(x,f(x))
grid on
```



Si en vez de la malla sólo queremos representar el eje OX, podemos dibujar la función y=0.

```
x=0:0.1:1;
y1=f(x);
y2=zeros(length(x));
plot(x,y1,x,y2)
```



Ej 3. Defina en un script g.m la función g(x)=3/4-1/(1+x^2). Evalúe g(2). Represente la función junto con el eje horizontal en el intervalo [-5,5]. (Sol. 0.55)

Funciones anónimas

Es posible en Matlab definir funciones sin usar archivos .m, por ejemplo la siguiente linea define la función h(x)=x^2+1

```
h=@(x) x.^2+1
```

```
h =

    @(x)x.^2+1
```

La sintaxis para definir una función anónima es la siguiente

`nombrefuncion=@(argumentos) expresion`

Por ejemplo, la siguiente linea define la función n2(a,b)=sqrt(a^2+b^2)

```
n2=@(a,b) sqrt(a.^2+b.^2)
```

```
n2 =

    @(a,b)sqrt(a.^2+b.^2)
```

Ej 4. Sin emplear un archivo .m, defina la función h2(x)=sin(x)^2-cos(x)^2. Halle el valor de h2(0) y de h2(2). (Solución: h2(0)=-1, h2(2)=0.6536)

Localización gráfica de raíces.

En ocasiones podemos usar la gráfica de una función para encontrar un punto cercano a una raíz o un intervalo que contenga a una raíz.

Ej 5. Defina en un archivo g.m la función g(x)=exp(x)-5x. a) Dibuje la gráfica de g en el intervalo [0,2] junto con el eje OX. b) Represente de nuevo la función en un intervalo más pequeño, de longitud 0.5 que contenga una raiz de g.

Ej 6. Defina de forma anónima las funciones h1(x)=e^(2x) y h2(x)=2-x^2. a) Represente conjuntamente ambas gráficas en el intervalo [-2,1]. b) Represente la función h1(x)-h2(x) en el intervalo [-2,1]. Observe que las raíces de h1(x)-h2(x) son los valores de x en los que h1(x)=h2(x), esto es, las abscisas de los puntos del plano donde se cortan las gráficas de h1 y h2.

Bisección

Recordamos el pseudocódigo que permite implementar el método de bisección

```
1. Entrar f , a, b, tol
2. Mientras b-a >= tol
3. Hacer c = (a+b)/2
4. Si f (c) = 0 entonces c es raíz. Fin
5. Si sgn f (a) ~= sgn f (c) entonces b = c.
6. Si sgn f (a) = sgn f (c) entonces a = c.
7. Ir a 2
```

Siguiendo estos pasos se ha implementado el método en el archivo biseccion.m

```
type biseccion.m
```

```
% Método de bisección para una función f definida.
a=0.6;
b=0.7;
tol=0.0001;
while b-a>=tol
    % disp([a b]);
    c=(a+b)/2;
    if f(c)==0
        break
    end
    if sign(f(a))~=sign(f(c))
        b=c;
    else
        a=c;
    end
end
% disp(c)
```

En la implementación se ha empleado la instrucción `break` que interrupe el bucle en que se encuentre, poniendo fin al mismo.

El código escrito tiene algunas limitaciones. Observe que para cambiar los valores iniciales de a y b debe editar el código, y que la función debe llamarse f.

Ej 7. Halle aproximaciones a las dos soluciones de la ecuación sin(x)-exp(x^2)+9/10=0. (Sol. aprox. x=0.1131, x=0.6318)

Ej 8. El código del programa contiene dos comentarios que evitan que se muestre el progreso del método y el resultado final. Quite los símbolos de comentario.

Regula falsi

El método de regula falsi es similar al método de bisección, pero los intervalos no se dividen por la mitad. En este método para cada intervalo, se halla la recta que

pasa por los puntos de la gráfica cuyas abscisas son los extremos del intervalo. El punto de corte de esa recta con el eje OX es la correspondiente iteración del método.

Recordamos el pseudocódigo del método de regula falsi

```
1. Entrar f , a, b, tol
2. Mientras b-a>=tol
3. Hacer c = b-(f(b)*(b-a))/(f(b)-f(a))
4. Si f (c) = 0 entonces c es raíz. Fin
5. Si sgn f (a) ~= sgn f (c) entonces b = c.
6. Si sgn f (a) = sgn f (c) entonces a = c.
7. Ir a 2
```

Ej 9. Modifique el archivo biseccion.m para implementar el método de regula falsi en un programa llamado regula.m.

Ej 10. En el código del método de regula falsi, incluya comentarios que expliquen cada uno de los pasos del método.

Ej 11. Emplee el método de regula falsi en el intervalo [3,4] para encontrar una aproximación a una raíz de f(x)=cos(x/2). (Sol. aprox. x=3.1416)

Ej 12. ¿Puede emplearse alguno de los métodos vistos hasta ahora para encontrar una raíz de f(x)=exp(x)-5*x en el intervalo [1,2]?

Ej 13. Modifique alguno de los métodos para que el criterio de salida del bucle sea abs(f(c))<tol. (Indicación: defina una variable llamada fin con el valor cero. El bucle debe ejecutarse mientras fin sea cero. Dentro del bucle la variable fin tomará el valor 1 cuando abs(f(c))<tol.)

(Siguiendo esta técnica, pueden implementarse distintos criterios para salir del bucle, e incluso varios criterios en un mismo programa.)

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
08-Mar-2017
```

Published with MATLAB® R2013a

Práctica 4. Resolución de ecuaciones (II)

Esta práctica tratará el método de la secante, el método de Newton-Raphson y los métodos de iteración funcional.

Contents

Método de la secante

El método de la secante busca aproximaciones a una raíz de una función de forma iterativa. Dadas dos iteraciones, se halla la recta que pasa por los puntos de la gráfica de la función cuyas abscisas son esas dos iteraciones. La siguiente iteración es el punto de corte de dicha recta con el eje OX.

Recordamos el pseudocódigo que permite implementar el método de la secante:

```
1. Entrar f , a, b, tol
2. Mientras |b - a| >= tol
3. Hacer c =b-f(b)*(b-a)/(f(b)-f(a))
4. Si f (c) = 0 entonces c es raíz. Fin
5. Hacer a = b y b = c.
6. Ir a 2
```

```
% Implementación del método de la secante para una función f definida.
a=1;
b=2.618;
tol=0.0001;
while abs(b-a)>=tol
    c=b-f(b)*(b-a)/(f(b) - f(a));
    % fprintf('a= %d, b=%d, c=%d, f(c)=%d\n',a,b,c,f(c))
    if f(c)==0
        disp('Encontrada raíz.')
        break
    end
    a=b;
    b=c;
end
```

Podemos observar la salida del algoritmo mostrando c y f(c).

```
c
f(c)
```

```
c =

    0.4429


ans =

   8.8148e-09
```

Ej 1. Dibuje la función f(x)=x-5*log(x) en el intervalo [0,2]. Encuentre una raíz de f mediante el método de la secante, determinando antes unos valores apropiados para a y b que definan un intervalo de longitud 0.2. (Sol. Por ejemplo con a=1.2, b=1.4 se llega a la aproximación 1.295856)

Ej 2. Aplique el método de la secante para a=1, b=4 y f(x)=(x-2).*(x-7).*(x-2) (Sol. aprox. 1.999989, solución exacta x=2).

Ej 3. ¿Puede aplicarse el método de la secante para a=1, b=3 y f=(x-2).^2 ?

Ej 4. Aplique el método de la secante para a=1, b=(3+sqrt(5))/2 y f(x)=-4*x^3+ 3*(3+sqrt(5))*x^2-3*(1+sqrt(5))*x-3-sqrt(5). (Sol. aprox x=1.309017)

El comando fprintf

Para mostrar en pantalla resultados con texto y valores numéricos podemos usar el comando fprintf.

```
fprintf('La raíz cuadrada de dos vale %d aproximadamente.\n',sqrt(2))
```

```
La raíz cuadrada de dos vale 1.414214e+00 aproximadamente.
```

Vemos que en el ejemplo hemos puesto una cadena de caracteres que indica el formato en el que vamos a presentar el resultado, y después de la coma el valor que vamos a mostrar.

Al mostrar el resultado, donde pone %d se sustituye el valor de sqrt(2) en formato decimal.

En el siguiente ejemplo se observa como al no poner \n el siguiente fprintf muestra la salida a continuación de la anterior. Para pasar a la siguiente línea empleamos \n. También vemos que al poner %f se sustituye el correspondiente valor en formato de punto fijo.

```
fprintf('Mostramos tres números.')
fprintf('Uno es %d mientras que otro es %f y el tercero es %f',sqrt(2),sqrt(3),sqrt(4))
fprintf('Sin hacer cambio de linea')
```

```
Mostramos tres números.Uno es 1.414214e+00 mientras que otro es 1.732051 y el tercero es 2.000000Sin hacer cambio de linea
```

Ej 5. En el código de la secante, elimine el símbolo de comentario antes de fprintf para poder ver el proceso del método. Modifique el método de la secante para que se vayan contando las iteraciones efectuadas y se muestren en la instrucción fprintf.

Método de Newton-Raphson

Recordamos el pseudocódigo del método

```
1. Entrar f , x0, tol
2. Hacer x1 = x0 - f(x0)/f'(x0)
3. Si |x0 - x1| < tol entonces escribir x1. Fin
4. En caso contrario hacer x0 = x1 e ir a 2.
```

Observamos que para aplicar este método necesitamos evaluar la derivada de la función f. En la implementación newton.m utilizaremos una función fprima.m que permite evaluar la derivada de la función f.m. (Modifique el archivo f.m para que contenga la función e^x+x, y compruebe que el archivo fprima contiene la derivada de f)

Aunque en el pseudocódigo se incluye un salto a la linea 2, en la implementación hay un bucle que se repetirá mientras la variable `seguir` valga `true`. Dicha variable valdrá `false` cuando dos iteraciones sucesivas estén lo suficientemente próximas, lo que hará que el bucle termine. Esta implementación sólo incluye este criterio de parada.

```
% Método de Newton-Raphson para una función f definida.
% Deben estar definidas la función f.m y su derivada fprima.m
x0=2;
tol=0.00001;
seguir=true;
while seguir
x1=x0-f(x0)/fprima(x0);
% fprintf('x1 = %d \n',x1);
if abs(x0-x1)<tol
    seguir=false;
end
x0=x1;
end
```

Ej 6. Represente la función f(x)=cosh(x).*cos(x)-1 en el intervalo [4,5]. Se pretende encontrar la raíz de f que está en ese intervalo mediante el método de Newton-Raphson. (La derivada de la función cosh(x), coseno hiperbólico, es la función senh(x), seno hiperbólico) ¿Se consigue aproximar dicha raíz tomando como aproximación inicial x0=4? ¿y x0=4.1? (Sol. Si x0=4, se aproxima otra raíz, y si x0=4.1 se obtiene la aproximación x1=4.730041)

Ej 7. Aplique el método de Newton-Raphson a la ecuación f(x)=x^3+94*x^2-389*x+294=0 tomando como aproximación inicial a) x0=1.9, b) x0=2.1, c) x0=2. (Sol. Para x0=1.9 se aproxima a la raíz x=1, para x0=2.1 se aproxima a la raíz x=3, para x0=2 se aproxima a la raíz x=-98).

Ej 8. Modifique el método de Newton-Raphson de forma que se detenga el programa (y muestre el mensaje 'Derivada pequeña') antes de calcular x1 si el valor de f'(x0) en valor absoluto es menor que delta=0.001. (Puede probar el programa con f(x)=(x-2).*(x-7).*(x-2) y x0=5.33333333333)

Derivación simbólica

En el método de Newton-Raphson, para aproximar la solución de f(x)=0, es necesario tener definida la derivada de la función f. En Matlab (no así en Octave) se pueden hallar derivadas simbólicas mediante el comando diff. Observe en los ejemplos cómo se incluye la función a derivar entre comillas simples.

```
diff(sym('sin(x^2)'))
diff(sym('(sin(x)^2)'))
```

```
ans =

2*x*cos(x^2)


ans =

2*cos(x)*sin(x)
```

El resultado del comando diff lo podemos copiar y pegar en la definición de la función fprima.

Al usar el comando diff emplearemos los operadores de multiplicación, división y potenciación sin el punto que indica operaciones con vectores. Si los usamos, por ejemplo si ejecutamos la instrucción diff(sym('x.^2')), obtendremos un error.

Ej 9. Halle de forma simbólica la derivada de la función f(x)=cos(x)/sen(x). (Sol. f '(x)= -1 -cos(x)^2/sin(x)^2).

Iteración funcional

En este apartado nos planteamos el problema de hallar un punto fijo de la función g, es decir, hallar x tal que g(x)=x. Para hallar una aproximación al punto fijo de g, implementaremos un método de iteración funcional en el que x(k+1) se obtiene a partir de x(k) de la siguiente manera : x(k+1)=g(x(k)).

Una posible implementación del método puede encontrarse en el programa iteracion.m que requiere que se haya definido la función g.m .

```
% Iteración funcional para encontrar un punto fijo g(x)=x
% Debe proporcionarse la función g.m
% Implementa tres condiciones para salir del bucle
x0=2;
tol1=0.00001;
tol2=0.0001;
Nmax=100;
seguir=true;
N=0;
while seguir
    x1=g(x0);
    N=N+1;
    fprintf('N = %d, x1 = %d, g(x1)= %d \n',N,x1,g(x1));
    % Primera condición: Si dos aproximaciones sucesivas están lo bastante
    % cerca, se sale del bucle.
    if abs(x1-x0)<tol1
        seguir=false;
    end
    % Segunda condición: Si g(x1) es lo bastante cercano a x1, se sale
    % del bucle.
    if abs(g(x1)-x1)<tol2
        seguir=false;
    end
    % Tercera condición. Si se han efectuado más de Nmax iteraciones, se
    % sale del bucle.
    if N>Nmax
        seguir=false;
    end
% Actualizamos el valor de x0
x0=x1;
end
% Al salir del bucle mostramos el valor de la última iteración.
fprintf('Fin del bucle: x0 = %d, g(x0)= %d \n',x0,g(x0));
```

```
N = 1, x1 = -4.161468e-01, g(x1)= 9.146533e-01
N = 2, x1 = 9.146533e-01, g(x1)= 6.100653e-01
N = 3, x1 = 6.100653e-01, g(x1)= 8.196106e-01
N = 4, x1 = 8.196106e-01, g(x1)= 6.825059e-01
N = 5, x1 = 6.825059e-01, g(x1)= 7.759946e-01
N = 6, x1 = 7.759946e-01, g(x1)= 7.137247e-01
N = 7, x1 = 7.137247e-01, g(x1)= 7.559287e-01
N = 8, x1 = 7.559287e-01, g(x1)= 7.276348e-01
N = 9, x1 = 7.276348e-01, g(x1)= 7.467496e-01
N = 10, x1 = 7.467496e-01, g(x1)= 7.339006e-01
```

```
N = 11, x1 = 7.339006e-01, g(x1)= 7.425676e-01
N = 12, x1 = 7.425676e-01, g(x1)= 7.367349e-01
N = 13, x1 = 7.367349e-01, g(x1)= 7.406663e-01
N = 14, x1 = 7.406663e-01, g(x1)= 7.380191e-01
N = 15, x1 = 7.380191e-01, g(x1)= 7.398028e-01
N = 16, x1 = 7.398028e-01, g(x1)= 7.386015e-01
N = 17, x1 = 7.386015e-01, g(x1)= 7.394108e-01
N = 18, x1 = 7.394108e-01, g(x1)= 7.388657e-01
N = 19, x1 = 7.388657e-01, g(x1)= 7.392329e-01
N = 20, x1 = 7.392329e-01, g(x1)= 7.389856e-01
N = 21, x1 = 7.389856e-01, g(x1)= 7.391522e-01
N = 22, x1 = 7.391522e-01, g(x1)= 7.390400e-01
N = 23, x1 = 7.390400e-01, g(x1)= 7.391156e-01
Fin del bucle: x0 = 7.390400e-01, g(x0)= 7.391156e-01
```

Ej 10. Represente conjuntamente las gráficas y=x e y=cos(x) en el intervalo [0,3]. Ejecute el programa iteración.m para obtener una aproximación a una raíz de cos(x)=x con x0=0.25. (Sol. aprox. x=7.390261e-01, que se obtiene realizando 22 iteraciones)

Ej 11. Modifique la ecuación x^3-2.1*x^2-x+2.1=0 para que quede en la forma g(x)=x. Utilize el programa iteracion.m para intentar encontrar una raíz cercana a x0=2. (Sol. Si se plantea la ecuación x^3-2.1x^2+2.1=x no se obtiene convergencia. Si se plantea la ecuación x=-2.1/(x^2-2.1x-1) se obtiene una aproximación de otra raíz x=1. Si se plantea x=(2.1x^2+x-2.1)/x^2) se obtiene x0=2.099932 aproximación de la raíz exacta x=2.1).

Ej 12. Modifique el programa para que muestre un mensaje explicativo según el criterio de parada que se verifique.

Ej 13. Para hallar la raíz cuadrada de un número R se propone un método iterativo x(k+1)=g(x(k)) con g(x)=x*(x^2+3*R)/(3x^2+R). Programe el método de forma que la ejecución termine si la iteración cumple abs(x(n)^2-R)<delta=0.000001. Aplique el método para x0=1, R=2. (Sol. aprox. x=1.414214e+00, que se obtiene realizando tres iteraciones)

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
10-Mar-2017
```

Published with MATLAB® R2013a

Práctica 5. Sistemas de ecuaciones lineales I (Método de Gauss)

En la práctica 2 vimos cómo expresar un sistema de ecuaciones lineales en forma matricial, cómo comprobar que un vector es la solución de un sistema Ax=b, y cómo usar la 'división izquierda' para hallar la solución de un sistema Ax=b con matriz A regular.

También vimos ejemplos de sistemas mal condicionados, y sistemas donde la 'división izquierda' no da una solución porque el sistema es incompatible.

En esta práctica veremos cómo resolver sistemas triangulares y el método de Gauss para resolver sistemas de ecuaciones lineales.

Contents

Resolución de un sistema triangular superior.

Consideremos un sistema triangular de tres ecuaciones y tres incógnitas como el siguiente:

```
A11*x1 + A12*x2 + A13*x3 = b1,
         A22*x2 + A23*x3 = b2,
                  A33*x3 = b3.
```

Para hallar su solución podemos despejar x3 en la última ecuación, x3=b3/A33. Con este valor de x3, podemos despejar x2 en la segunda ecuación x2=(b2-A23*x3)/A22. Y con los valores hallados de x3 y x2 podemos despejar x1 en la primera ecuación x1=(b1-A12*x2-A13*x3)/A11.

Emplearemos este método para resolver un sistema de n ecuaciones lineales y n incógnitas cuya matriz de coeficientes tiene todos los coeficientes por debajo de la diagonal iguales a cero, es decir, es triangular superior.

Consideramos un sistema como el siguiente:

```
A(1,1)*x(1) + A(1,2)*x(2) + ... +  A(1,n-1)*x(n-1)  +  A(1,n)*x(n)  = b(1),
              A(2,2)*x(2) + ... +  A(2,n-1)*x(n-1)  +  A(2,n)*x(n)  = b(2),
```

...

```
                                  A(n-1,n-1)*x(n-1) + A(n-1,n)*x(n) = b(n-1),
                                                      A(n,n)*x(n) = b(n).
```

Podemos despejar x(n) de la última ecuación con lo que x(n)=b(n)/A(n,n). Conociendo x(n) podemos despejar x(n-1) de la penúltima ecuación.

Siguiendo este proceso, supongamos que ya hemos hallado x(n), x(n-1), ... x(i+1) y queremos despejar x(i) de la ecuación i-ésima

```
       A(i,i)*x(i) + A(i,i+1)*x(i+1) + ... + A(i,n)*x(n) = b(i),
```

entonces tenemos

```
       x(i)=(b(i) - S )/A(i,i),
```

donde S = A(i,i+1)*x(i+1) + ... + A(i,n)*x(n) .

Nos proponemos expresar el valor de S como un producto escalar de un vector fila por un vector columna.

```
S = A(i,i+1)*x(i+1) + ... + A(i,n)*x(n) .
```

En Matlab, el vector fila A(i,i+1:n) contiene los elementos que están en la fila i, y en las columnas desde la i+1 hasta la n, es decir A(i,i+1), A(i,i+2), A (i,i+3), ... A(i,n-1), A(i,n).

El vector columna x(i+1:n) contiene los elementos de x que van desde el i+1 hasta el n, es decir x(i+1), x(i+2), ..., x(n-1), x(n). Por tanto el valor de S se puede calcular como S=A(i,i+1:n)*x(i+1:n), con lo que usamos las operaciones entre vectores de Matlab.

Podemos escribir el pseudocódigo que permite resolver el sistema.

```
Entrada A, b.
n = número de filas de A.
Inicializamos x.
Para i=n, n-1, n-2, ..., 1
   x(i)= (b(i)- A(i,i+1:n)*x(i+1:n))/A(i,i)
Siguiente i.
```

El siguiente código de Matlab permite resolver un sistema triangular superior.

```
% Matriz del sistema
A=[1 3 2 2 1; 0 1 2 4 1; 0 0 2 1 -1; 0 0 0 3 -2; 0 0 0 0 2];
% Término independiente
b=[-2;-1;-2;-9;6];
% Hallamos el número de ecuaciones del sistema.
n=length(A);
% Definimos un vector-columna para almacenar la solución.
x=zeros(n,1);
% El bucle va desde la última ecuación hasta la primera
for i=n:-1:1
   x(i)= (b(i)- A(i,i+1:n)*x(i+1:n))/A(i,i);
end
disp('La solución del sistema es ')
disp(x)
```

```
La solución del sistema es
     1
    -2
     1
    -1
     3
```

Ej 1. Empleando el código que hemos visto escriba un script que permita resolver el sistema

```
x1 + 2*x2 + 3*x3 +   x4 +   x5 = 7,
       x2 + 4*x3 +   x4 -   x5 = 6,
          -   x3 + 2*x4 -   x5 = -5,
                   3*x4 +   x5 = -2,
                          2*x5 = 2.
```

Compruebe el resultado con la 'división izquierda' (Sol. x1=1, x2=0, x3=2, x4=-1, x5=1).

Ej 2. Modifique el programa para que imprima (usando disp o fprintf) un mensaje como 'El elemento 3 de la diagonal es cero' si algún elemento de la diagonal vale cero, pero que no interrumpa el método. Aplique el programa a la resolución del sistema A*x=b donde A=[1,2,3,1,1;0,2,4,-1,-1;0,0,0,2,-1;0,0,0,3,1;0,0,0,0,-2], b=[2;3;-5;-5;-2]. (Sol. x5=1, x4=-2.) (El sistema planteado tiene infinitas soluciones, es compatible indeterminado. Resolviendo a mano se podrían dar las soluciones dependiendo de un parámetro : x5=1, x4=-2, x3=x1-1, x2=-2*x1+3)

Resolución de un sistema triangular inferior.

Consideremos ahora un sistema de n ecuaciones lineales y n incógnitas cuya matriz de coeficientes tiene todos los coeficientes por encima de la diagonal iguales a cero, es decir, es triangular inferior.

```
A(1,1) *x(1)                                                    =b(1),
A(2,1) *x(1) +  A(2,2) *x(2)                                    =b(2),
...
A(n-1,1)*x(1) + A(n-1,2)*x(2) + ... + A(n-1,n-1)*x(n-1)          =b(n-1),
A(n,1) *x(1) +   A(n,2)*x(2) + ... +   A(n,n-1)*x(n-1) + A(n,n)*x(n)=b(n).
```

En este sistema se puede despejar x(1) de la primera ecuación. Una vez hallado el valor de x(1), se puede hallar x(2) despejando en la segunda ecuación. Supongamos que ya hemos hallado x(1), x(2), ..., x(i-1). Considerando la ecuación i-ésima

(A(i,1)*x(1) + A(i,2)*x(2) + ... + A(i,i-1)*x(i-1)) + A(i,i)*x(i) =b(i)

podemos despejar x(i).

Ej 3. a) Despeje (en papel) x(i) en la última ecuación. b) Exprese (en papel) el valor de x(i) usando un producto escalar de un vector fila por un vector columna. c) Programe la resolución de un sistema triangular inferior. d) Compruebe el programa resolviendo el sistema A*x=b donde A=[1,0,0,0,0; 0,2,0,0,0; 1,1,1,0,0; 0,-1,0,3,0; -2,0,1,0,-2], b=[1;4;6;10;-9]. (Sol. x1=1, x2=2, x3=3, x4=4, x5=5)

Método de Gauss (Ejemplo del proceso de hacer ceros).

El método de Gauss permite resolver sistemas de ecuaciones lineales. Este método consiste en realizar, en primer lugar, determinadas operaciones elementales por filas sobre la matriz ampliada del sistema hasta que ésta se transforme en otra en la que todas sus columnas salvo la última formen una matriz triangular superior (proceso de hacer ceros). Y, a continuación, resolver el sistema triangular superior asociado a la matriz resultante (que es equivalente al original).

Veamos un ejemplo de la primera parte del método de Gauss (proceso de hacer ceros):

Consideramos el siguiente sistema de tres ecuaciones lineales y tres incógnitas

```
x + 2*y + 3*z = 5,
2*x +   y       = 1,
x +   y + 2*z = 4.
```

Para hacer las operaciones elementales sobre las filas de la matriz ampliada [A b] lo que haremos será realizar las mismas operaciones elementales sobre la matriz A y sobre el vector b de su formulación matricial.

Definimos la matriz de coeficientes del sistema y el vector de términos independientes.

```
A=[1 2 3; 2 1 0; 1 1 2];
b=[5;1;4];
```

Hacemos ceros en la primera columna debajo del elemento pivote A(1,1) .

A la segunda fila le restamos dos veces la primera

```
A(2,:)=A(2,:)-2*A(1,:) ;
b(2)=b(2)-2*b(1) ;
% A la tercera fila le restamos la primera
A(3,:)=A(3,:)-A(1,:) ;
b(3)=b(3)-b(1) ;
% Vemos que hemos hecho ceros en la primera columna
disp([A b])
```

```
     1     2     3     5
     0    -3    -6    -9
     0    -1    -1    -1
```

Para hacer cero en A(3,2) debemos restar a la tercera fila un múltiplo de la segunda.

```
A(3,:)=A(3,:)-(-1/-3)*A(2,:) ;
b(3)=b(3)-(-1/-3)*b(2) ;
% Vemos que hemos hecho ceros debajo de la diagonal, con lo que el sistema
% que resulta es triangular superior.
disp([A b])
```

```
     1     2     3     5
     0    -3    -6    -9
     0     0     1     2
```

Método de Gauss (Programación del método).

En lo que sigue vamos a implementar el método de Gauss para sistemas en los que en el proceso de hacer ceros no es preciso permutar filas. La programación del método de Gauss cuando se precisa permutar filas en el proceso de hacer ceros es algo más complicada y no la veremos aquí.

Siguiendo la notación dada en clase al explicar el método de Gauss, la variable i indicará la columna en la que vamos a hacer ceros. La variable j indica la fila en la que estamos haciendo ceros.

Para simplificar la notación y ahorrar memoria, no usaremos superíndices, llamaremos A0 a la matriz original del sistema y b0 al término independiente original. La matriz A y el vector b contendrán los sucesivos sistemas equivalentes, si se acaban de hacer ceros en la columna i, A indicará la matriz con superíndice i+1.

Dado i, suponemos que ya se han hecho ceros en las columnas 1, ..., i-1 y hacemos ceros en la columna i en las filas j>=i+1 mediante la operación por filas 'sumar a la fila j la fila i multiplicada por lj ' donde lj es el valor lj=-A(j,i)/A(i,i) .

El programa ProgGauss.m implementa el proceso de hacer ceros del método de Gauss para resolver sistemas en los que en el proceso de hacer ceros no es preciso permutar filas.

Dicho programa incluye las instrucciones tic y toc que sirven para mostrar el tiempo que emplea Matlab en ejecutar un conjunto de instrucciones. Basta con poner la instrucción tic antes del conjunto de instrucciones y toc al final. Al ejecutarse la instrucción toc se muestra el tiempo que emplea Matlab en ejecutar el conjunto de instrucciones.

Definimos en forma matricial el sistema que queremos resolver.

```
A0=[1,2,3,2 ; 3,-3,-6,1 ; 2,1,7,0 ; 2,3,2,2];
```

```
b0=[12;-10;22;9];
% Inicializamos la matriz A y el vector b en los que efectuaremos las
% operaciones elementales por filas.
A=A0;
b=b0;
n=length(A);
```

Haremos ceros en la columna i, donde i va desde la primera columna a la penúltima.

```
for i=1:n-1
   % j va desde la fila i+1 hasta la última.
   for j=i+1:n
   lj=- A(j,i)/A(i,i);
   % Sumamos a la fila j un múltiplo de la fila i.
   A(j,:)= A(j,:)+lj * A(i,:);
   % Efectuamos la misma operación en el vector b.
   b(j)=b(j)+lj * b(i);
   end
   % fprintf('Tras hacer ceros en la columna %d\n',i);
   % disp('queda el sistema equivalente:')
   % disp([A b])
end
```

Una vez transformado el sistema original en otro equivalente pero triangular superior, hay que resolver éste último y para ello se puede utilizar el método que se vió al principio de esta práctica para resolver sistemas triangulares superiores o también, por comodidad, la división izquierda.

```
x=A\b
```

```
x =

     1
    -1
     3
     2
```

Podemos comprobar que la solución del sistema triangular A*x=b obtenida es solución del sistema original A0*x=b0.

```
A0*x-b0
```

```
ans =

     0
     0
     0
     0
```

Ej 4. Comprueba que el sistema A0*x=b0 dado por A0=[1,0,3,2,-1 ; 3,0,-2,1,1 ; 2,7,1,0,3 ; 1,2,3,-1,2; 1,2,3,-1,0] b0=[5;4;3;2;1] es compatible determinado y que A0\b0 calcula la solución del sistema, Comprueba que, sin embargo, el método de Gauss implementado no obtiene la solución del sistema. ¿A qué se debe? (Indicación : observe las operaciones que se realizan con los elementos A(i,i).)

Ej 5. Escriba un programa que aplique el método de Gauss al sistema A0*x=b0 donde A0=[1,0,3,2,-1 ; 3,1,-2,1,1 ; 2,7,1,0,3 ; 1,2,3,-1,2; 1,2,3,-1,0]; b0= [17;16;44;22;20] de forma que transforme el sistema a uno triangular superior equivalente, y luego aplique el método visto para resolver sistemas triangulares superiores. (Sol. x1=5, x2=4, x3=3, x4=2, x5=1).

Las operaciones que se aplican en esta implementación del método de Gauss actuan sobre toda la fila a pesar de que los primeros elementos de cada fila serán cero, con lo cual se están efectuando operaciones innecesarias.

Así al hacer ceros en la columna i, en las filas j=i+1,...,n los elementos A(j,i) son cero. Se proponen dos formas de evitar operaciones innecesarias modificando la línea A(j,:)= A(j,:)+lj * A(i,:);

i) Emplear un bucle for con variable k que vaya operando en las columnas de la i+1 hasta la n. ii) Hacer la operación por filas indicando que deben considerarse las columnas de la i+1 hasta la n, es decir, debe asignarse sólamente A(j,i+1:n).

Ej 6. Compruebe (con papel) que, en el método de Gauss implementado, los elementos A(j,i) para j=i+1,...,n son cero.

Ej 7. Implemente el método de Gauss sustituyendo la línea A(j,:)= A(j,:)+lj * A(i,:) por un bucle for como se indica en i) y asignando A(j,i)=0.

Puede comprobar el método con el mismo ejemplo A0=[1,2,3,2 ; 3,-3,-6,1 ; 2,1,7,0 ; 2,3,2,2]; b0=[12;-10;22;9];

Ej 8. Implemente el método de Gauss sustituyendo la línea A(j,:)= A(j,:)+lj * A(i,:) por una línea que comienze por A(j,i+1:n)= ... y asignando A(j,i)=0.

Puede comprobar el método con el mismo ejemplo A0=[1,2,3,2 ; 3,-3,-6,1 ; 2,1,7,0 ; 2,3,2,2]; b0=[12;-10;22;9];

Ej 9. ¿Cuál de las dos implementaciones sugeridas en i) y ii) cree que es más eficiente? Calcule con las instrucciones tic y toc el tiempo que tardan los dos nuevos programas en resolver un sistema.

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
16-Mar-2017
```

Published with MATLAB® R2013a

Práctica 6. Sistemas de ecuaciones lineales II (Descomposición LU)

En esta práctica estudiaremos la implementación de algunos métodos de factorización LU para resolver sistemas Ax=b con A regular y factorizable en la forma LU. Concretamente se tratarán los métodos de Doolittle y de Crout.

Para ello, estudiaremos en primer lugar la implementación con MatLab de las factorizaciones de Doolittle y de Crout de una matriz A, que consisten en encontrar las matrices L y U tales que L*U = A, siendo L una matriz triangular inferior y U una matriz triangular superior, con unos en la diagonal de L en el caso de Doolittle, y con unos en la diagonal de U en el caso de Crout.

Antes de esto resultará útil saber cómo hallar con MatLab los sumatorios usando vectores.

Contents

Vectorización del sumatorio

En primer lugar, veamos como implementar los sumatorios que aparecen en la factorización de Doolittle de una matriz mediante un producto escalar de dos vectores. Los sumatorios que aparecen en la factorización de Crout son análogos.

En la factorización de Doolittle, al hallar U(k,j) debemos implementar el sumatorio de L(k,r)*U(r,j) donde r=1, 2, ..., k-1.

Es decir, debemos hallar L(k,1)*U(1,j) + L(k,2)*U(2,j) + ... + L(k, k-1)*U(k-1 ,j)

Este sumatorio puede vectorizarse como un producto escalar del vector fila L(k, 1:k-1) por el vector columna U(1:k-1 ,j).

De forma similar, al hallar L(i,k) debemos implementar el sumatorio de L(i,r)*U(r,k) donde r=1, 2, ..., k-1.

Este sumatorio puede vectorizarse como el producto escalar L(i , 1:k-1)*U(1:k-1 , k)

Factorización de Doolittle de una matriz

En la factorización de Doolitte la matriz L contiene unos en la diagonal. Comenzamos definiendo la matriz A que vamos a factorizar e inicializando variables.

```
A=[[6,16,21,33] ; [2,20,18,16] ; [3,18,19,21] ; [1,2,3,4]];
n=length(A);
% Inicializamos las matrices U y L como matrices cuadradas de orden n
% formadas por ceros.
U=zeros(n);
L=zeros(n);
```

Ahora implementamos el caso k=1, ya que en el algoritmo correspondiente aparecen sumatorios en los que el índice r recorre desde 1 hasta k-1 cuando k=2,...n. Cuando k=1, se entiende que este sumatorio no debe realizarse, y por esto en Matlab debemos hacer aparte el caso k=1. Observe cómo se asigna la primera fila de U y la primera columna de L mediante vectores.

```
U(1, 1:n)=A(1, 1:n);
L(1,1)=1;
L(2:n , 1)=(  A(2:n , 1)  )/U(1,1);
```

Hallaremos el resto de los elementos de L y de U siguiendo el siguiente orden :

```
Para 2<=k<=n

Para k<=j<=n
 U(k,j)
Siguiente j

L(k,k)=1

Para k+1<=i<=n
 L(i,k)
Siguiente i

Siguiente k
```

En los elementos que requieren calcular un sumatorio, estos se han calculado antes por claridad, aunque podría haberse incluido el sumatorio en la misma linea.

```
for k=2:n
   for j=k:n
       suma=L(k , 1:k-1)*U(1:k-1 , j);
       U(k,j)=A(k,j)-suma;
   end
   L(k,k)=1;
   for i=k+1:n
      suma=L(i , 1:k-1)*U(1:k-1 , k);
      L(i,k)=(A(i,k)-suma) / U(k,k);
   end
end
```

Mostramos el resultado y comprobamos la factorización A=L*U.

```
disp('Resultado :')
disp('L =');
disp(L);
disp('U =');
disp(U);
disp('Comprobación :')
disp('A-L*U =');
disp(A-L*U);
```

```
 Resultado :
 L =
     1.0000         0         0         0
     0.3333    1.0000         0         0
     0.5000    0.6818    1.0000         0
     0.1667   -0.0455         0    1.0000

 U =
     6.0000   16.0000   21.0000   33.0000
          0   14.6667   11.0000    5.0000
          0         0    1.0000    1.0909
          0         0         0   -1.2727

 Comprobación :
 A-L*U =
      0     0     0     0
      0     0     0     0
      0     0     0     0
      0     0     0     0
```

Ej 1. Emplee el programa proporcionado para hallar la factorización de Doolittle de la matriz A=[[1,2,-6,2] ; [2,-1,-5,-3] ; [3,-54,77,-69] ; [4,3,-138,-104]];

(Sol. L=[[1,0,0,0] ; [2,1,0,0] ; [3,12,1,0] ; [4,1,-11,1]];

U=[[1,2,-6,2] ; [0,-5,7,-7] ; [0,0,11,9] ; [0,0,0,-6]];)

Factorización de Crout de una matriz

De forma similar se puede implementar la factorización de Crout, en la que aparecen unos en la diagonal de U. Después de considerar el caso k=1, para hallar la primera columna de L y la primera fila de U, podemos calcular el resto de elementos de L y de U en el siguiente orden.

```
Para 2<=k<=n

Para k<=i<=n
 L(i,k)
Siguiente i

U(k,k)=1

Para k+1<=j<=n
 U(k,j)
Siguiente j

Siguiente k
```

Ej 2. Implemente la factorización de Crout y compruébela factorizando la matriz A=[[2,0,-6,-4] ; [11,1,-22,-17]; [3,4,38,17]; [-1,-2,-19,-9]];

(Sol. L=[[2,0,0,0] ; [11,1,0,0] ; [3,4,3,0] ; [-1,-2,0,-1]]

U=[[1,0,-3,-2] ; [0,1,11,5] ; [0,0,1,1] ; [0,0,0,1]])

Resolución de sistemas usando una descomposición L*U.

Si suponemos que A = L*U, el sistema A*x = b puede escribirse de la forma L*U*x = b. Para hallar x podemos introducir un vector auxiliar y = U*x, de forma que nuestro sistema se escribe como L*y = b, donde despejar y se reduce a resolver un sistema triangular. Una vez conocido el vector y, para hallar el vector x basta con resolver el sistema U*x = y, que es de nuevo un sistema triangular.

Así pues, resolver un sistema Ax=b usando la descomposición LU se reduce a hallar una factorización LU de la matriz A , y a resolver a continuación dos sistemas triangulares. Tanto la factorización LU de A como la resolución de los sistemas triangulares se realiza con pequeños programas como los que hemos descrito en esta práctica 6 y en la práctica 5.

De forma concisa repetimos los pasos que deden darse para resolver un sistema usando una descomposición L*U :

```
1- Descomponer A = L*U.
2- Resolver L y = b con lo que obtenemos y.
3- Resolver U x = y con lo que obtenemos x.
```

Ej 3. Sabiendo que A = L*U con L y U las matrices dadas a continuación y que b es el vector proporcionado, resuelva el sistema A*x = b siguiendo los pasos descritos en esta sección (Emplee, por comodidad, la división izquierda para resolver los dos sistemas triangulares). Halle A*x-b para comprobar que el vector x hallado es solución del sistema. (Sol. x=[1; 0; -2; 3].)

```
L=[[1, 0, 0, 0]; [2, 1, 0, 0]; [-2, 3, 1, 0]; [-1, -1, -2, 1]] ;
U=[[-1, 4, 1, 2]; [0, 2, -2, 0]; [0, 0, -2, -1]; [0, 0, 0, 3]] ;
b=[3; 10; 7; 0] ;

% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
17-Mar-2017
```

Published with MATLAB® R2013a

Práctica 7. Sistemas de ecuaciones lineales III (Métodos iterativos)

En esta práctica implementaremos los métodos iterativos de Jacobi y de Gauss-Seidel y veremos un par de criterios de convergencia que se pueden aplicar a estos métodos.

Contents

Matrices D, L y U para una matriz A.

En clase se vió como expresar el método de Jacobi y Gauss-Seidel a partir de unas matrices D, L y U de forma que A=D-L-U donde D es una matriz diagonal cuya diagonal coincide con la de la matriz A, L es una matriz triangular inferior que bajo su diagonal tiene a los opuestos de los coeficientes de A que están bajo la diagonal de A, y U es una matriz triangular superior que sobre su diagonal tiene a los opuestos de los coeficientes de A que están sobre la diagonal de A.

Veamos como construir estas matrices D, L y U usando comandos de Matlab.

```
A=[
    [1 .2 .3 .4]; [.5 6 .7 .8] ; [.9 .10 11 .12]; [.13 .14 .15 16]
];
```

Si A es una matriz, diag(A) es un vector columna con los elementos de la diagonal de A.

Si v es un vector con n componentes, diag(v) nos da una matriz cuadrada de orden n con las componentes de v en la diagonal. Por tanto diag(diag(A)) es una matriz diagonal cuyos elementos de la diagonal coinciden con los de la matriz A.

```
D=diag(diag(A))
```

```
D =

     1     0     0     0
     0     6     0     0
     0     0    11     0
     0     0     0    16
```

El comando tril nos da la parte inferior de una matriz. Veamos unos ejemplos.

```
tril(A,0)
tril(A,1)
tril(A,-1)
```

```
ans =

    1.0000         0         0         0
    0.5000    6.0000         0         0
    0.9000    0.1000   11.0000         0
    0.1300    0.1400    0.1500   16.0000


ans =

    1.0000    0.2000         0         0
    0.5000    6.0000    0.7000         0
    0.9000    0.1000   11.0000    0.1200
    0.1300    0.1400    0.1500   16.0000


ans =

         0         0         0         0
    0.5000         0         0         0
    0.9000    0.1000         0         0
    0.1300    0.1400    0.1500         0
```

La instrucción tril(A,0) se puede escribir como tril(A). Para definir la matriz L usamos la siguiente instrucción:

```
L=-tril(A,-1)
```

```
L =

         0         0         0         0
   -0.5000         0         0         0
   -0.9000   -0.1000         0         0
   -0.1300   -0.1400   -0.1500         0
```

El comando triu nos da la parte superior de una matriz.

```
triu(A,0)
triu(A,1)
triu(A,-1)
```

```
ans =

    1.0000    0.2000    0.3000    0.4000
         0    6.0000    0.7000    0.8000
         0         0   11.0000    0.1200
         0         0         0   16.0000


ans =

         0    0.2000    0.3000    0.4000
         0         0    0.7000    0.8000
         0         0         0    0.1200
         0         0         0         0


ans =

    1.0000    0.2000    0.3000    0.4000
    0.5000    6.0000    0.7000    0.8000
         0    0.1000   11.0000    0.1200
         0         0    0.1500   16.0000
```

De igual forma triu(A,0) puede escribirse como triu(A). Para definir la matriz U usamos la siguiente instrucción:

```
U=-triu(A,1)
```

```
U =

         0   -0.2000   -0.3000   -0.4000
         0         0   -0.7000   -0.8000
         0         0         0   -0.1200
         0         0         0         0
```

Con estas definiciones la matriz A se puede expresar como D-L-U. Comprobémoslo con la siguente instrucción, que nos devuelve una matriz llena de ceros indicando que las dos matrices son iguales elemento a elemento.

```
A-(D-L-U)
```

```
ans =

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Método de Jacobi

Veamos cómo emplear el método de Jacobi para resolver un sistema A*x=b. Ya hemos definido la matriz del sistema A, y definimos ahora el vector término independiente b como un vector columna.

```
b=[-0.1; -12.6; 33.22; -63.7];
```

Este método iterativo comienza con una aproximación inicial que podemos llamar x0, y dada una iteración xi calcula la siguiente resolviendo el sistema de ecuaciones lineales con matriz de coeficientes D y vector de términos independientes (L+U)*xi+b. Para resolver dicho sistema con MatLab utilizaremos, por comodidad, la división izquierda.

Vamos a calcular las primeras iteraciones del método de Jacobi para resolver A*x=b tomando como aproximación inicial el vector x0 con cuatro filas, una columna y todas sus componentes iguales a 1. (Se puede comprobar que la solución exacta del sistema es [1; -2; 3; -4]).

```
x0=ones(4,1)
x1=D\((L+U)*x0+b)
x2=D\((L+U)*x1+b)
x3=D\((L+U)*x2+b)
x4=D\((L+U)*x3+b)
```

```
x0 =

     1
     1
     1
     1


x1 =

   -1.0000
   -2.4333
    2.9182
   -4.0075


x2 =

    1.1142
   -1.8228
    3.1677
   -3.9792


x3 =

    0.9059
   -2.0319
    2.9888
   -4.0041


x4 =

    1.0113
   -1.9903
    3.0080
   -3.9989
```

Para mostrar los datos en una tabla debemos trasponer los vectores.

```
disp('       k       x(1)      x(2)      x(3)      x(4)   ')
disp('   ------------------------------------------------')
disp([1 transpose(x1)])
disp([2 transpose(x2)])
disp([3 transpose(x3)])
disp([4 transpose(x4)])
```

```
       k       x(1)      x(2)      x(3)      x(4)
   ------------------------------------------------
    1.0000   -1.0000   -2.4333    2.9182   -4.0075

    2.0000    1.1142   -1.8228    3.1677   -3.9792

    3.0000    0.9059   -2.0319    2.9888   -4.0041

    4.0000    1.0113   -1.9903    3.0080   -3.9989
```

Ej 1. Halle la quinta iteración en el ejemplo descrito. (Solución: x5=[0.9952;-2.0020;2.9990;-4.0003])

Ej 2. Considere el sistema A*x = b donde A=[[3 2 -1 1]; [-1 2 2 0]; [0 2 3 1]; [1 -1 0 2]] y b=[11;2;1;-5] . Halle la vigésima iteración del método de Jacobi comenzando con la aproximación inicial x0=[2; 2; 2; 2]. (Indicación: llame x a todas las iteraciones, y dentro de un bucle for calcule cada iteración x resolviendo, usando la división izquierda, el sistema de ecuaciones lineales con matriz de coeficientes D y vector de términos independientes (L+U)*x+b). (Solución: x20=[2.0024;2.9997;-0.9982;-2.0022]. Se puede

comprobar que la solución exacta del sistema es x=[2;3;-1;-2])

Ej 3. Considere el sistema A*x = b donde A=[[2 3 4 1]; [-1 2 3 0]; [0 2 3 1]; [1 -1 0 1]] y b=[7;1;1;-3] . Halle la vigésima iteración del método de Jacobi comenzando con la aproximación inicial x0=[2; 2; 2; 2]. (Solución: x20=[20.3593;-35.8838;34.9481;-33.9740]. Se puede comprobar que la solución exacta del sistema es x=[2;3;-1;-2], aunque el método de Jacobi no converge en este caso)

Método de Gauss-Seidel

El método de Gauss-Seidel se puede implementar de forma similar teniendo en cuenta que, dada una iteración xi, dicho método calcula la siguiente resolviendo el sistema de ecuaciones lineales con matriz de coeficientes D-L y vector de términos independientes U*xi+b.

Ej 3b. Modifique el programa realizado en el último ejercicio de forma que se halle la vigésima iteración del método de Gauss-Seidel. (Solución: x20= [88.7700;-686.8669;652.5354;-778.6369]. En este caso el método de Gauss-Seidel tampoco converge)

Convergencia. Radio Espectral.

Dada una aproximación inicial x(0) y un método iterativo de la forma M*x(k)=N*x(k-1)+b con M regular, este método es convergente si y solo si el radio espectral de la matriz B=inv(M)*N es menor que 1.

El radio espectral de una matriz es el máximo de las normas de los valores propios de la matriz.

Dada una matriz, podemos calcular sus valores propios (eigenvalues) con la función de Matlab eig(matriz) y si aplicamos la función abs a este vector obtendremos un vector cuyas componentes son las normas de los valores propios de la matriz; por último, la función max aplicada sobre el vector de las normas proporciona el máximo de las mismas, es decir, el radio espectral de la matriz.

```
BJ=inv(D)*(L+U)
matriz=BJ;
r= max(abs(eig(matriz)))
```

```
BJ =

         0   -0.2000   -0.3000   -0.4000
   -0.0833         0   -0.1167   -0.1333
   -0.0818   -0.0091         0   -0.0109
   -0.0081   -0.0088   -0.0094         0


r =

    0.2441
```

Ej 4. Halle el radio espectral de la matriz B=[[1 0 1]; [0 1 3]; [1 2 3]]. Si M es una matriz regular y B=inv(M)*N ¿es convergente el método iterativo de la forma M*x(k)=N*x(k-1)+b ? ¿Por qué? (Solución: el radio espectral de B es r=4.8284 y, por tanto, el método no es convergente)

Ej 5. Escriba un programa en el que para una matriz A dada se muestren en pantalla los radios espectrales de las matrices B de Jacobi y de Gauss-Seidel así como un mensaje que informe de si el método de Jacobi para resolver un sistema con matriz de coeficientes A converge o no converge, y otro que informe de si el método de Gauss-Seidel para resolver un tal sistema converge o no converge. Pruebe el programa con la matriz A=[[2 1 1]; [2 1 0]; [1 2 3]]. (Sol: el radio espectral de la matriz B del método de Jacobi es rJ=1.2965, el radio espectral de la matriz B del método de Gauss-Seidel es rGS=0.5000, por tanto el método de Jacobi no converge y el método de Gauss-Seidel si converge.)

Convergencia. Matriz estrictamente diagonal dominante.

En algunos casos puede asegurarse la convergencia de los métodos de Jacobi y Gauss-Seidel sin necesidad de hallar el radio espectral de una matriz.

Si en un sistema A*x=b se tiene que para cada fila i de la matriz A el coeficiente A(i,i) en valor absoluto es mayor que la suma de los valores absolutos del resto de los coeficientes de la fila (es decir, A es estrictamente diagonal dominante), entonces los métodos de Jacobi y de Gauss-Seidel convergen.

La función diagdom.m permite saber si una matriz dada es o no estrictamente diagonal dominante. En el código de esta función, la matriz A es el argumento de la función, y el valor que devuelve la función es la variable diagdom.

Ej 6. Estudie la función diagdom.m y explique la condición que aparece en el bucle while.

Ej 7. Construya una matriz de orden 3 por 3 que sea estrictamente diagonal dominante, y otra del mismo orden que no lo sea. Escriba un programa que muestre en pantalla esas dos matrices e informe de si cada una de ellas es estrictamente diagonal dominante o no (dicho programa deberá utilizar la función diagdom.m).

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
05-Apr-2017
```

Published with MATLAB® R2013a

Práctica 8. Interpolación

En esta práctica vamos a ver cómo definir y utilizar polinomios en Matlab y cómo hallar con MatLab el polinomio de interpolación de Lagrange, es decir, el polinomio de grado menor o igual que n que interpola n+1 datos de Lagrange. Este polinomio lo hallaremos resolviendo un sistema de ecuaciones lineales y, también, usando la forma de Newton de dicho polinomio. Para esta última necesitaremos aprender a construir con MatLab la tabla de diferencias divididas.

Contents

Polinomios en Matlab

En Matlab se pueden expresar polinomios mediante vectores. Concretamente el polinomio p1(x) = aN*x^N + ... + a2*x^2 + a1*x +a0 se expresa mediante un vector con sus coeficientes, p1 = [aN, ..., a2, a1, a0]. Por ejemplo el polinomio p(x) = x^3 + 3*x^2 - 7*x + 5 se expresa como

```
p=[1 3 -7 5]
```

```
p =

     1     3    -7     5
```

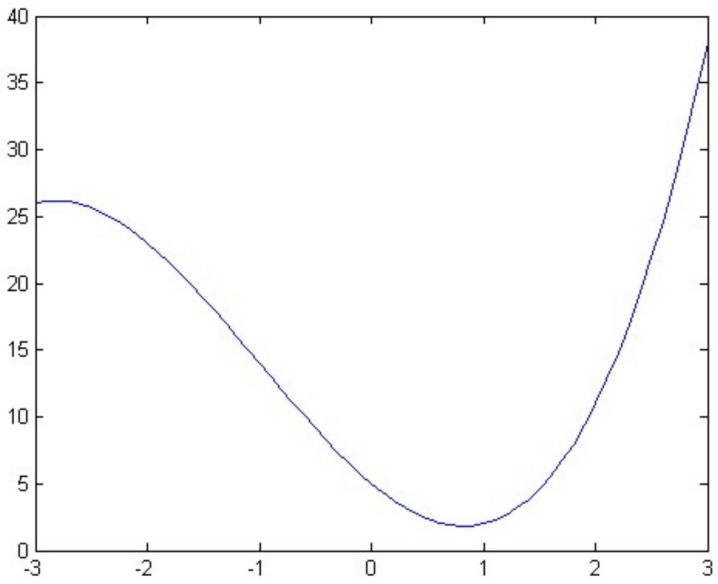
Para evaluar un polinomio en x0 empleamos el comando polyval(polinomio, x0), donde x0 puede ser un valor numérico o un vector. Por ejemplo para evaluar el polinomio p en 1 y obtener p(1)=1^3 + 3*1^2 - 7*1 + 5 = 2 usamos la instrucción

```
polyval(p,1)
```

```
ans =

     2
```

Si quisieramos representar gráficamente la función p(x) podríamos hacerlo definiendo un vector de abscisas xD y evaluando p en ese vector para obtener un vector de ordenadas yD, del siguiente modo:

```
xD=[-3:0.1:3];
yD=polyval(p,xD);
plot(xD,yD)
```



Para hallar las raíces de un polinomio p podemos emplear el comando roots. La siguiente instrucción obtiene las raíces de p, vemos que una de ellas es real y las otras dos son complejas.

```
roots(p)
```

```
ans =

  -4.7111 + 0.0000i
   0.8556 + 0.5739i
   0.8556 - 0.5739i
```

Para sumar dos polinomios debemos tener en cuenta sus grados. Debemos rellenar con ceros el polinomio de menor grado, y así sumar dos vectores de la misma dimensión. Veamos como efectuar la siguiente suma : (x^5 + 2*x^3 + x) + (x^2 + 2*x +1) = x^5 + 2*x^3 + x^2 + 3*x + 1 .

```
p1=[1, 0, 2, 0, 1, 0];
p2=[1, 2, 1];
p1+[ 0, 0, 0, p2]
```

```
ans =

     1     0     2     1     3     1
```

Para multiplicar dos polinomios emplearemos el comando conv. Así para efectuar la multiplicación (x^2 + 2*x + 3)*(-x^3 + 1) = -x^5 - 2*x^4 - 3*x^3 + x^2 + 2*x + 3 definiremos dos vectores q1 y q2, y luego los multiplicaremos

```
q1=[1 2 3]
q2=[-1 0 0 1]
conv(q1,q2)
```

```
q1 =

     1     2     3


q2 =

    -1     0     0     1


ans =

    -1    -2    -3     1     2     3
```

Ej 1. Halle el producto de los dos polinomios p1(x)=x^2 + 2*x + 1, p2(x)= x^3 + x^2 + 1. Evalue el resultado en x=2. (Sol. p1(x)*p2(x)= x^5 + 3*x^4 + 3*x^3 + 2*x^2 + 2*x + 1. p1(2)*p2(2) = 117)

Polinomio de interpolación de Lagrange

Dados los nodos x0, x1, x2, x3, ..., xN y los valores en dichos nodos y0, y1, y2, y3, ..., yN , queremos hallar el polinomio

p(x)=aN*x^N + ... + a2*x^2 + a1*x +a0

tal que p(x0)=y0, p(x1)=y1, ..., p(xN)=yN.

Decir que el polinomio p(x)=aN*x^N + ... + a2*x^2 + a1*x +a0 verifica las condiciones de interpolación p(x0)=y0, p(x1)=y1, ..., p(xN)=yN equivale a decir que los coeficientes aN,...,a2,a1,a0 del polinomio p(x) satisfacen un sistema de ecuaciones lineales cuyo vector de términos independientes es el vector y=[y0; y1; y2; y3; ...; yN], y cuya matriz de coeficientes es la matriz de Vandermonde formada por los elementos x(i,j)=xi^(n-j) con i,j=0,...,n (que puede obtenerse en Matlab mediante el comando vander(x), donde x=[x0, x1, x2, x3, ..., xN]).

Por tanto, los coeficientes del polinomio de interpolación de Lagrange p(x) se pueden obtener resolviendo dicho sistema de ecuaciones lineales.

Como ejemplo nos planteamos encontrar el polinomio de grado menor o igual que 3 que cumple p(1)=1, p(2)=1, p(3)=2, p(4)=6. (Ref: Diez lecciones de Cálculo Numérico. J. M. Sanz Serna.)

```
x=[1, 2, 3, 4]
y=[1; 1; 2; 6]
vander(x)
```

```
x =

     1     2     3     4
```

```
y =

     1
     1
     2
     6


ans =

     1     1     1     1
     8     4     2     1
    27     9     3     1
    64    16     4     1
```

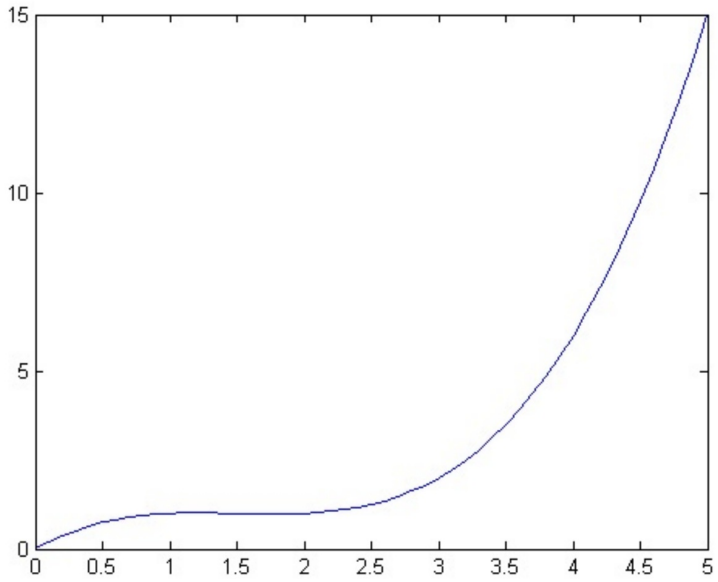
Obtenemos la solución del sistema vander(x)*p=y mediante la 'división izquierda'. Para ello el vector y debe ser un vector columna.

```
p=vander(x)\y
```

```
p =

    0.3333
   -1.5000
    2.1667
    0.0000
```

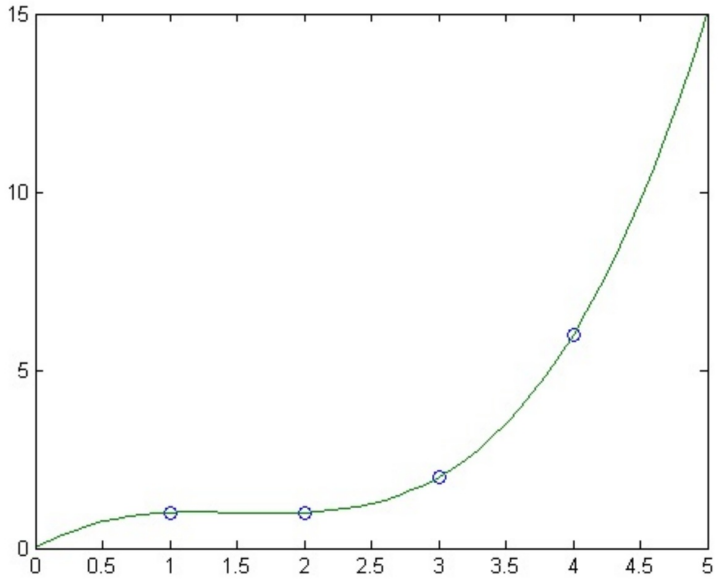
Representamos el polinomio p evaluándolo en los valores del vector xd

```
xd=[0:0.1:5];
yd=polyval(p,xd);
plot(xd,yd)
```



Para observar gráficamente el polinomio de interpolación representaremos los datos de interpolación (los vectores x e y) mediante círculos, y representaremos el polinomio p mediante los vectores xd e yd.

```
plot(x,y,'o',xd,yd)
```



Ej 2. Halle el polinomio q de grado cuatro que verifica que q(2)= 3, q(4)=5, q(5)=8, q(7)=10, q(10)= 4. Halle el determinante y el número de condición (usando la norma 2) de la matriz de Vandermonde asociada a los valores dados. (Observe que el determinante es distinto de cero y que el número de condición es mucho mayor que uno) (Sol. Coeficientes de q = [0.0347; -0.8917; 7.5931; -23.7583; 26.7222]. Determinante = 129600. Número de condición = 4.1024e+005)

Diferencias divididas

A continuación vamos a construir con Matlab la tabla de diferencias divididas para el polinomio de interpolación de Lagrange p(x) con nodos x(0), x(1), ..., x(N-1), x(N) (Ref: Diez lecciones de Cálculo Numérico. J. M. Sanz Serna.) (Ref: Numerical methods using Matlab. J. H. Mathews y K. D. Fink.)

Construiremos una tabla como la que se muestra, donde xN-1 es el nodo x(N-1), xN-2 el nodo x(N-2), etc.

```
x0  p[x0]
x1  p[x1]  p[x0,x1]
x2  p[x2]  p[x1,x2]    p[x0,x1,x2]
 .    .        .            .       .
 .    .        .            .          .
 .    .        .            .             .
xN  p[xN]  p[xN-1,xN]  p[xN-2,xN-1,xN]  ...  p[x0,...,xN]
```

Definimos los nodos, el número de nodos N y los valores del polinomio p(x) en tales nodos.

```
nodos=[1 2 3 4];
N=length(nodos);
pnodos=[1, 1, 2, 6];
```

Inicializamos una matriz M, con NaN, donde iremos almacenando las diferencias divididas.

```
M=NaN(N,N+1)
% Rellenamos la primera columna
M(:,1)=nodos
% Rellenamos la segunda columna (son las diferencias divididas de orden 0)
M(:,2)=pnodos
```

```
M =

   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN


M =

     1   NaN   NaN   NaN   NaN
     2   NaN   NaN   NaN   NaN
     3   NaN   NaN   NaN   NaN
     4   NaN   NaN   NaN   NaN


M =

     1     1   NaN   NaN   NaN
     2     1   NaN   NaN   NaN
```

```
     3     2   NaN   NaN   NaN
     4     6   NaN   NaN   NaN
```

Calcularemos las diferencias divididas de orden i=1, 2, ..., N-1 que colocaremos en la columna i+2 de la matriz.

```
for j=3:N+1
    for i=j-1:N
     % Las diferencias divididas de orden i (situadas en la columna i+2)
     % comienzan en la fila i+1 y acaban en la fila N.

%
% Reproducimos la parte de la matriz M necesaria para hallar el elemento
% M(i,j)
%
%  M(i-j+2,1)  M(i-j+2,2)
%                         .
%                            .
%                               .
%                                 M(i-1,j-1)
%  M(i,1)     .    .   .   .   .   M(i,j-1)    M(i,j)
%
% Puede comprobarse que siguiendo el elemento M(i-j+2,2) en diagonal (esto
% es, sumándole j-2 al indice de las filas y de las columnas) se llega al
% elemento M(i,j) .

    M(i,j)=(M(i,j-1) - M(i-1,j-1)) / (M(i,1)   - M(i-j+2,1)  );
    end
end
```

La matriz M contiene la tabla de diferencias divididas (Observemos que las diferencias divididas que se utilizan para escribir la forma de Newton del polinomio de Lagrange p(x) son los coeficientes M(i,i+1) de la matriz M con i=1,..N).

```
M
```

```
M =

    1.0000    1.0000       NaN       NaN       NaN
    2.0000    1.0000         0       NaN       NaN
    3.0000    2.0000    1.0000    0.5000       NaN
    4.0000    6.0000    4.0000    1.5000    0.3333
```

Ej 3. Construya la tabla de diferencias divididas correspondiente al polinomio p(x) que interpola la función coseno en los nodos 0, 1 ,2, 3, 4. Localice el valor de p[0, 1 ,2] en la matriz construida. (Sol. Diferencias divididas = 1.0000 , -0.4597 , -0.2484 , 0.1466 , -0.0147) (Ref: Numerical methods using Matlab. J. H. Mathews y K. D. Fink.)

Forma de Newton del polinomio de interpolación de Lagrange

Con las diferencias divididas calculadas podemos construir el polinomio de interpolación usando la forma de Newton, donde xN-1 es el nodo x(N-1): P(x) =p[x0] + p[x0,x1]*(x-x0) + p[x0,x1,x2]*(x-x0)*(x-x1) + ... + p[x0,x1,...,xN]*(x-x0)*(x-x1)* ... *(x-xN-1) .

En primer lugar veamos cómo construir el polinomio q(x)=(x-x0)*(x-x1)* ... *(x-xN) a partir de los valores x=[x0, x1, x2, ..., xN]. Para ello, usaremos un acumulador de producto, que comenzará valiendo el polinomio constante [1], y le iremos multiplicando los monomios (x-xj) que se expresan en forma de vectores como [1, -xj].

```
q=[1]
for i=2:N
monomio=[1,  -nodos(i-1)]
q=conv(q,monomio)
end
```

```
q =

     1


monomio =

     1    -1


q =

     1    -1


monomio =
```

```
     1    -2

q =

     1    -3     2

monomio =

     1    -3

q =

     1    -6    11    -6
```

El polinomio p(x) se obtiene, usando la forma de Newton, mediante una suma de polinomios cuyos sumandos son los polinomios q (calculados como en el bucle anterior) multiplicados por la correspondiente diferencia dividida. Observe en el siguiente bucle cómo se calcula el polinomio p(x) mediante el proceso descrito utilizando un acumulador de suma:

```
q=[1]
newton=[pnodos(1)]
for i=2:N
monomio=[1,  -nodos(i-1)]
q=conv(q,monomio)
newton=[0, newton] + M(i,i+1)*q
end
```

```
q =

     1

newton =

     1

monomio =

     1    -1

q =

     1    -1

newton =

     0     1

monomio =

     1    -2

q =

     1    -3     2

newton =

    0.5000   -1.5000    2.0000

monomio =

     1    -3

q =

     1    -6    11    -6

newton =
```

```
    0.3333   -1.5000    2.1667         0
```

Ej 4. Escribe el polinomio p(x) que se acaba de obtener. ¿Respecto de qué base está expresado? Expresa p(x) respecto de la base de Newton del problema.

Ej 5. ¿Por qué se necesita un 0 en el sumando [0, newton]? (Observe los grados de los polinomios que se suman en la forma de Newton)

Ej 6. Si nodos es [x0, x1, ..., xN] ¿para i=2,..,N qué nodos son nodos(i-1) y qué diferencias divididas son M(i,i+1) ? (Piense el caso i=2, luego el caso i=3, ...)

Ej 7. Se quiere aproximar la función seno en el intervalo [0 , 3.14] mediante un polinomio de grado 6. i) Halle la tabla de diferencia divididas correspondiente a los nodos 0, 0.5, 1, 1.5, 2, 2.5, 3 y a los valores de la función seno. (Sol. Diferencias divididas = 0 , 0.9589 , -0.2348 , -0.1182 , 0.0336, 0.0025, -0.0013) ii) Halle el polinomio de interpolación correspondiente a los datos descritos usando la forma de Newton. iii) Represente conjuntamente los datos de interpolación (en rojo) y el polinomio obtenido (en azul). iv) Represente conjuntamente el polinomio obtenido (en azul) y la función seno (en rojo).

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
22-Mar-2017
```

Published with MATLAB® R2013a

Práctica 9. Mínimos cuadrados

En esta práctica veremos cómo hallar con MatLab, de dos formas distintas, la recta y la parábola que ajustan por mínimos cuadrados un conjunto de puntos del plano. La primera forma consiste en hallar el sistema formado por las ecuaciones normales del ajuste y resolverlo. La segunda, consiste en utilizar 'la división izquierda' de MatLab sobre un sistema A*X=B en el que A no es una matriz cuadrada.

Contents

Ajuste lineal por mínimos cuadrados

Dado un conjunto D de m puntos del plano (x1, y1), (x2, y2),..., (xm, ym) (o nube de puntos) con m>2, queremos hallar una recta y = a0+a1*x que 'pase lo más cerca posible' de esos puntos. Más concretamente, queremos hallar valores a0 y a1 que hagan que a0+a1*xi esté cerca de yi con i=1,..m, en el sentido de que la suma con i=1,...m de los valores (a0+a1*xi - yi)^2 sea la mínima posible. Tales valores a0 y a1 existen y son únicos y, por tanto, determinan una única recta en el plano. A dicha recta se la conoce como recta de mínimos cuadrados que ajusta D.

Para encontrar los valores a0 y a1 que determinan la recta y = a0 + a1 x de mínimos cuadrados que ajusta D hay que resolver un sistema de ecuaciones lineales compatible determinado que es el sistema formado por las ecuaciones normales de este ajuste (véanse los apuntes de teoría). Dicho sistema se puede escribir del siguiente modo (compruébalo):

```
M*transpose(M)*C = M*Y
```

donde:

```
M=[[1,...,1];[x1,...,xm]]
```

```
transpose(M) representa en MatLab la matriz transpuesta de la matriz M
```

```
C=[a0;a1] ,
```

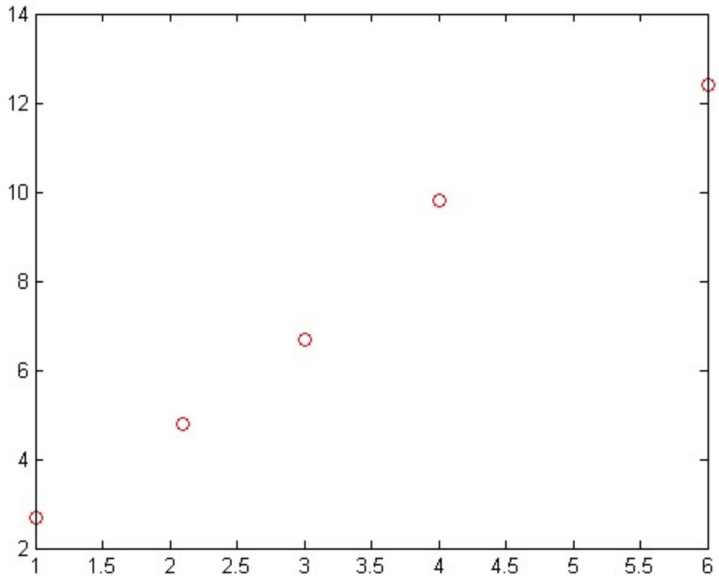
```
Y=[y1;...;ym].
```

Veamos un ejemplo. Nos planteamos hallar la recta de mínimos cuadrados que ajusta los puntos (1, 2.7) , (2.1, 4.8) , (3 , 6.7) , (4 , 9.8) y (6 , 12.4).

Comenzamos representando gráficamente los puntos que hay que ajustar.

```
x=[1  2.1  3  4  6]
y=[2.7 4.8 6.7 9.8 12.4]
plot(x,y,'or')
```

```
x =

    1.0000    2.1000    3.0000    4.0000    6.0000


y =

    2.7000    4.8000    6.7000    9.8000   12.4000
```



Hallamos el sistema de las ecuaciones normales del ajuste lineal por mínimos cuadrados de dichos puntos (su matriz de coeficientes y su vector de términos independientes).

```
m=length(x);
M=[ones(1,m);x];
disp('Matriz de coeficientes del sistema de las ecuaciones normales del ajuste :')
disp(M*transpose(M))
Y=transpose(y);
disp('Vector de términos independientes del sistema de las ecuaciones normales del ajuste :')
disp(M*Y)
```

```
Matriz de coeficientes del sistema de las ecuaciones normales del ajuste :
    5.0000   16.1000
   16.1000   66.4100

Vector de términos independientes del sistema de las ecuaciones normales del ajuste :
   36.4000
  146.4800
```

Resolvemos el sistema de las ecuaciones normales usando, por comodidad, la división izquierda (antes comprobamos que la matriz de coeficientes de dicho sistema es regular). Las componentes del vector solución de dicho sistema son los coeficientes de la recta de mínimos cuadrados que ajusta los puntos dados.

```
det(M*transpose(M))
% Resolvemos el sistema de las ecuaciones normales usando la división izquierda
C=(M*transpose(M))\(M*Y)
disp('Los coeficientes de la recta y=a0+a1*x son : ')
a0=C(1)
a1=C(2)
```

```
ans =

   72.8400


C =

    0.8099
    2.0093

Los coeficientes de la recta y=a0+a1*x son :

a0 =

    0.8099


a1 =

    2.0093
```

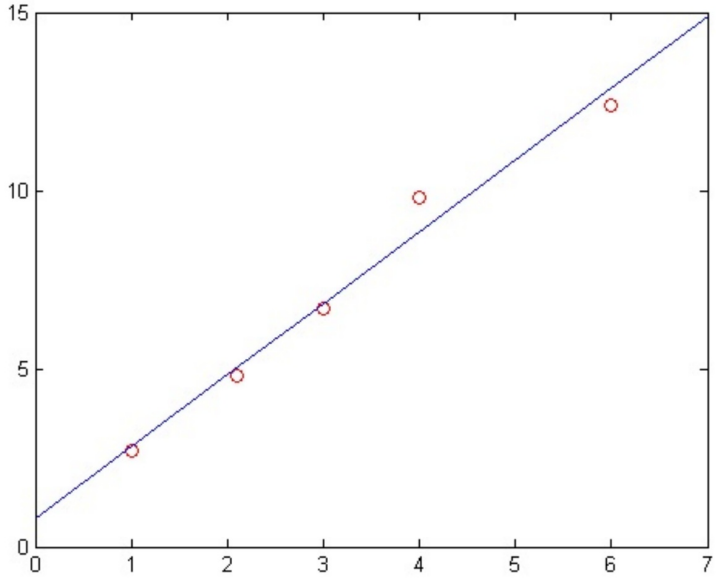
Hallamos el número de condición de la matriz de coeficientes del sistema con la norma 2 (la matriz de coeficientes del sistema de las ecuaciones normales del ajuste puede estar mal condicionada, lo que puede provocar que al resolver dicho sistema con el ordenador el vector solución, C, que se obtiene se vea distorsionado por los errores de redondeo).

```
cond(M*transpose(M),2)
```

```
ans =

   67.9934
```

Por último, representamos gráficamente los puntos (en rojo) y la recta de mínimos cuadrados que los ajusta (en azul).

```
xd=[0:7];
yd=a0+a1*xd;
plot(x,y,'or',xd,yd,'b')
```



Con MatLab también se puede hallar la recta de mínimos cuadrados y = a0+a1*x que ajusta los puntos (x1, y1), (x2, y2),..., (xm, ym), con m>2, usando la 'división izquierda' sobre el sistema A*X=B , donde:

A=[[x1 1];[x2 1];...;[xm 1]]

X=[a1;a0] ,

B=[y1;...;ym].

Observe que la matriz A no es una matriz cuadrada, y que si los puntos (x1, y1), (x2, y2),..., (xm, ym) no están alineados entonces el sistema A*X=B es incompatible.

Las componentes del vector X que se obtiene al usar la 'división izquierda' sobre A*X=B son los coeficientes de la recta de mínimos cuadrados que ajusta los puntos dados (observe que éstos aparecen en orden inverso a como aparecían en el vector C que se obtuvo al resolver el sistema de las ecuaciones normales del ajuste).

Con este modo de hallar la recta de mínimos cuadrados y = a0+a1*x no se obtiene el sistema de las ecuaciones normales del ajuste.

A continuación aparece el código de MatLab correspondiente a esta segunda forma de hallar la recta de mínimos cuadrados y = a0+a1*x que ajusta los puntos (x1, y1), (x2, y2),..., (xm, ym), con m>2 :

```
x=[1  2.1  3  4  6];
y=[2.7 4.8 6.7 9.8 12.4];
m=length(x);
A=[transpose(x),ones(m,1)]
B=transpose(y)
X=A\B
disp('Los coeficientes de la recta y=a0+a1*x son : ')
a0=X(2)
a1=X(1)
```

```
A =

    1.0000    1.0000
```

```
    2.1000    1.0000
    3.0000    1.0000
    4.0000    1.0000
    6.0000    1.0000


B =

    2.7000
    4.8000
    6.7000
    9.8000
   12.4000


X =

    2.0093
    0.8099

Los coeficientes de la recta y=a0+a1*x son :

a0 =

    0.8099


a1 =

    2.0093
```

Ej 1. En la oxidación del dióxido de azufre se obtiene trióxido de azufre. La siguiente tabla muestra los valores de Kep (constante de equilibrio termodinámica para gases) de la reacción

```
2 SO2 + O2 <--> 2 SO3
```

para distintas temperaturas en grados Kelvin. (Tabla 20.3, Química General, Petrucci)

```
T      800    850    900    950  1000  1050    1100    1170
Kep   9.1e2  1.7e2  4.2e1  1.0e1   3.2   1.0  3.9e-1  1.2e-1
```

i) Represente gráficamente en color rojo los datos de la tabla, considerando las temperaturas en el eje de abscisas y los valores de Kep en el eje de ordenadas.

ii) Defina un vector x cuyas componentes son los inversos de los valores de las temperaturas y un vector y cuyas componentes son los logaritmos neperianos de los valores de las constantes de equilibrio. Represente gráficamente en color azul el conjunto D de los puntos del plano cuyas abscisas y ordenadas son, respectivamente, las componentes de los vectores x e y.

iii) Halle la recta que ajusta por mínimos cuadrados D. (Sol. y= - 21.5809 + 2.2722e+004 * x)

iv) Halle las ecuaciones normales del ajuste del apartado anterior (antes de calcularlas ejecute primero el comando format long y cuando termine de resolver el apartado vuelva al formato inicial ejecutando format short). (Sol. Matriz de coeficientes= [[8.000000000000000 0.008306385994466]; [0.008306385994466 0.000008753173498]] Vector de términos independientes=[16.090776481585713;0.019630613572036])

v) Sabiendo que la constante de equilibrio termodinámica (Keq), la entalpía de la reacción (VH) y la variación de la entalpía (VS) cumplen la siguiente ecuación

$$\ln Keq = \frac{-VH}{R} * \frac{1}{T} + \frac{VS}{R}$$

donde R es la constante de los gases, R= 8.3145 J /(mol * K), determine la entalpía de la reacción suponiendo que x=1/T, e y=ln Keq. (Sol. Entalpía de la reacción = -1.8892e+005 J/mol) .

```
T = [800  850  900  950  1000  1050  1100  1170];
Kep=[9.1e2  1.7e2  4.2e1   1.0e1   3.2   1.0   3.9e-1   1.2e-1] ;
```

Ajuste parabólico por mínimos cuadrados

Dado un conjunto D de m puntos del plano (x1, y1), (x2, y2),..., (xm, ym) (o nube de puntos) con m>3, queremos hallar una parábola y = a0+a1*x+a2*x^2 que 'pase lo más cerca posible' de esos puntos. Más concretamente, queremos encontrar valores a0, a1 y a2 que hagan que a0+a1*xi+a2*xi^2 esté cerca de yi con i=1,..m, en el sentido de que la suma con i=1,...m de los valores (a0+a1*xi+a2*xi^2 - yi)^2 sea la mínima posible. Tales valores a0, a1 y a2 existen y son únicos y, por tanto, determinan una única parábola en el plano. A dicha parábola se la conoce como parábola de mínimos cuadrados que ajusta D.

Para encontrar los valores a0, a1 y a2 que determinan la parábola y = a0+a1*x+a2*x^2 de mínimos cuadrados que ajusta D hay que resolver un sistema de ecuaciones lineales compatible determinado que es el sistema formado por las ecuaciones normales de este ajuste (véanse los apuntes de teoría). Dicho sistema se puede escribir del siguiente modo (compruébalo):

```
M*transpose(M)*C = M*Y
```

donde:

M=[[1,...,1];[x1,...,xm];[x1^2,x2^2,...,xm^2]]

transpose(M) representa en MatLab la matriz transpuesta de la matriz M

C=[a0;a1;a2] ,

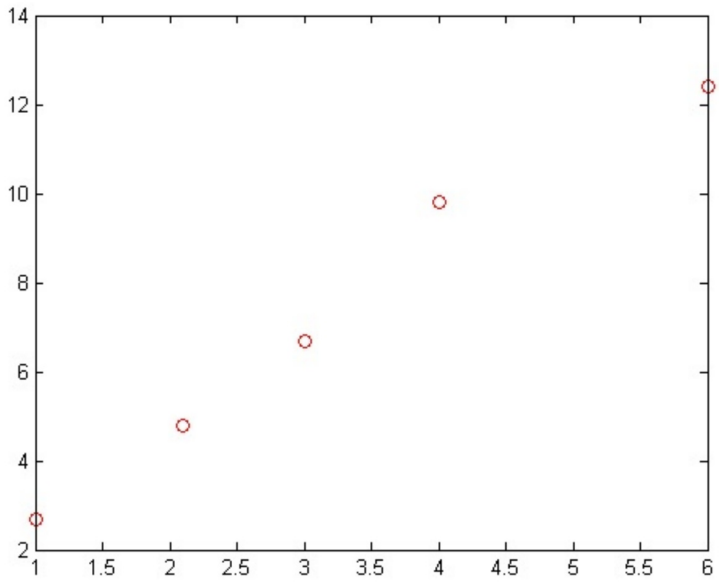
Y=[y1;...;ym].

Veamos un ejemplo. Nos planteamos hallar la parábola de mínimos cuadrados que ajusta los puntos (1, 2.7) , (2.1, 4.8) , (3 , 6.7) , (4 , 9.8) y (6 , 12.4).

Recordamos la representación gráfica de los puntos que hay que ajustar (ya se vió en el apartado anterior de la práctica).

```
x=[1  2.1  3  4  6]
y=[2.7 4.8 6.7 9.8 12.4]
plot(x,y,'or')
```

```
x =

    1.0000    2.1000    3.0000    4.0000    6.0000


y =

    2.7000    4.8000    6.7000    9.8000   12.4000
```



Hallamos el sistema de las ecuaciones normales del ajuste parabólico por mínimos cuadrados de dichos puntos (su matriz de coeficientes y su vector de términos independientes).

```
m=length(x);
M=[ones(1,m);x;x.^2];
disp('Matriz de coeficientes del sistema de las ecuaciones normales del ajuste :')
disp(M*transpose(M))
Y=transpose(y);
disp('Vector de términos independientes del sistema de las ecuaciones normales del ajuste :')
disp(M*Y)
```

```
Matriz de coeficientes del sistema de las ecuaciones normales del ajuste :
   1.0e+03 *

    0.0050    0.0161    0.0664
    0.0161    0.0664    0.3173
    0.0664    0.3173    1.6534

Vector de términos independientes del sistema de las ecuaciones normales del ajuste :
   36.4000
  146.4800
  687.3680
```

Resolvemos el sistema de las ecuaciones normales usando, por comodidad, la división izquierda (antes comprobamos que la matriz de coeficientes de dicho sistema es regular). Las componentes del vector solución de dicho sistema son los coeficientes de la parábola de mínimos cuadrados que ajusta los puntos dados.

```
det(M*transpose(M))
% Resolvemos el sistema de las ecuaciones normales usando la división izquierda
C=(M*transpose(M))\(M*Y)
disp('Los coeficientes de la parábola y=a0+a1*x+a2*x^2 son : ')
a0=C(1)
a1=C(2)
a2=C(3)
```

```
ans =

   2.7088e+03


C =

   -0.1955
    2.7546
   -0.1050

Los coeficientes de la parábola y=a0+a1*x+a2*x^2 son :

a0 =

   -0.1955


a1 =

    2.7546


a2 =

   -0.1050
```

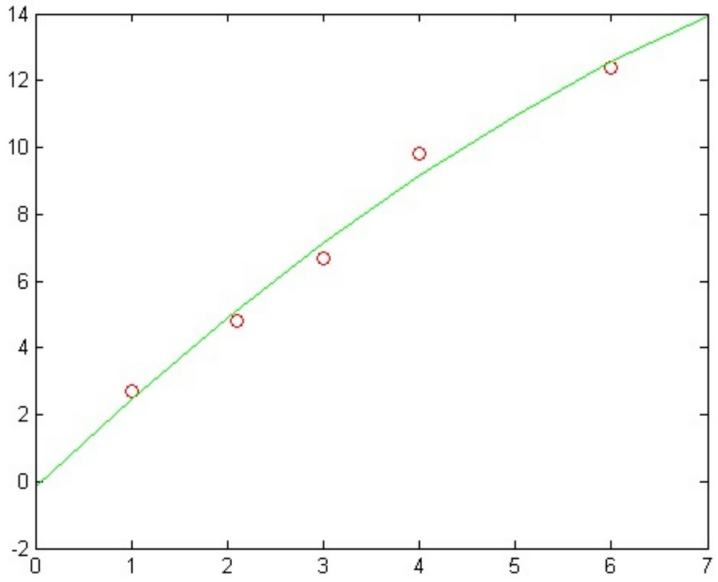
Hallamos el número de condición de la matriz de coeficientes del sistema con la norma 2 (la matriz de coeficientes del sistema de las ecuaciones normales del ajuste puede estar mal condicionada, lo que puede provocar que al resolver dicho sistema con el ordenador el vector solución, C, que se obtiene se vea distorsionado por los errores de redondeo).

```
cond(M*transpose(M),2)
```

```
ans =

   8.0597e+03
```

Por último, representamos gráficamente los puntos (en rojo) y la parábola de mínimos cuadrados que los ajusta (en verde).

```
xd=[0:7];
yd=a0+a1*xd+a2*xd.^2;
plot(x,y,'or',xd,yd,'g')
```



Con MatLab también se puede hallar la parábola de mínimos cuadrados y = a0+a1*x+a2*x^2 que ajusta los puntos (x1, y1), (x2, y2),..., (xm, ym), con m>2, usando la 'división izquierda' sobre el sistema A*X=B , donde:

`A=[[x1^2  x1  1];[x2^2  x2  1];...;[xm^2  xm  1]]`

`X=[a2;a1;a0] ,`

`B=[y1;...;ym].`

Observe que la matriz A no es una matriz cuadrada, y que si los puntos (x1, y1), (x2, y2),..., (xm, ym) no están situados sobre una parábola entonces el sistema A*X=B es incompatible.

Las componentes del vector X que se obtiene al usar la 'división izquierda' sobre A*X=B son los coeficientes de la parábola de mínimos cuadrados que ajusta los puntos dados (observe que éstos aparecen en orden inverso a como aparecían en el vector C que se obtuvo al resolver el sistema de las ecuaciones normales del ajuste).

Con este modo de hallar la parábola de mínimos cuadrados y=a0+a1*x+a2*x^2 no se obtiene el sistema de las ecuaciones normales del ajuste.

A continuación aparece el código de MatLab correspondiente a esta segunda forma de hallar la parábola de mínimos cuadrados y = a0+a1*x+a2*x^2 que ajusta los puntos (x1, y1), (x2, y2),..., (xm, ym), con m>3 :

```
x=[1  2.1  3  4  6];
y=[2.7 4.8 6.7 9.8 12.4];
m=length(x);
A=[transpose(x.^2),transpose(x),ones(m,1)]
B=transpose(y)
X=A\B
disp('Los coeficientes de la parábola y=a0+a1*x+a2*x^2 son : ')
a0=X(3)
a1=X(2)
a2=X(1)
```

```
A =

    1.0000    1.0000    1.0000
    4.4100    2.1000    1.0000
    9.0000    3.0000    1.0000
   16.0000    4.0000    1.0000
   36.0000    6.0000    1.0000


B =

    2.7000
    4.8000
    6.7000
    9.8000
   12.4000


X =

   -0.1050
    2.7546
   -0.1955
```

```
Los coeficientes de la parábola y=a0+a1*x+a2*x^2 son :

a0 =

   -0.1955


a1 =

    2.7546


a2 =

   -0.1050
```

Ej 2. La siguiente tabla muestra el número atómico y el radio atómico en picómetros de los metales de la primera serie de transición, salvo el Vanadio (Tabla 24.1, Química General, Petrucci).

Elemento	Sc	Ti	Cr	Mn	Fe	Co	Ni	Cu	Zn
Número atómico	21	22	24	25	26	27	28	29	30
Radio metálico	161	145	125	124	124	125	128	128	133

i) Halle la parábola que ajusta por mínimos cuadrados los puntos del plano que se obtienen tomando en el eje de abscisas los números atómicos y en el de ordenadas los radios metálicos. (Sol. y = 934.1174 - 61.0538*x + 1.1477*x^2)

ii) Represente gráficamente en el intervalo [20,32] los puntos del apartado anterior (en azul) junto con la parábola de mínimos cuadrados que los ajusta (en verde).

iii) Suponga que tal parábola puede usarse para estimar el radio atómico del Vanadio (número atómico 23). ¿Qué error relativo se estaría cometiendo al considerar dicha estimación si el radio atómico del Vanadio es 132? (Sol. Radio estimado = 137.0093, error relativo = -0.0379).

```
x=[  21  22  24  25  26  27  28  29  30];
y=[ 161 145 125 124 124 125 128 128 133];

% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
19-Apr-2017
```

Published with MATLAB® R2013a

Práctica 10. Derivación e integración numéricas.

Aunque Matlab cuenta con comandos para realizar derivadas e integrales simbólicamente, en ocasiones no se conoce la expresión de la función a derivar o integrar y sólo se conocen algunos valores de dicha función. En otras ocasiones, aunque se dispone de la expresión de la función, resulta costoso hallar su función derivada o una primitiva de la función. Incluso, a veces, o no existe una primitiva de la función o bien la primitiva de la función no puede expresarse usando funciones elementales.

Contents

En esta práctica veremos algunas de las fórmulas que permiten aproximar el valor de la derivada de una función en un punto (fórmulas de derivación numérica) y el valor de la integral definida de una función (fórmulas de integración numérica).

Las fórmulas de derivación y las de integración numérica suelen requerir los valores de la función a derivar o a integrar en varios puntos. Estos valores pueden estar recogidos en un vector o hallarse evaluando la función.

Derivación numérica.

La fórmula de diferencia centrada en dos nodos permite aproximar la derivada de una función en un punto a partir de dos evaluaciones de dicha función.

```
difc=@(x,h) (f(x+h)-f(x-h))/(2*h)
```

```
difc =

    @(x,h)(f(x+h)-f(x-h))/(2*h)
```

Ej 1. Emplee la fórmula de diferencia centrada en dos nodos para aproximar la derivada de la función f(x)=sqrt(1-x^2) en el punto x=sqrt(2)/2 para h=0.01 (Sol. -1.0002)

Ej 2. El script malp.m usa la fórmula de diferencia centrada en dos nodos para aproximar la derivada de f(x)=sqrt(1-x^2) en el punto x=sqrt(2)/2 para distintos valores h=10^(-i). Modifique el programa para que pueda representarse en el eje horizontal los valores de i y en el eje vertical el valor absoluto del error absoluto cometido. Vuelva a modificar el programa para que en el eje vertical se muestre log10(abs(y)) donde y=(-1)-derivadaap es el error absoluto cometido. Observe que un valor de h más pequeño no da necesariamente menor error. De los valores h empleados, ¿cuáles dan el error más pequeño? (Sol. h=10^(-6), h=10^(-7).)

Ej 3. Modifique el script malp.m para aproximar el valor de la derivada de f(x)=exp(x) en el punto x=0. Sabiendo que el resultado exacto es f '(0)=exp(0)=1 ¿qué valor de h da el error más pequeño? (Cap 6. Diez lecciones de cálculo numérico. Sanz-Serna) (Sol. h=10^(-5))

Ej 4. Para aproximar f'(c) se considera la fórmula de derivación numérica en cinco nodos siguiente:

$$\frac{f(x-2h)}{12h} - \frac{2f(x-h)}{3h} + \frac{2f(x+h)}{3h} - \frac{f(x+2h)}{12h}$$

Implemente dicha fórmula. Use la fórmula para hallar la derivada de la función f(x)=exp(x) en x=1 para h=10^(-3). Sabiendo que el valor exacto de la derivada es exp(1) ¿qué error se comete al considerar el valor que da la fórmula de cinco nodos anterior? (Sol. 5.0182e-013)

Integración numérica.

A continuación vamos a trabajar con las fórmulas del trapecio y de Simpson y sus respectivas fórmulas compuestas.

Fórmula del trapecio

La fórmula del trapecio aproxima el valor de la integral definida entre x=a y x=b por el área del trapecio determinado por el eje OX, la recta x=a, la recta x=b y la recta que une los puntos (a,f(a)) y (b,f(b)).

```
f=@(x) x.^2+x+2;
a=2;
b=3;
trapecio=(b-a)*(f(a)+f(b))/2
```

```
trapecio =

    11
```

Fórmula del trapecio compuesta

Si dividimos un intervalo [a,b] en n intervalos de longitud h=(b-a)/n con nodos xi=a+i*h, podemos aplicar a cada uno de esos intervalos la fórmula del trapecio y obtener la fórmula del trapecio compuesta (h/2)*(f(a) + 2*f(a+h) + 2*f(a+2h) + 2*f(a+3h) + ... + 2*f(a+(n-1)h) + f(b))

En la función trapecioc.m se encuentra una implementación del método.

Ej 5. Incluya en la función trapecioc.m un comentario en la segunda linea indicando el objetivo de la función y otros comentarios explicando el significado de las variables a, b y n.

Ej 6. Usando la fórmula del trapecio compuesta, aproxime el valor de la integral definida entre x=-1 y x=1 de la función f(x)=sqrt(1-x^2) usando n=2, n=100 y n=1000 nodos. ¿El área de qué figura se está calculando cuando n=2? Sabiendo que el valor exacto de la integral es pi/2, halle el error cometido en las aproximaciones obtenidas. (Sol: 1, 1.5691, 1.5707. Errores absolutos 0.5708, 0.0017, 5.2588e-005)

Fórmula de Simpson

En la fórmula de Simpson, el valor de la integral definida se aproxima mediante la integral definida de una parábola que interpola la función integrando en tres puntos.

```
f=@(x) x.^2+x+2;
a=-1;
b=1;
simpson=(b-a)*(f(a)+ 4*f( (a+b)/2 )+f(b))/6
```

```
simpson =

    4.6667
```

Ej 7. Defina una función anónima con argumentos a y b que aproxime la integral de una función f (definida previamente) entre x=a y x=b usando la fórmula de Simpson. Use dicha fórmula para aproximar la integral definida de cos(x) entre 0 y pi/2. (Sol. 1.0023)

Fórmula de Simpson compuesta.

Si dividimos un intervalo [a,b] en n intervalos de longitud h=(b-a)/n con nodos xi=a+i*h, podemos aplicar a cada uno de esos intervalos la fórmula de Simpson y obtener la fórmula de Simpson compuesta.

Ej 8. Para h=(b-a)/n y xi=a+i*h, implemente la fórmula de Simpson compuesta. (Puede inicializar un acumulador como s=h*f(a)/6, y vaya sumando 2*h*f(a+(i-1/2)*h)/3 + h*f(a+i*h)/3 para i=1,2,..., n-1; para completar el último intervalo, sume 2*h*f(a+(n-1/2)*h)/3 + h*f(b)/6. Compruebe la fórmula aproximando el valor de la integral de sen(x) entre 0 y pi considerando n=10. (Sol. 2.0000)

Ej 9. Usando la fórmula de integración del trapecio compuesta aproxime la integral de la función x(t)=t^3-t^2+t+2 entre t=0 y t=2.2 considerando n=10. Aproxime la misma integral mediante la fórmula de Simpson compuesta para n=5. Sabiendo que el valor exacto de la integral es 68453/7500, ¿cuál de los dos resultados se ha aproximado más al valor exacto? (Sol. Trapecio compuesta = 9.1679. Simpson compuesta = 9.1271.)

Ej 10. Para hallar el trabajo de expansión irreversible isotérmica de un mol de dióxido de carbono se necesita hallar el valor de la integral de la función P(V)=R*T/(V-b)-a/V^2 desde V1=0.0729762 litros hasta V2=26.4115 litros. Considere a=3.5924 L^2*atm/mol^2, b=0.04267 L/mol, R=0.08205746 (atm*L)/(mol*K) y T=323.15 K . (Sol. aprox. 130.3907 L*atm (usando Simpson compuesta con 5000 subintervalos))

Derivación e integración numéricas mediante aproximación.

Otra manera de hallar el valor de la derivada de una función en un punto (o el valor de la integral definida de una función) a partir de los valores de dicha función consiste en hallar un polinomio que aproxime tales valores y evaluar en dicho punto la derivada de ese polinomio (o hallar la integral definida de dicho polinomio).

En el siguiente ejercicio se aproxima una derivada a partir de un ajuste parabólico de unos datos.

Ej 11. Si f(t) es la concentración de peróxido de hidrógeno (H2O2) en función del tiempo, la derivada -df(t)/dt es la velocidad de la reacción de descomposición del peróxido de hidrógeno

H2O2 ---> H2O + (1/2) O2

Se quiere estimar dicha velocidad, sin conocer el orden de la reacción, a partir de un ajuste parabólico de los siguientes datos (Tabla 15.1, Química General, Petrucci)

tiempo en segundos 0 200 400 600 1200 1800 3000

[H2O2] en moles/litro 2.32 2.01 1.72 1.49 0.98 0.62 0.25

```
x=[   0  200  400  600 1200 1800 3000];
y=[2.32 2.01 1.72 1.49 0.98 0.62 0.25];
```

i) Represente los datos de la tabla considerando los tiempos en el eje de abscisas y las concentraciones en el eje de ordenadas.

ii) Halle un ajuste parabólico de los valores de y respecto a los de x, y represente la parábola obtenida junto con los datos mencionados en i). (Sol: 2.2828e-007*x^2 - 0.0014*x + 2.2683)

iii) Derive (en papel) la parábola obtenida. Evalúe la expresión de dicha derivada en x=0 y en x=1400 para hallar la velocidad de reacción para t=0 y para t=1400. (Observe que aunque en la gráfica de la parábola obtenida la pendiente de la tangente a cualquier punto es negativa, la velocidad de reacción, por definición, es positiva) (Sol: Para t=0, velocidad=13.5*10^(-4) Moles/segundo. Para t=1400, velocidad=7.13*10^(-4) Moles/segundo.)

iv) Represente el vector x en el eje horizontal y el vector log(y) en el eje vertical. (Observe que la gráfica es similar a una recta) (Sabiendo que la reacción es de primer

orden, se cumple que ln f(t) = -k*t + f(0), con lo que los datos obtenidos en los apartados ii) y iii) son una aproximación, pero no representan un modelo del proceso.)

```
% Matemáticas II. Grado en Ingeniería Química. (A. Palomares)
disp(date)
```

```
19-Apr-2017
```

Published with MATLAB® R2013a