Práctica 0. Introducción.

Práctica 1. Ecuaciones Diferenciales Ordinarias.

Práctica 2. Ecuaciones en Derivadas Parciales.

Práctica 3. Resolución de ecuaciones (I)

Práctica 4. Resolución de ecuaciones (II)

Práctica 0. Introducción.

Una de las maneras de seguir las prácticas en Matlab 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
- Ejecutar una demo
- Uso de la ayuda
- Orden de las operaciones
- Acumulador (suma)
- Vectorización (suma)
- Gráficas.
- Ejercicios finales.

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 dónde 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.

   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 como '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
```

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 =

  Columns 1 through 11
```

```
     2     4     9     3     6     0     3     2     1     8     6

  Columns 12 through 13

     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 3. 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.

Gráficas.

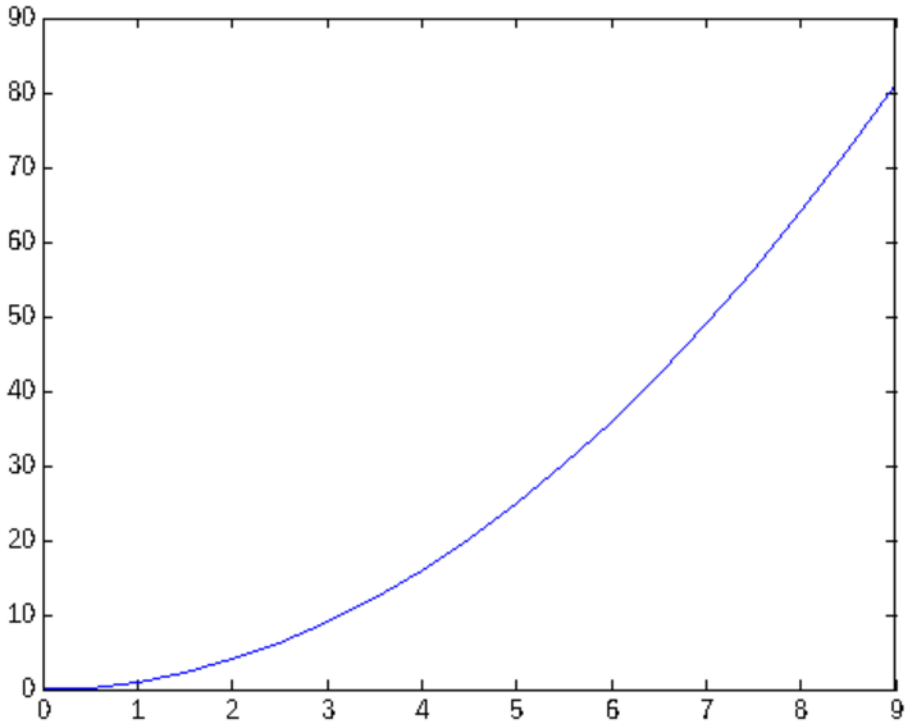
Una gráfica en Matlab puede representarse de forma sencilla usando el comando plot.

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 6

         0    0.5000    1.0000    1.5000    2.0000    2.5000

  Columns 7 through 12

    3.0000    3.5000    4.0000    4.5000    5.0000    5.5000

  Columns 13 through 18

    6.0000    6.5000    7.0000    7.5000    8.0000    8.5000

  Column 19

    9.0000


y =

  Columns 1 through 6

         0    0.2500    1.0000    2.2500    4.0000    6.2500

  Columns 7 through 12

    9.0000   12.2500   16.0000   20.2500   25.0000   30.2500

  Columns 13 through 18

   36.0000   42.2500   49.0000   56.2500   64.0000   72.2500

  Column 19

   81.0000
```

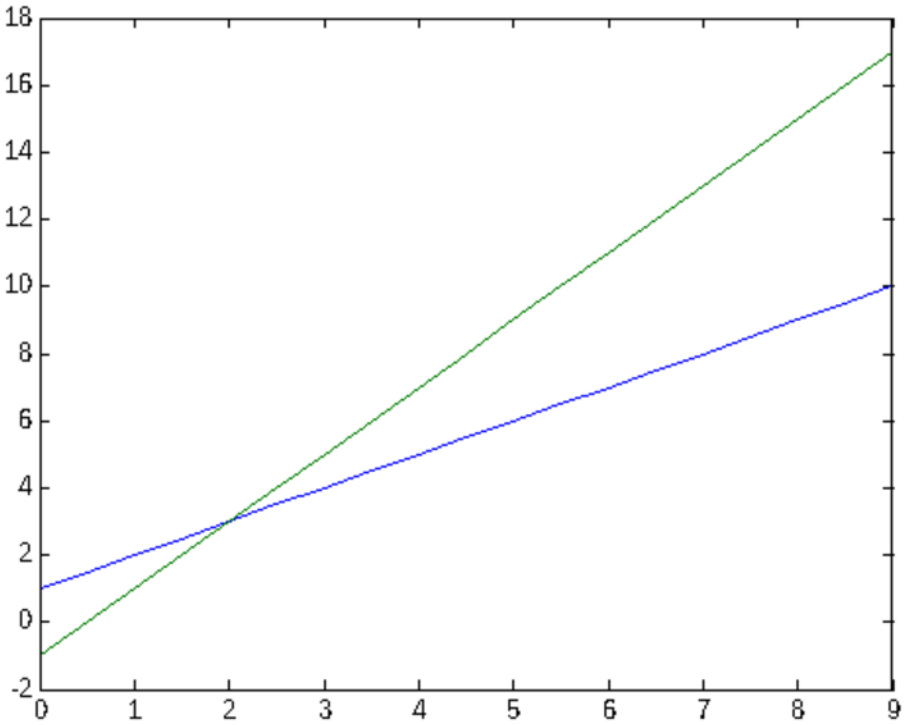


Observe que al definir el vector y estamos aplicando la operación 'elevar al cuadrado' a todas las componentes del vector x, por eso ponemos un punto antes del símbolo de potencia.

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)
```



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 y del polinomio p(x)=x-x^3/6+x^5/120 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 y del polinomio p(x) 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 y del polinomio q(x)=1-x^2/2+x^4/24 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 a-b. (Solución : a=0.2955, b=0.2955, a-b=-4.3339e-008)

Observe que si se muestran a y b con cinco 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 a-b. (Solución : a=0.9211, b=0.9211, a-b= -5.6727e-006).

Al acabar la clase se recomienda que almacene o se envíe los script que haya creado. A continuación puede reiniciar el ordenador, o apagarlo, dependiendo de si hay un turno de prácticas después del suyo, o no.

```
% Ecuaciones Diferenciales y Cálculo Numérico. (A. Palomares)
% Grado en Ingeniería de Tecnologías de Telecomunicación.
disp(date)
```

```
22-Feb-2024
```

Published with MATLAB® 7.5

Práctica 1. Ecuaciones Diferenciales Ordinarias.

Veremos en esta práctica cómo resolver simbólicamente ecuaciones diferenciales usando Wolfram|Alpha y cómo obtener soluciones numéricas mediante el método de Euler.

Contents

- Solución simbólica de una ecuación diferencial
- Solución general de una ecuación diferencial
- Solución particular de un PVI
- Circuito RLC
- Solución numérica un PVI. Método de Euler.
- Tabla de resultados
- Gráfica de la poligonal
- Comparación con los valores exactos
- Tabla de errores
- Gráfica
- Ejercicios finales

Solución simbólica de una ecuación diferencial

Para calcular una solución exacta (analítica) de una ecuación diferencial basta escribirla adecuadamente en la página de Wolfram|Alpha

Solución general de una ecuación diferencial

Para este ejemplo abrimos la página www.wolframalpha.com y escribimos DSolve[y'(x)-2*x*y(x)=0] En los resultados puede ver la clasificación de la ecuación, la solución general en términos de una constante c_1, el campo de pendientes e incluso una muestra de la familia de soluciones en términos del valor de y(0).

Observe que hemos escrito la función incógnita con su variable y(x) . Wolfram|Alpha es flexible con la notación que usamos, a cambio debemos prestar atención a cómo ha interpretado nuestra petición y, en ocasiones, nos puede pedir información adicional.

Ej 1 . Halle la solución general de x*y'(x)-y^2+9=0 . (solución : y(x) = -(3 (e^(6 c_1) x^6 - 1))/(e^(6 c_1) x^6 + 1))

Solución particular de un PVI

De forma similar, incluyendo las condiciones iniciales, se obtiene la solución particular de una problema de valores iniciales. Introduzca la instrucción DSolve[y'(x)-2*x*y(x)==0, y(0)==2]

Vemos que en este caso la solución no incluye la constante c_1 y

obtenemos la gráfica de y(x) .

Ej 2. Halle la solución de x*y'(x)-y^2+9=0 que cumple que y(1)=7 . (solución : y(x) = (6 x^6 + 15)/(5 - 2 x^6)).

Circuito RLC

También es posible resolver ecuaciones de orden superior. Escriba la instrucción DSolve [0.2*i''(t)+330*i'(t)+(1/(20*10^-6))*i(t)=0, i(0)=0, i'(0)=30] donde la función i(t) es la corriente, variable del problema, no es la unidad de los números complejos. La ecuación está tomada de https://youtu.be/w9Z4Ly98Lcw?t=143 donde se resuelve el problema paso a paso.

Solución numérica un PVI. Método de Euler.

Nos planteamos aplicar el método de Euler al siguiente problema de valor inicial y'(x)=f(x,y) para x en el intervalo (x0,xn), y(x0)=y0 .

Comenzamos definiendo dos variables para la condición inicial primero x0 y a continuación y0 .

```
x0=0.0
y0=3.0
```

```
x0 =

     0
```

```
y0 =

     3
```

Para implementar el método de Euler aproximamos el valor de la solución en unos puntos x0, x1, ..., xn también llamados nodos. Definimos ahora el valor de último punto xn, el número de intervalos que definen los puntos n y la distancia entre los puntos h.

```
xn = 1
n = 10
h = (xn - x0)/n
```

```
xn =

     1
```

```
n =
```

```
    10
```

```
h =

    0.1000
```

Ahora necesitamos definir la función f. Veremos más adelante cómo definir funciones en un script, pero ahora lo haremos mediante una función anónima con la siguiente sintaxis con lo que tendremos f(x,y)=2*x*y .

```
f=@(x,y) 2*x*y
```

```
f =

    @(x,y)2*x*y
```

Definimos los puntos x1, x2, ..., xn en un vector, indicando el primer punto, x0, luego la longitud de los intervalos o paso, h, y luego el último punto. Inicializamos Y como un vector de ceros (una fila, n+1 columnas).

```
X=[x0 : h : xn]
Y=zeros(1,n+1)
```

```
X =

  Columns 1 through 6

         0    0.1000    0.2000    0.3000    0.4000    0.5000

  Columns 7 through 11

    0.6000    0.7000    0.8000    0.9000    1.0000


Y =

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

Ahora definimos los valores de los puntos yi . Como ilustración hemos sacado la definición de Y(2) fuera del bucle.

```
Y(1)=y0;
Y(2)=Y(1) + f(X(1), Y(1)) * (X(2) - X(1));

for i= 3:n+1
Y(i)=Y(i-1) + f(X(i-1), Y(i-1)) * (X(i) - X(i-1));
```

```
end
```

Tabla de resultados

Para ver los valores hallados, trasponemos el vector con los puntos de X para que quede como una columna, hacemos lo mismo con Y y mostramos los dos vectores uno junto a otro.

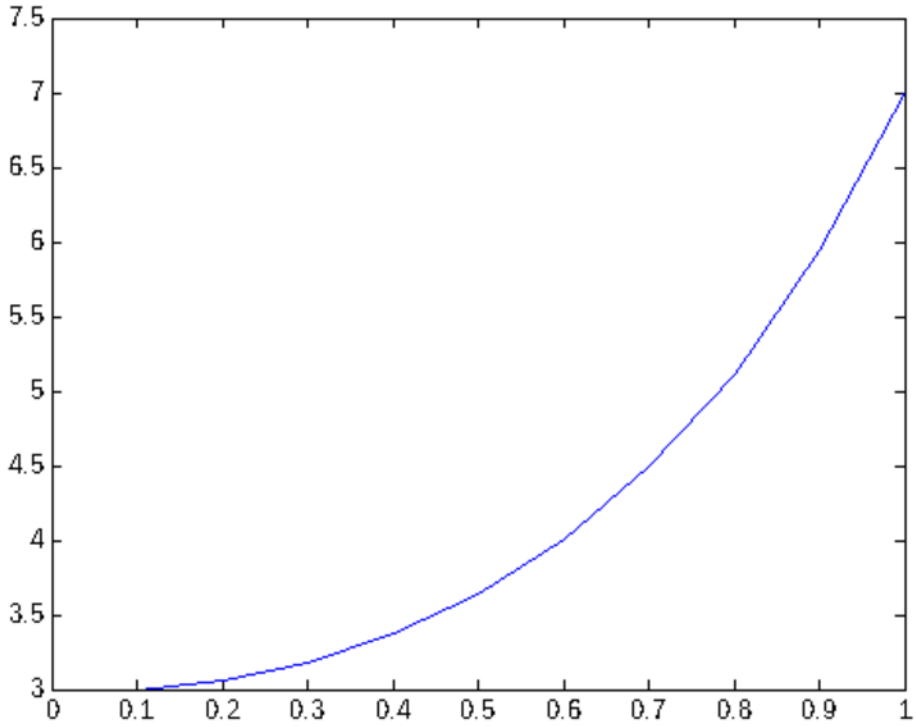
```
[X' , Y']
```

```
ans =

         0    3.0000
    0.1000    3.0000
    0.2000    3.0600
    0.3000    3.1824
    0.4000    3.3733
    0.5000    3.6432
    0.6000    4.0075
    0.7000    4.4884
    0.8000    5.1168
    0.9000    5.9355
    1.0000    7.0039
```

Gráfica de la poligonal

Podemos ver la gráfica de la poligonal formada por los puntos (xi, yi) . (El código del método está en el script MetEuler.m)

```
plot(X,Y)
```



Comparación con los valores exactos

En este caso podemos comparar los resultados obtenidos por el método numérico con los valores exactos porque podemos definir la función exacta (analítica) y evaluarla en los puntos del vector X, para lo cual hemos puesto un punto antes del signo de elevado al definir la función

```
yexacta=@(x) 3*exp(x.^2)
Ye=yexacta(X)


yexacta =

    @(x)3*exp(x.^2)


Ye =

  Columns 1 through 6

    3.0000    3.0302    3.1224    3.2825    3.5205    3.8521

  Columns 7 through 11

    4.3000    4.8969    5.6894    6.7437    8.1548
```

Tabla de errores

Para ver mejor los resultados los mostramos en columnas. La primera columna son los puntos x0, x1, ..., xn. En la segunda columna se ven los valores hallados por el método de Euler. En la tercera columna están los valores exactos y en la cuarta la diferencia entre los valores exactos y los aproximados, con lo que podemos ver los errores que se cometen.

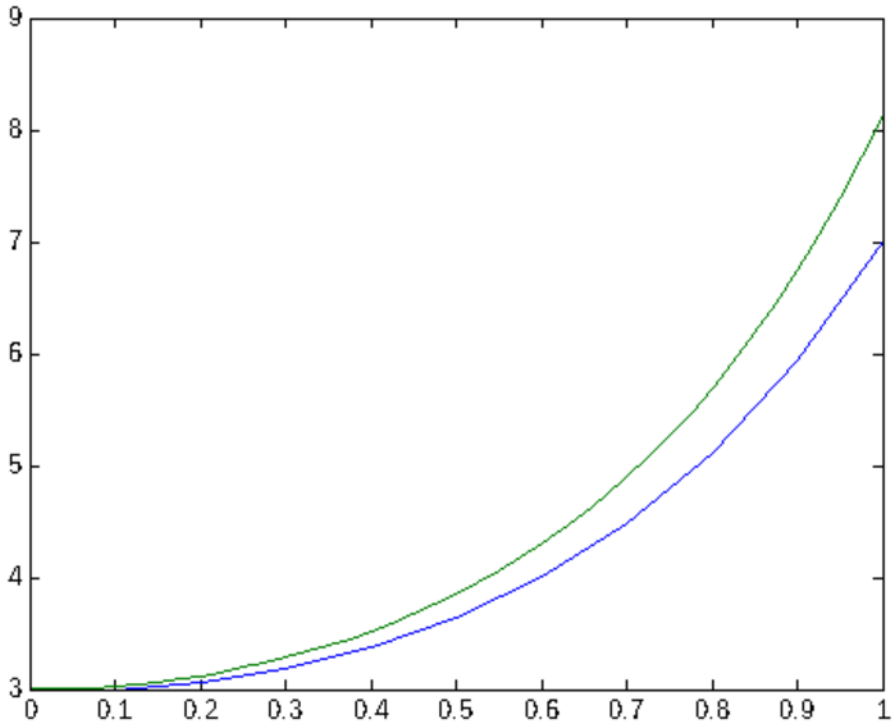
```
[X', Y', Ye', (Ye-Y)']


ans =

         0    3.0000    3.0000         0
    0.1000    3.0000    3.0302    0.0302
    0.2000    3.0600    3.1224    0.0624
    0.3000    3.1824    3.2825    0.1001
    0.4000    3.3733    3.5205    0.1472
    0.5000    3.6432    3.8521    0.2089
    0.6000    4.0075    4.3000    0.2925
    0.7000    4.4884    4.8969    0.4085
    0.8000    5.1168    5.6894    0.5726
    0.9000    5.9355    6.7437    0.8082
    1.0000    7.0039    8.1548    1.1509
```

Vemos que en los primeros nodos los errores son relativamente pequeños 0.0302, 0.0624,... pero que van creciendo conforme nos alejamos de la condición inicial

Gráfica

Para comparar gráficamente los resultados con la solución exacta del problema, definimos un vector XE con más puntos, definimos una función yexacta con la solución exacta del problema 3*e^(x^2), y evaluamos la función en todos los puntos de XE, para lo cual hemos puesto un punto antes del signo de elevado al definir la función.

```
XE=[x0:h/4:xn];
YE=yexacta(XE);
plot(X,Y,XE,YE)
```



Ej 3. En la gráfica se representan dos funciones ¿cuál es la exacta y cuál la aproximada? (Indicación: mire la tabla de errores.)

Ejercicios finales

Ej 4. Se quiere emplear el método de Euler con h = 0.05 para hallar una solución aproximada de y(2) en la misma ecuación diferencial del ejemplo anterior (y'(x)=f(x,y)=x+y, y(0)=3).

a) Halle el número n de partes en las que se debe dividir el intervalo [0, 2] y obtenga dicha solución aproximada.

b) Use el mismo paso h=0.05 para hallar una aproximación a y(3).

Ej 5. Se considera el problema de valor inicial y'(x) = 3y^2 -x^2, y(0)=0.15

a) Halle una aproximación de y(3) usando h=0.1.

b) Halle una aproximación de y(3) usando h=0.05.

c) Suponiendo que y(3) vale -1.673381304457, ¿cuál de los dos valores de h permite obtener la mejor aproximación de y(3)? (solución : Exacto - aproximado=0.0001, Exacto - aproximado=5.418e-05, más pequeño)

Ej 6. Dentro del bucle del método de Euler, ¿se puede sustituir (X(i) - X(i-1)) por h? Justifique la respuesta.

Ej 7. Halle una aproximación de la solución de y'(x)=2y-(3/10)y^2, y(0)=1/10 en x=4 usando el método de Euler en el intervalo (0,4) para n=25. Represente la poligonal del Euler.

Sabiendo que la solución exacta es y(x)= (20 e^(2x)) / (197 + 3e^(2x)) ¿Qué error absoluto se está cometiendo? (Valor exacto menos valor aproximado). (Sol. y(4)=~ 6.5164. Error=0.00653 .)

```
% Ecuaciones Diferenciales y Cálculo Numérico. (A. Palomares)
% Grado en Ingeniería de Tecnologías de Telecomunicación.
disp(date)
```

```
01-Mar-2023
```

Published with MATLAB® 7.5

Práctica 2. Ecuaciones en Derivadas Parciales.

Veremos cómo implementar métodos numéricos basados en diferencias finitas para resolver aproximadamente la ecuación de ondas, la ecuación del calor y la ecuación de Poisson.

Contents

- Método explícito para el problema mixto para la ecuación de ondas
- Condiciones iniciales para la ecuación de ondas.
- Primeros valores de U.
- Bucle general para la ecuación de ondas.
- Resultados para la ecuación de ondas.
- Método explícito para el problema mixto para la ecuación del calor
- Inicialización para la ecuación del calor.
- Resultados de la ecuación del calor.
- Método para la ecuación de Poisson.
- Inicialización.
- Variables correspondientes a los puntos del borde.
- Esquema de diferencias finitas.

Método explícito para el problema mixto para la ecuación de ondas

En esta sección vamos a implementar en Matlab/Octave el método explicado en teoría. Debemos tener en cuenta que aunque matemáticamente se pueda definir un subíndice de una variable como cero, los elementos de las matrices (y los índices en Matlab/Octave) empiezan en 1.

Como ejemplo, nos planteamos el problema

(parcial segunda de u respecto de t dos veces) - 4*(parcial segunda de u respecto de x dos veces) = 0 para t>=0, x en [0,6],

u(0,x)=x*(x-6)^2,

parcial de u respecto de t en (0,x)=0, para x en [0,6],

u(t,0)=u(t,2)=0, para t>=0. donde discretizamos en el dominio [0,2] x [0,6] de forma que en el dominio del tiempo haya 2 subintervalos y en el dominio de x haya 3 subintervalos.

```
% Comenzamos definiendo la función phi que define la deformación inicial de la cuerda.
% La definimos de forma que se pueda usar en vectores.
phi=@(x) x.*(x-6).^2;
```

La solución del problema será una función u(t,x) donde la variable t está en el intervalo [0,T] dividido en N intervalos y la variable x está en el intervalo [0,ele] dividido en M intervalos.

```
T=2;
ele=6;
N=2;
M=3;
```

Como el intervalo [0,T] está dividido en N subintervalos, cada uno de estos subintervalos tendrá longitud Deltat=T/N. De la misma manera, como el intervalo [0,ele] está dividido en M subintervalos, cada uno tendrá longitud Deltax=ele/M. Con esto podemos definir los puntos que utilizamos para dividir los intervalos (y que dan la 'red uniforme' de la que se habla en la teoría).

```
Deltat=T/N;
Deltax=ele/M;
nodost=[0:Deltat:T];
nodosx=[0:Deltax:ele];
```

Definimos el coeficiente c a partir de la ecuación en derivadas parciales y el valor de q que se usará en las ecuaciones de diferencias finitas.

```
c=2; % c^2 es el coeficiente de la parcial segunda de u respecto de x dos veces.
q=c*Deltat/Deltax;
```

Inicializamos una matriz en la que almacenaremos las aproximaciones de u en los puntos de la red uniforme.

```
U=zeros(N+1,M+1);
```

Condiciones iniciales para la ecuación de ondas.

Definimos la deformación inicial de la cuerda. U(1,j)=phi(Deltax*(j-1)) . Ahorramos un bucle aplicando phi a los nodos en el eje x.

```
U(1,:)=phi(nodosx);
% Ahora imponemos que la cuerda está sujeta en el extremo izquierdo x=0,
% que es equivalente a definir en un bucle U[i,1]=0 para i=1,2,...,N+1 .
U(:,1)=0;
% También la cuerda está sujeta en el extremo derecho x=ele ,
% que es equivalente a definir en un bucle U[i,M+1]=0 para i=1,2,...,N+1 .
U(:,M+1)=0;
```

Primeros valores de U.

Aquí usamos diferencias finitas para calcular U en el siguiente instante tras la deformación inicial de la cuerda. En la teoría estos valores se

notaban con subíndices U1,j con i=1, pero aquí usamos la notación U(2,j). No llegamos hasta U(2,M+1) porque ya sabemos que vale cero.

```
for j=2:M
    U(2,j)=(1-q^2)*U(1,j)+(q^2/2)*( U(1,j+1)+U(1,j-1) );
end
```

Bucle general para la ecuación de ondas.

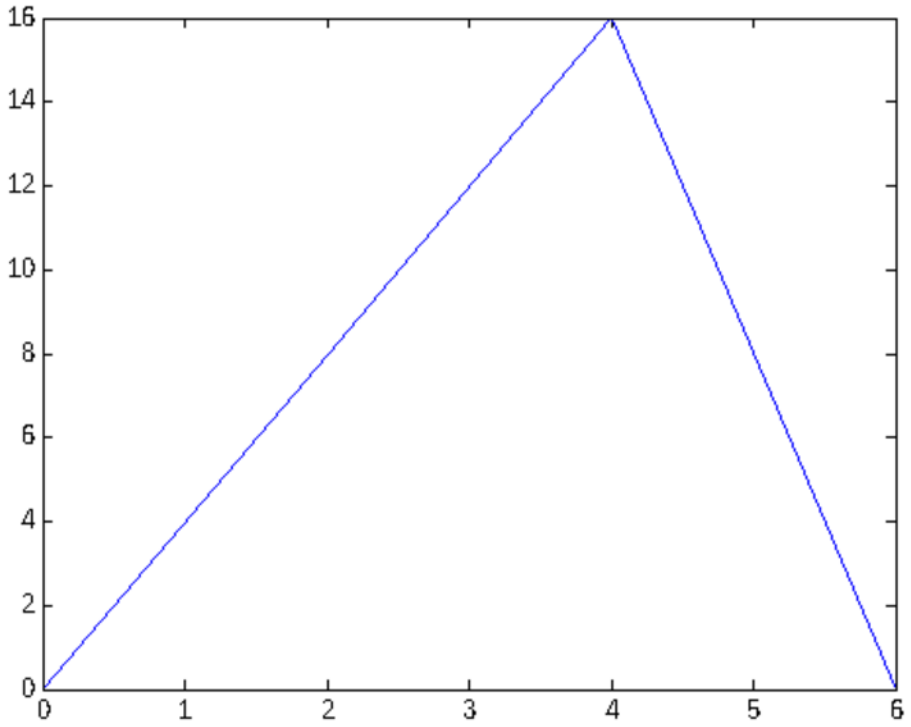
Ahora usamos el esquema de diferencias finitas visto en teoría para el resto de puntos de la red uniforme. Empezamos calculando U(3,j), porque el U(2,j) ya se calculó en el anterior bucle.

```
for i=2:N
 for j=2:M
 U(i+1,j)=2*(1-q^2)*U(i,j)+q^2*( U(i,j+1)+U(i,j-1) )-U(i-1,j) ;
 end
end
```

Resultados para la ecuación de ondas.

La matriz U contiene los valores aproximados de la función u.

```
U
plot(nodosx, U(1,:) ) % Gráfica de la posición inicial i=1, t=0 .
disp('Pulse una tecla para ver la siguiente gráfica');
pause
plot(nodosx, U(2,:) )  %Gráfica del momento i=2, t=Deltax .
% Representar U.
% stem3(nodost,nodosx,U')


U =

     0    32    16     0
     0     8    16     0
     0   -16    -8     0

Pulse una tecla para ver la siguiente gráfica
```



Ej. Se quiere hallar una aproximación a la solución del siguiente problema mixto para la ecuación de ondas

(parcial segunda de u respecto de t dos veces) - (parcial segunda de u respecto de x dos veces) = 0 para t>=0, x en [0,2],

u(0,x)=sen(pi*x/2),

parcial de u respecto de t en (0,x)=0, para x en [0,2],

u(t,0)=u(t,2)=0, para t>=0.

Aplique el método explícito en diferencias finitas, dividiendo el dominio [0, 6] x [0, 2] en intervalos de longitud 2 en la variable t e intervalos de longitud 1 en el dominio de la variable x . (Sol. U=[0 1 0 ; 0 -3 0; 0 17 0; 0 -99 0])

Método explícito para el problema mixto para la ecuación del calor

Es este apartado nos planteamos hallar la solución aproximada de la ecuación del calor. Usaremos la misma notación y la misma discretización del dominio que en la ecuación anterior. El cambio en la ecuación en derivadas parciales afectará a las ecuaciones en diferencias finitas.

Nos planteamos el problema (parcial de u respecto a t) - (parcial segunda

de u respecto a x dos veces) = 0, para t>=0, x en [0,3],

u(0,x)=sen(pi*x/3), para x en [0,3],

u(t,0)=u(t,3)=0, t>=0, donde consideramos el dominio del tiempo [0,9] dividido en 3 subintervalos y el dominio [0,3] para la variable x, dividido en 3 intervalos.

Inicialización para la ecuación del calor.

Comenzamos definiendo la función phi que es la temperatura inicial de la barra, y discretizamos en los ejes del tiempo y del espacio.

```
phi=@(x) sin(pi*x/3);
T=9;
ele=3;
N=3;
M=3;
Deltat=T/N;
Deltax=ele/M;
nodost=[0:Deltat:T];
nodosx=[0:Deltax:ele];
```

Definimos el coeficiente de la ecuación en derivadas parciales y una matriz que contendrá las aproximaciones a los valores de u.

```
c=1; % Coeficiente de la parcial segunda de u respecto de x dos veces.
q=c*Deltat/(Deltax^2);
U=zeros(N+1,M+1);
```

Imponemos las condiciones de contorno.

```
U(1,:)=phi(nodosx);
U(:,1)=0;  %U(i,1)=0 para i=1,2,...,N+1 .
U(:,M+1)=0;  %U(i,M+1)=0 para i=1,2,...,N+1 .
```

Ahora escribimos el bucle general usando el esquema en diferencias finitas. En este caso tenemos que empezar por i=1, para calcular U(2,j). Esto cambia con respecto al caso anterior.

```
for i=1:N
 for j=2:M
 U(i+1,j)=(1-2*q)*U(i,j)+q*( U(i,j+1)+U(i,j-1) ) ;
 end
end
```

Resultados de la ecuación del calor.

Para comprobar los resultados recordamos que -sqrt(3) = -1.7321, 2*sqrt(3) = 3.4641, -4*sqrt(3)= -6.9282 .

```
U
```

```
U =

         0    0.8660    0.8660         0
         0   -1.7321   -1.7321         0
         0    3.4641    3.4641         0
         0   -6.9282   -6.9282         0
```

Ej. Considere el siguiente problema mixto para la ecuación del calor (parcial de u respecto a t) - 2*(parcial segunda de u respecto a x dos veces) = 0, para t>=0, x en [0,4],

u(0,x)=x*(4-x), para x en [0,4],

u(t,0)=u(t,4)=0, t>=0.

Halle una aproximación a la solución usando el método explícito en diferencias finitas en el dominio [0,4] x [0,4], dividiendo en intervalos de longitud 2 en el eje de variable t y en intervalos de longitud 1 en el eje de la variable x . (Sol. U=[0 3 4 3 0 ; 0 -5 -4 -5 0 ; 0 19 -12 19 0])

Método para la ecuación de Poisson.

En esta sección nos planteamos la ecuación de Poisson (laplaciano de u igual a una función dada f). Al implementar este método, tendremos que resolver un sistema de ecuaciones lineales para obtener las aproximaciones de la función U. Al plantear el sistema lineal, el vector término independiente contiene valores de f(xi,yj) y el vector de soluciones contiene valores de u. Será necesario reordenar los elementos de una matriz para que tengan forma de vector.

Nos planteamos el problema de aproximar una función u(x,y) definida en un dominio D=[0,3]x[0,3] de forma que

su laplaciano cumple (parcial segunda de u respecto a x dos veces) + (parcial segunda de u respecto a y dos veces) = 6(x+y), para (x,y) en D.

u(x,y)= x^3+y^3, para (x,y) en la frontera de D. Para el método consideramos el dominio de la x dividido en tres subintervalos y el dominio de la y dividido en tres subintervalos.

Inicialización.

Extremos del intervalo para el eje x y subintervalos.

```
a=0;
b=3;
N=3;
```

```
Deltax=(b-a)/N;
nodosx=[a:Deltax:b];
```

Extremos del intervalo para el eje y y subintervalos.

```
c=0;
d=3;
M=3;
Deltay=(d-c)/M;
nodosy=[c:Deltay:d];

% Definición de la función en la frontera.
phi=@(x,y)  x^3+y^3;

% Definición de la función f igual al laplaciano.
f=@(x,y) 6*(x+y);

% Inicializamos U
U=zeros(N+1,M+1);

% Inicializamos A y B para el sistema de ecuaciones.
A=zeros( (N+1)*(M+1), (N+1)*(M+1) );
B=zeros( (N+1)*(M+1), 1 );
```

La función q transforma los valores i=1,..,N+1, j=1,...,M+1 en valores de q=1,...,(N+1)*(M+1).

```
q=@(i,j) (j-1)*(N+1)+i;
```

Variables correspondientes a los puntos del borde.

Usamos la función phi de las condiciones de contorno para definir las ecuaciones correspondientes a los valores de u en el borde.

```
for j=1:M+1
    % U(1,j)=phi(a, nodosy(j) );
    A( q(1,j),q(1,j) )=1;
    B( q(1,j) )=phi(a, nodosy(j) );

    % U(N+1,j)=phi(b, nodosy(j) );
    A( q(N+1,j),q(N+1,j) )=1;
    B( q(N+1,j) )=phi(b, nodosy(j) );
end

for i=1:N+1
    %U(i,1)=phi( nodosx(i), c);
    A( q(i,1),q(i,1) )=1;
    B( q(i,1) )=phi( nodosx(i), c);

    %U(i,M+1)=phi( nodosx(i), d);
    A( q(i,M+1),q(i,M+1) )=1;
    B( q(i,M+1) )=phi( nodosx(i), d);
end
```

Esquema de diferencias finitas.

Montamos el sistema de ecuaciones que nos permite hallar los valores de U. Estas ecuaciones se deducen al aplicar diferencias finitas a los puntos interiores del dominio.

Queremos escribir la siguiente ecuación en forma matricial (1/Deltax^2)*U(i+1,j) - (2/Deltax^2)*U(i,j) + (1/Deltax^2)*U(i-1,j) + + (1/Deltay^2)*U(i,j+1)-(2/Deltay^2)*U(i,j)+(1/Deltay^2)*U(i,j-1) = f(nodosx(i), nodosy(j)) Cada valor de (i,j) nos da una ecuación distinta, que numeramos como ec=q(i,j) con término independiente f(nodosx(i), nodosy(j)) y con coeficicientes (1/Deltax^2), ... que introducimos en la matriz A.

```
for i=2:N
    for j=2:M
    % Estamos en la ecuación con término independiente
    % f( nodosx( i ), nodosy( j ) )
    ec = q(i,j); %Número de la ecuación.
    %
    A( ec, q(i+1,j) ) = (1/Deltax^2);
    A( ec, q(i,j) )      = -(2/Deltax^2) -(2/Deltay^2);
    A( ec, q(i-1,j) )  =  (1/Deltax^2);
    %
    A( ec, q(i,j+1) ) = (1/Deltay^2);
    A( ec, q(i,j-1) )  = (1/Deltay^2);
    %
    B( ec ) = f( nodosx( i ), nodosy( j ) );
    end
end
```

Resolvemos el sistema con la 'división izquierda'.

```
X=A\B


X =

         0
    1.0000
    8.0000
   27.0000
    1.0000
    2.0000
    9.0000
   28.0000
    8.0000
    9.0000
   16.0000
   35.0000
   27.0000
   28.0000
   35.0000
   54.0000
```

El vector X tiene (N+1)*(M+1) componentes que son aproximaciones de los valores de U en los nodos. Recuperamos la matriz U.

```
for i=1:N+1
    for j=1:M+1
        U(i,j)=X( q(i,j) );
    end
end

U
```

```
U =

         0    1.0000    8.0000   27.0000
    1.0000    2.0000    9.0000   28.0000
    8.0000    9.0000   16.0000   35.0000
   27.0000   28.0000   35.0000   54.0000
```

Ej. Aplique el método en diferencias finitas para hallar una aproximación a la solución del siguiente problema de contorno para la ecuación de Laplace en el dominio D formado por los puntos (x,y) que cumplen 0<x<4, 0<y<2 .

(parcial segunda de u respecto a x dos veces) + (parcial segunda de u respecto a y dos veces) = 0, para (x,y) en D.

u(x,y)=y^2-x^2, para (x,y) en la frontera de D.

Para el método, divida el dominio en ambas variables en subintervalos de longitud 1. Compare los valores aproximados obtenidos con los de la función u(x,y)=y^2-x^2. (Sol. U = [0 1 4 ; -1 0 3 ; -4 -3 0 ; -9 -8 -5 ; -16 -15 -12] = valores_exactos).

```
% Ecuaciones Diferenciales y Cálculo Numérico. (A. Palomares)
% Grado en Ingeniería de Tecnologías de Telecomunicación.
disp(date)
```

```
20-Apr-2023
```

Published with MATLAB® 7.5

Práctica 3. Resolución de ecuaciones (I)

Muchos problemas de las ciencias y las ingenierías requieren hallar una solución de una ecuación, o equivalentemente, una raíz de una función.

Así pues, dada una función f, nos plantearemos encontrar valores x que hagan f(x)=0. En esta práctica veremos métodos para hallar con Matlab las raíces de una función. En concreto trataremos los métodos de bisección, regula-falsi, secante, los métodos de iteración funcional y el de Newton-Raphson.

Contents

- Funciones en archivos .m
- Representación del eje OX
- Funciones anónimas
- Localización gráfica de raíces.
- Bisección
- Regula-Falsi

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 el eje OX.

Funciones en archivos .m

Pueden definirse funciones en Matlab mediante archivos .m cuya primera linea comienza con `function`. Una linea como la siguiente 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 devuelve 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.

Es posible definir una función con varias salidas:

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

Observe en el siguiente ejemplo una 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;
```

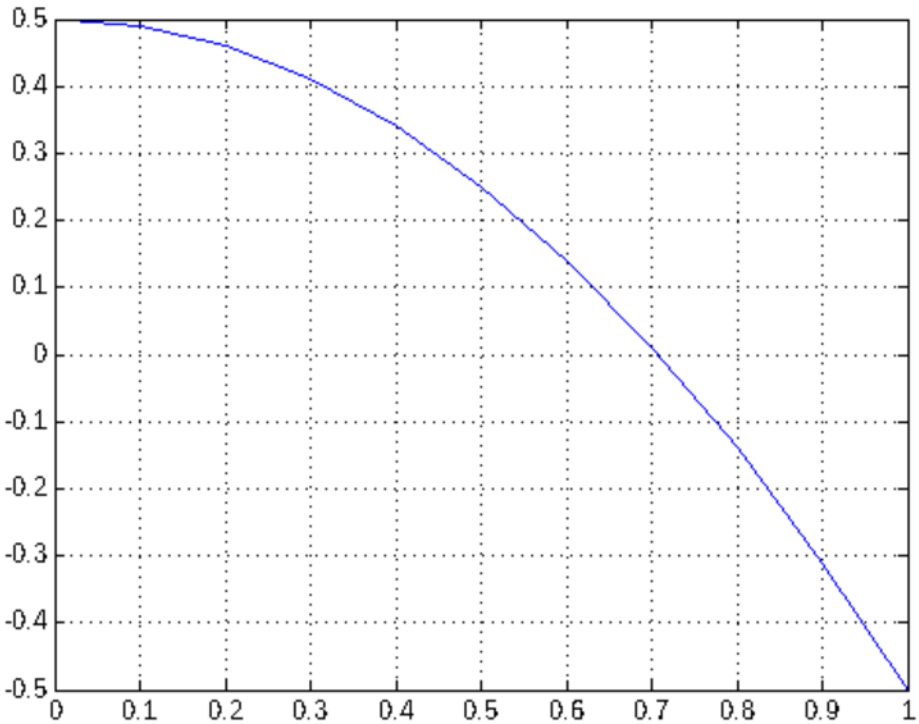
Ej 1. Defina un vector x con valores desde -1 a 1 con paso 0.1. Compruebe que la función f.m puede aplicarse al vector x. (Sol. f(x)=[-0.5, -0.31, -0.14, 0.01, ...])

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.

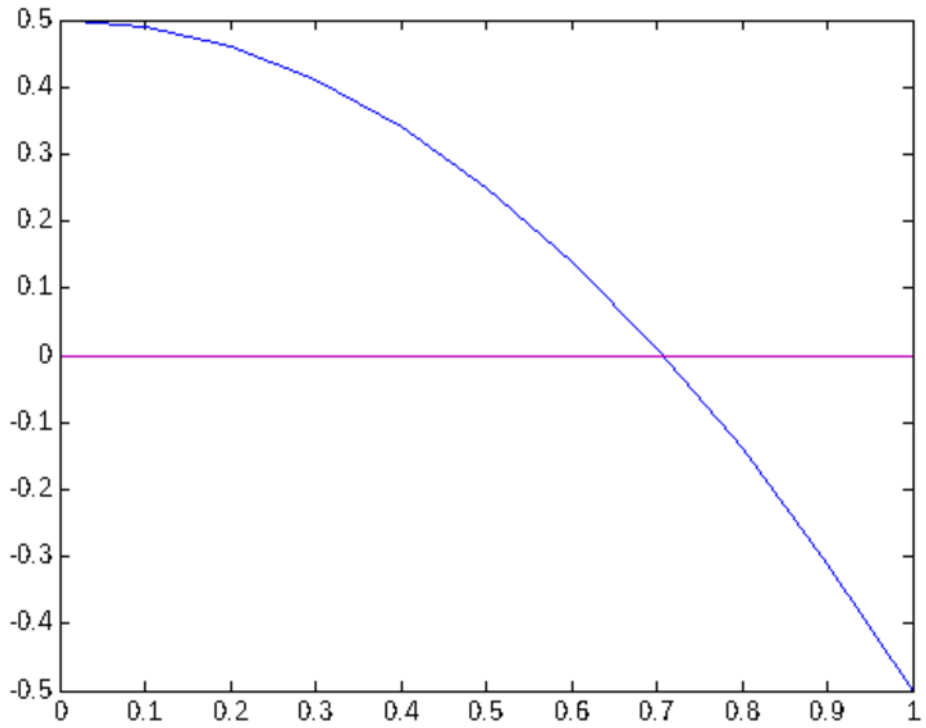
```
x=0:0.1:1;
plot(x,f(x))
grid on
```



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

Como vimos, 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 encontrar 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 puntos en los que h1(x)=h2(x), esto es, donde se cortan las gráficas de h1 y h2.

Bisección

Vamos a implementar el método de bisección sin necesidad de almacenar en memoria todas las iteraciones. En esta implementación los extremos del intervalor que contiene la raíz son los valores de a y b. El punto medio del intervalo lo llamaremos c.

Mostramos el pseudocódigo que permite implementar el método de bisección

```
1. Entrar f , a, b, tol
2. Mientras b-a>=tol
3. Hacer c = a + (1/2)(b-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
```

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=0.7;
b=0.8;
tol=0.0001;
while b-a>=tol
    % disp([a b]);
    c=a+(b-a)/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 raíces de la ecuación sin(x)-exp(x^2)+9/10=0. (Sol. aprox. x=0.1132, x=0.6317)

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 en los extremos del intervalo. El punto que divide los intervalos es el corte de la recta hallada con el eje OX. En la implementación almacenamos este punto en la variable c.

Vemos el pseudocódigo del método de regula-falsi

```
1. Entrar f , a, b, tol
2. Mientras b-a>=tol
3. Hacer c = b - (b-a)*f(b) / (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 `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.)

```
% Ecuaciones Diferenciales y Cálculo Numérico. (A. Palomares)
% Grado en Ingeniería de Tecnologías de Telecomunicación.
disp(date)
```

```
11-Apr-2023
```

Published with MATLAB® 7.5

Práctica 4. Resolución de ecuaciones (II)

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

Contents

- Método de la secante
- El comando fprintf
- Iteración funcional
- Método de Newton-Raphson
- Derivación simbólica

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 esos dos valores de la función. La siguiente iteración es el corte con el eje OX de la recta.

El siguiente pseudocódigo permite implementar el método de la secante:

```
1. Entrar f , a, b, tol
2. Mientras |b - a| >= tol
3. Hacer c = b - (b-a)*f(b)/(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 - (b-a)*f(b)/(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-009
```

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+000 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 usando notación exponencial.

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 (con seis dígitos tras el punto decimal).

```
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+000 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.

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-001, g(x1)= 9.146533e-001
N = 2, x1 = 9.146533e-001, g(x1)= 6.100653e-001
N = 3, x1 = 6.100653e-001, g(x1)= 8.196106e-001
N = 4, x1 = 8.196106e-001, g(x1)= 6.825059e-001
N = 5, x1 = 6.825059e-001, g(x1)= 7.759946e-001
N = 6, x1 = 7.759946e-001, g(x1)= 7.137247e-001
N = 7, x1 = 7.137247e-001, g(x1)= 7.559287e-001
N = 8, x1 = 7.559287e-001, g(x1)= 7.276348e-001
N = 9, x1 = 7.276348e-001, g(x1)= 7.467496e-001
N = 10, x1 = 7.467496e-001, g(x1)= 7.339006e-001
N = 11, x1 = 7.339006e-001, g(x1)= 7.425676e-001
N = 12, x1 = 7.425676e-001, g(x1)= 7.367349e-001
N = 13, x1 = 7.367349e-001, g(x1)= 7.406663e-001
N = 14, x1 = 7.406663e-001, g(x1)= 7.380191e-001
N = 15, x1 = 7.380191e-001, g(x1)= 7.398028e-001
N = 16, x1 = 7.398028e-001, g(x1)= 7.386015e-001
N = 17, x1 = 7.386015e-001, g(x1)= 7.394108e-001
N = 18, x1 = 7.394108e-001, g(x1)= 7.388657e-001
N = 19, x1 = 7.388657e-001, g(x1)= 7.392329e-001
N = 20, x1 = 7.392329e-001, g(x1)= 7.389856e-001
N = 21, x1 = 7.389856e-001, g(x1)= 7.391522e-001
N = 22, x1 = 7.391522e-001, g(x1)= 7.390400e-001
N = 23, x1 = 7.390400e-001, g(x1)= 7.391156e-001
Fin del bucle: x0 = 7.390400e-001, g(x0)= 7.391156e-001
```

Ej 6. 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. (Sol. aprox. x=7.390367e-001)

Ej 7. 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 8. Modifique el programa para que muestre un mensaje explicativo según el criterio de parada que se verifique.

Ej 9. 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.

Método de Newton-Raphson

Mostramos un 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 10. 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 sinh(x), seno hiperbólico) ¿Se consigue aproximar dicha raíz tomando como aproximación inicial x0=4? ¿y x0=4.1? (Sol. Con Matlab R2007b, si x0=4, se aproxima otra raíz, y si x0=4.1 se obtiene la aproximación x1=4.730041)

Ej 11. 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 12. 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 se pueden hallar derivadas simbólicas mediante el comando diff. En Octave se requiere instalar un paquete externo. En Wolfram|Alpha se pueden hacer derivadas simbólicas mediante D, por ejemplo D[Sin[x], x] . 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*cos(x^2)*x


ans =

2*sin(x)*cos(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.

Práctica 4. Resolución de ecuaciones (II)

Ej 13. 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).

```
% Ecuaciones Diferenciales y Cálculo Numérico. (A. Palomares)
% Grado en Ingeniería de Tecnologías de Telecomunicación.
disp(date)
```

```
20-Apr-2023
```

Published with MATLAB® 7.5